

Supplement for OSLC Workshop

Tracked Resource Set Reference Application

Lab Exercises

Contents

LAB 0	GETTING ADDITIONAL SETUP	6
	0.1 SET UP THE LAB ENVIRONMENT	6
LAB 1	ENABLING TRACKED RESOURCE SET PROVIDER	7
	1.1 UNDERSTAND AND EXPLORE THE LYO TRS TOOLKIT	7
	1.2 EXAMINE HOW THE TRS TOOLKIT HELPS CREATE A TRS PROVIDER VIA JAX-RS	9
	1.3 EXAMINE HOW THE TRS TOOLKIT HELPS CREATE A TRS PROVIDER VIA GENERIC SERVLET	11
	1.4 CHANGE LOG CONSIDERATIONS	12
	1.5 RUN THE TRS COMPLIANCE TEST SUITE TO VALIDATE THE TRS REFERENCE APPLICATION'S TRS PROVIDER	15
	1.6 SUMMARY	15
LAB 2	INTEGRATING TRS PROVIDER WITH LIFECYCLE QUERY ENGINE (LQE) (OPTIONAL)	16
	2.1 ADD TRS PROVIDER TO LQE AS DATA SOURCE	16
	2.2 SUMMARY	20
APPENDIX A.	TROUBLESHOOTING	21
APPENDIX B.	SOURCES	22
APPENDIX C.	LEGAL	23
	1. NOTICES	23
	2. ABOUT THIS CONTENT	23

Overview

Objectives:

By working through the material for the TRS Reference Application, you will learn how to perform the following tasks:

- Use the TRS toolkit to create a TRS provider using JAX-RS or servlets
- Configure and use the TRS compliance test suite to verify the validity of a TRS provider
- Integrate the TRS provider into the Lifecycle Query Engine in RELM
- Run a simple SPARQL query to view the data indexed from the TRS provider

While this reference application depends on certain technologies, the core TRS concepts demonstrated within may be applied to other languages, frameworks, and resource domains.

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.*

Length: 1.5 - 2 hours depending

Introduction

In this workshop, you'll examining the TRS Reference application which uses the Eclipse Lyo TRS toolkit. The reference application is written in Java and demonstrates the implementation of a TRS Provider using change requests as the resources that are being created, modified and deleted. There are two implementations. In one implementation, JAX-RS, an annotation based Java RESTful specification, and Apache Wink, a JAX-RS server side component, are used to define and serve up the TRS provider endpoints. In the other implementation, a generic web servlet approach is used.

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.

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 through an already-working implementation, so you can read and try this code by using the web pages provided by the TRS Reference Application and Java debugger if you need.

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 this scenario, Allison the tools administrator has been asked if data from a third party change management system can be integrated into RELM so that traceability can include data from this tool

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. It also shows how these classes can be used with regular servlet applications and provides a guided walkthrough to see how TRS providers update their data feed over time. The lab also describes advanced topics to consider in full implementations such as persistence and how to prune the change log over time.

Lab 2: Integrating TRS provider with Lifecycle Query Engine (LQE) (optional) – this lab provides steps needed for the integration with LQE, as well as executing a simple query to view data indexed by LQE.

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. Note that these steps have been completed for your convenience in the Lab machines.

http://wiki.eclipse.org/Lyo/TRSSDK#TRS_Reference_Application

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 as well as utilizing TRS toolkit in simple servlet applications

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 on 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

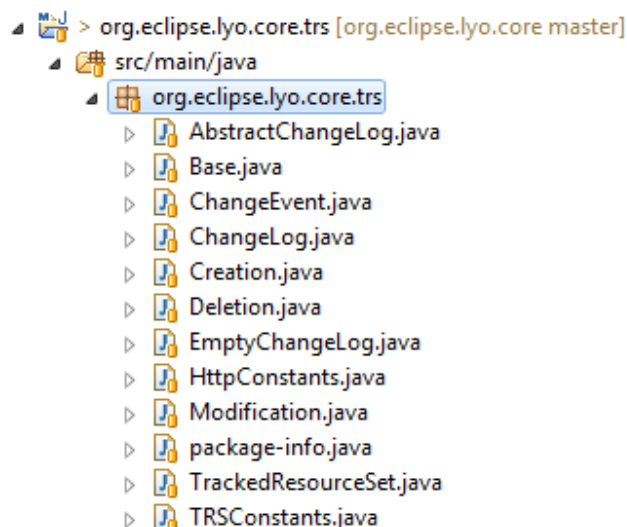
Lyo TRS Toolkit provides OSLC4J-annotated Java beans for TRS resources. To learn the toolkit, see:

<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 Exploring the TRS toolkit core classes

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 `change` and `previous` properties.

1.2 Examine how the TRS toolkit helps create a TRS provider via JAX-RS

1. Open the file **org.eclipse.lyo.rio.trs.resources.BaseResource.java**.

Notice `@Path("/trs/"+TRSConstants.TRS_TERM_BASE)` at the top of the class. This is a JAX-RS annotation that assigns a relative path (from the jax-rs servlet context) for the Base Resource RESTful endpoint. There are various constants and vocabularies defined in the toolkit to abstract the terms and namespaces from TRS provider implementers.

```

@Path("/trs/"+TRSConstants.TRS_TERM_BASE)
@OslcService(TRSConstants.TRS_BASE)
@Workspace(workspaceTitle = "Tracked Resource Set", collectionTitle = "Base")
public class BaseResource {
    @Context
    protected UriInfo uriInfo;
}

```

2. In the reference application, the Base Resource is paged so that, according to the specification, a **GET** on the base resources will do a redirect to the first page of base resources. This is what happens in the **getBase()** endpoint.

3. The other method, **getBasePage(page)**, returns the Base resource at the given page. The class **org.eclipse.lyo.core.trs.Base** (and **org.eclipse.lyo.core.trs.Page** in the 2.0 release) is provided by the TRS toolkit. It is basically a simple java bean that has been annotated with various OSLC4J annotations to allow the OSLC4J marshalling to create the proper RDF document for the JAX-RS configured serialization.
4. Both methods have additional annotations such as **@GET**, which identifies the type of HTTP traffic it can handle, as well as an **@Produces** that identifies the return type the method can handle. In all cases, the REST endpoints can return Turtle, RDF XML, XML, or JSON. JAX-RS dynamically returns the correct format based on the HTTP request that is received.
5. Open the file **org.eclipse.lyo.rio.trs.resources.TRSResource.java**
6. In similar fashion it also has an **@Path** annotation that identifies the root of the RESTful service for the Tracked Resource Set.
7. The reference application utilizes the segmented Changelog concept from the TRS Specification. The method **getTrackedResourceSet()** returns a tracked resource set with the most recent segment of the changelog.
8. The other method in this class, **getChangeLog(page)**, returns the segmented portion of the change as requested by the HTTP request

1.2.1 Trying it out

1. Ensure that the reference application is running. If it is not, in the Project Explorer view, in the **org.eclipse.lyo.rio.trs** project, right-click **Launch TRS Reference App.launch** and click **Run As -> Launch TRS Reference App**.
2. The TRS Reference Application includes an **index.html** file that contains links to the various capabilities provided by the application. To open the page, point your browser at <http://<hostname>:8082/org.eclipse.lyo.rio.trs> (e.g. <http://localhost:8082/org.eclipse.lyo.rio.trs>)
3. The first section of the page provides links to the Base and Tracked Resource Set as implemented by the **BaseResource** and **TRSResource** classes we observed earlier. Navigate to the Base Resources link. You'll see the Base Resources document in XML form. The format and syntax has been handled by the TRS toolkit's annotated beans and the OSLC4J serializers.
4. Navigate to the Tracked Resource Set link. The change log contained in the Tracked Resource Set should contain a **rdf:nil** to represent no changes have been made to the resources.
5. Return to the index page and click **Create Change Request**.

6. Type in a title and a description, and then click **Submit Bug**. Make note of the change request number.
7. Go back to the Tracked Resource Set page and refresh if necessary. Notice an entry in the change log section of the Tracked Resource Set containing a Creation event. If you repeat the creation event more than 3 times, you'll notice the change log being segmented with a previous element containing a URI to return the previous segment of the change log.

1.3 Examine how the TRS toolkit helps create a TRS provider via generic Servlet

1. Open the file **org.eclipse.lyo.rio.trs/src/main/webapp/WEB-INF/web.xml**. Notice the servlet mapping for the relative path **/restx/trs/base/**. This maps the Base resource RESTful endpoint to the servlet class **org.eclipse.lyo.rio.trs.servlet.BaseGeneric**.
2. Open the file **org.eclipse.lyo.rio.trs.servlet.BaseGeneric**. It has an overridden **doGet** method, which implements the root of the RESTful service for the Base Resource.
3. In the reference application, the Base Resource is paged so that, according to the specification, a get on the base resources will do a redirect to the first page of base resources. If the path for the Get request contains the page number, it returns the response corresponding to the Base resource at the given page. In a simple servlet implementation, we serialize the Base resource to the response stream using **OSLC4JMarshaller**. All implementations of the Base end point handles RDF XML and the 2.0 version adds support for text/turtle. The reference application can be extended to support various other media types based on the HTTP request.
4. Similarly, the file **web.xml** also contains the servlet mapping for two other endpoints for Tracked Resource Set and Change Log.
5. The servlet mapping for the relative path **/restx/trs/** maps the Tracked Resource Set RESTful endpoint to the servlet class **org.eclipse.lyo.rio.trs.servlet.TRSGeneric**.
6. Open the file **org.eclipse.lyo.rio.trs.servlet.TRSGeneric**. It has an overridden **doGet** method, which implements the root of the RESTful service for the Tracked Resource Set.
7. The reference application is utilizing the segmented Changelog concept from the TRS Specification. The method **toGet()** populates a **TrackedResourceSet** (a simple java bean provided by the toolkit for representing a tracked resource set) with the most recent segment of the changelog. It then serializes the TrackedResourceSet object to the response stream using **OSLC4JMarshaller**.

8. In a similar fashion, the Changelog endpoint, which is mapped to the relative path `/restx/trs/changelog/`, is implemented in the class `org.eclipse.lyo.rio.trs.servlet.ChangeLogGeneric.java`.

1.3.1 Trying it out

1. Ensure that the reference application is running. If it is not, in the Project Explorer view, in the `org.eclipse.lyo.rio.trs` project, right-click `Launch TRS Reference App.launch` and click `Run As -> Launch TRS Reference App`.
2. The TRS Reference Application includes an `index.html` file that contains links to the various capabilities provided by the application. To open the page, point your browser at `http://<hostname>:8082/org.eclipse.lyo.rio.trs` (e.g. `http://localhost:8082/org.eclipse.lyo.rio.trs`)
3. The first section of the page provides link "Generic Base Resources" to the Base and "Generic Tracked Resource Set" to Tracked Resource Set as implemented by the `BaseGeneric` and `TRSGeneric` classes we observed earlier. Navigate to the Generic Base Resources link. You'll see the Base Resources document in XML form.
4. Navigate to the **Generic Tracked Resource Set** link. The change log contained in the Tracked Resource Set should may contain entries if you tried it out already from 1.2. Otherwise an `rdf:nil` entry exists to represent no changes have been made to the resources.
5. Return to the index page and click **Create Change Request**.
6. Type in a title and a description, and then click **Submit Bug**. Make note of the change request number.
7. Go back to the Tracked Resource Set page and refresh if necessary. Notice an entry in the change log section of the Tracked Resource Set containing a Creation event. If you repeat the creation event more than 3 times, you'll notice the change log being segmented with a previous element containing a URI to return the previous segment of the change log.

1.4 Change Log Considerations

1.4.1 Generating Change Log Events

There are several ways to generate the change events that populate the change log. If accessing the resource server's code is possible then it may be easiest to simply alter the code where resources are created, modified, and deleted. The reference application does this in certain cases. For example, open the `ChangeRequestResource.java` file in the `org.eclipse.lyo.rio.trs.resources` package and browse to

the `deleteChangeRequest()` method. This method handles deleting a resource as well as updating the change log. The method call to `insertEventTypeToChangeLog` handles creating a deletion event and inserting it into the change log.

Another, perhaps cleaner way, to achieve the same result is to use the Java listener pattern. Open the **ChangeRequestListener.java** file in the **org.eclipse.lyo.rio.trs.cm** package. A new class can implement this `ChangeRequestListener` interface and override the `changeRequestAltered` method to be alerted to alterations of underlying resources. For example, open the **Persistence.java** class in the **org.eclipse.lyo.rio.trs** package and look at the `addListener()` method. The newly created listener can be registered with a method such as this. Then when alterations occur the listeners can be updated. Navigate to the `addChangeRequest()` method and notice that after several operations are performed to create a new resource and save it to disk that the `notifyListener` method is called. `notifyListener` will call all registered listeners and they can react how they wish to the resource alterations. In this case a change event could be created and inserted into the change log.

A final approach, which is often useful if direct access to the resource server's code is not available is a polling mechanism. The server with the TRS endpoint can periodically poll the resource server for changes and then generate change events based on what it discovers between polling intervals. It should be noted that this approach assumes the resource server has a query mechanism of some sort. Using a timestamp query parameter if available is recommended.

1.4.2 When to Generate Change Log Events

There are three types of change events: creation, modification, and deletion. Creation and deletion events are fairly straightforward, they correspond to when an underlying resource is created or deleted respectively. When to create modification events is not as clear. The server should not report unnecessary change events although it might happen, for example, if changes occur while the base is being computed. If a server knows that a resource will change twice, it can safely emit a single change event to cover both. If a server knows that it will be changing a set of resources, it can safely batch up the change events until it has changed all of them.

1.4.3 Persisting and Pruning the Change Log

The change log keeps track of recent modifications to change request resources. When a resource is created, modified, or deleted a change event is created. It is recommended that a resource server persist these events so that the change log may be restored if the resource server needs to be restarted. In order to keep the change log from growing without bounds, it should be pruned periodically.

Both persisting the change log, and pruning it, are considered best practices as they allow consumers to optimize updates. Instead of examining the entire base for details, consumers can processes a smaller set of data in the change log. Persisting the change log allows this optimization to take place even across server resets. Pruning the log allows a system administrator to control how far back in time consumers may read events in the change log before having to parse the base resource. A history of seven days is recommended as a starting point. However, the administrator should choose a prune time based on how frequently change events are created and how far back in time consumers typically need to search in order to stay in synch with the resource server.

Once both the Base and Change Log resources are initialized the `cutoffEvent` property should be set in the Base resource. The `cutoffEvent` property consists of a URL which points to the most recent change event that exists in both the Base and Change Log. Clients can examine this property to quickly determine what change events exist in the Change Log.

1.4.4 Example Implementation

1. Open the file **org.eclipse.lyo.rio.trs.util.TRSObject.java** file.
2. Navigate to the **loadChangeEvents** method and read the method comment.
3. The method starts by making a call to **FileUtil.fileload** which loads any existing change events from file into memory. It should be noted that in a production environment any calls to the **FileUtil** class would likely be replaced with calls to a true database.
4. The first for loop examines all change events and sorts them according to their order property. The second for loop makes calls to **isPrunningNecessary()** to determine if the current change event being processed should be removed due to its age. The reference application only performs pruning at server startup. In a production system, where a server may remaining for multiple days or weeks without reset, pruning should take place more frequently.
5. If events are pruned, the method saves the pruned list back to disk.
6. Navigate to the **initialize()** method and jump to the bottom of the method. Note the call to **setcutOffEventInner** and its corresponding comment. This call updates the `cutoffEvent` property of the base resource, allowing existing clients a mechanism for quickly determing what data the change log contains. The server is now initialized.
7. After initialization the reference application continues to maintain the change log. Navigate to the **insertEventTypeToChangeLog** method in the **TRSResource.java** file.

8. This method is called whenever a new change event is created in order to insert it into the change log.
9. Note at the bottom of the file how the call to **FileUtil.save** persists the new change event so it is available if the server is restarted.

1.5 Run the TRS compliance test suite to validate the TRS Reference Application's TRS provider

1. In the Project Explorer view, expand the **org.eclipse.lyo.testsuite.trs** project and navigate to the `src/main/resources/config.properties` file. The `configTrsEndpoint` property specifies the location of the TRS resource to be validated by the compliance test suite. The tests also examine the base resource, which is discovered by examining the TRS' `trs:base` property.
2. Edit the values to point to the location of the TRS Reference Application that you are running.
3. In the **org.eclipse.lyo.testsuite.trs** project, right-click `Launch All JUnits.launch` and click **Run As -> Launch All JUnits**.

The TRS compliance JUnit test suite runs and validates the contents of the Base Resources and the Tracked Resource Set of the reference app. The results of the JUnit test are displayed in the JUnit view.

It should be noted that additional tests exist in the Rio repository. These can be examined by loading the **org.eclipse.lyo.rio.trs.tests** project in the same way the **org.eclipse.lyo.rio.trs** was loaded. To run these tests, start the reference application and then navigate to **org.eclipse.lyo.rio.trs.tests/src/main/resources** and open the `config.properties` file. This file controls the content and headers that are sent to the resource server, which is the reference application by default. To run these tests against your own server, alter the contents of this file using the comments in the file for reference. A launch file exists in the **org.eclipse.lyo.rio.trs.tests** project to assist in running the tests.

1.6 Summary

This lab helped you explore the key classes contained in the TRS toolkit as well as trying out the TRS Reference Application to see how the change log is updated as new change requests are added over time. You also examined how these classes can also be used in generic servlet applications for situations where your existing application is not making use of JAX-RS or need fully capability from OSLC4J.

Lab 2 Integrating TRS provider with Lifecycle Query Engine (LQE) (optional)

Objectives

- Understand how TRS provider validates the authentication without user interaction
- Understand how TRS provider can be integrated with Lifecycle Query Engine (LQE)
- Understand how LQE uses TRS resources as a consumer of TRS provider
- 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 it to a TRS client. (Note that in this workshop, Lifecycle Query Engine (LQE) is used as a TRS client.)

This lab will show you how to configure LQE, which is a TRS client to retrieve data from the TRS provider in Lab 1 to start the indexing of the resources. You will then write a small SPARQL query to test that LQE contains the data from your TRS provider.

2.1 Add TRS provider to LQE as Data Source



The TRS Reference Application doesn't perform any authentication. If you use OAuth, you should review the TRS4JBugzilla application for extra steps required before adding your provider as a data source (creating friend and consumer keys).

The next step is to add TRS provider to LQE as its Data Source in order for LQE to start indexing. You will also query data by using SPARQL.

2.1.1 Steps

- ___1. In Browser, open LQE Data Sources page (<https://trsserver:9443/lqe/web/admin/data-sources>)
Note: "trsserver" is the host name where LQE runs.) and login it with the admin user id and its password. From this page, you are going to add TRS provider as LQE Data Source.
 - ___a. Click **Add Data Source** button at the right top
 - ___b. Fill the dialog as follow:

- __i. Data Source: Select **Data Source URL**
- __ii. Put <http://trsserver:8082/org.eclipse.lyo.rio.trs/rest/trs> into the edit box. (Note: “trsserver” is the host name where Tracked Resource Set Provider runs.)
- __iii. Label: **TRSRefApp**
- __iv. Authentication: **No authentication is required for this data source**

Add Data Source

1 Select Data Source 2 3

Property	Value
Data Source: *	<p>Lifecycle Query Engine has found the following data sources. Please select a data source. Alternatively, you can provide the URL for either the root services document or the data source.</p> <p><input type="radio"/> https://jazzreportingservices.ibm.com:9443/jts/trsUsers</p> <p><input type="radio"/> Root Services URL</p> <p><input checked="" type="radio"/> Data Source URL</p> <p><input type="text" value="http://trsserver:8082/org.eclipse.lyo.rio.trs/rest/trs"/></p> <p>Enter the URL of a Tracked Resource Set</p>
Label: *	<p><input type="text" value="TRSRefApp"/></p> <p>Enter a name to identify this data source</p>
Scheduling:	<p><input type="text"/> <input type="text"/></p> <p>The first time indexing of this data source will begin at the selected date and time in the local timezone. If left empty indexing will start immediately.</p>
Authentication:	<p><input type="radio"/> Specify OAuth authentication details for this data source</p> <p><input checked="" type="radio"/> No authentication is required for this data source</p>

< Back Next > Finish Cancel

- __c. Click **Next**

__i. In the **Refresh Rate** field, type "10 seconds" and click **Finish**. LQE begins indexing the base resources and, after 10 seconds, it indexes the change log.

Property	Value
Refresh Rate: * (seconds)	10 <small>Data source will be refreshed every specified number of seconds.</small>
Number of threads for first time indexing:	2 <small>Lifecycle Query Engine makes use of the specified number of threads for first time indexing. Selection of 2 to 4 threads is recommended.</small>
Number of threads for incremental indexing:	2 <small>Lifecycle Query Engine makes use of the specified number of threads for incremental indexing. Selection of 2 to 4 threads is recommended.</small>
Continue with incremental update in case of skipped resources:	<input checked="" type="checkbox"/> <small>While adding a data source, Lifecycle Query Engine may skip some resources if it encounters HTTP errors or timeouts. If checked, Lifecycle Query Engine will continue with incremental updates despite skipped resources. Otherwise, an administrator will need to review the skipped resources before continuing.</small>

< Back Next > Finish Cancel

2.1.2 Time for the final tests for LQE integration

__1. Wait for the index to be completed.

Data Source	Status
RTC https://trserver:9443/ccm/oslc/workitems/trs	✓ Up-to-date Last updated at 5:08 AM.
TRS4JBugzilla http://trserver:8085/OSLC4JBugzilla/services/trs	✓ Added data source successfully Last updated at 5:08 AM. Processed 11 resources. Duration: 12 seconds.

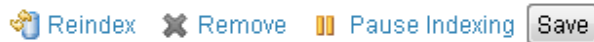
If you will see a red X icon like :

Data Source	Status	Actions
RTC https://trserver:9443/ccm/oslc/workitems/trs	✓ Up-to-date Last updated at 2:43 AM.	
TRS4JBugzilla http://trserver:8085/OSLC4JBugzilla/services/trs	✗ Update data source failed At 2:43 AM. Duration: 1 seconds. 1 error. Action may be required	

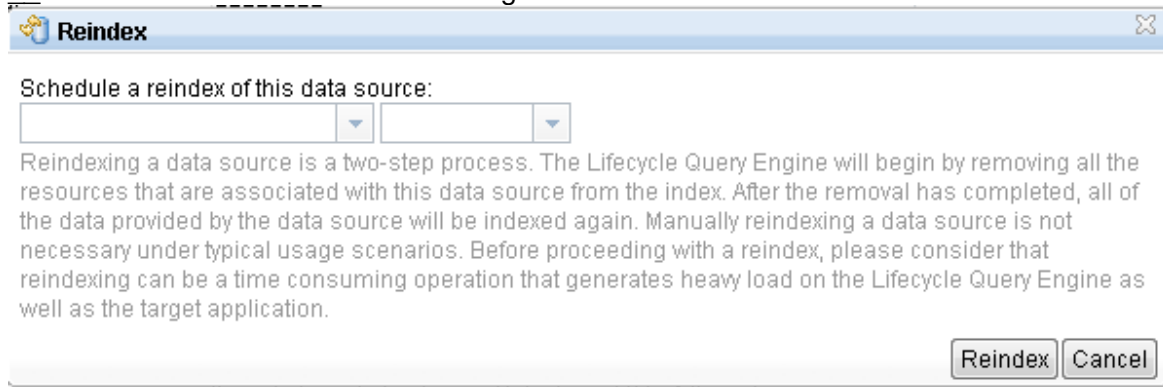
please try to re-index the resources by the following steps.

__a. Click a **pencil** icon.

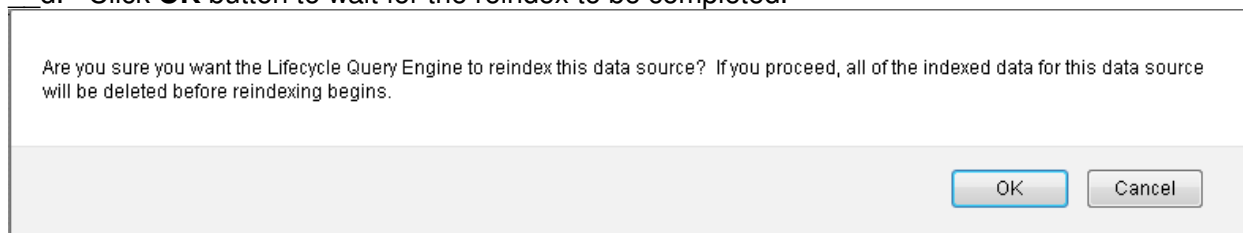
__b. Click **Reindex** link.



__c. Click **Reindex** button in the dialog.



__d. Click **OK** button to wait for the reindex to be completed.



__2. Click **Query** link near the top navigation bar. Ensure you are view the Query sub tab and not the text search.

__3. Type the following SPARQL in the text field. (You can copy it from “SampleSPARQL.txt” file on the desktop) . This query will search for one change request from the TRS Reference Application and display all the subject, predicate and object triples for that resource.

```
SELECT ?s ?p ?o
WHERE
{ GRAPH <http://localhost:8082/org.eclipse.lyo.rio.trs/rest/changeRequests/1> {
  ?s ?p ?o
}
}
```

Note: The details about SPARQL are available at <http://www.w3.org/TR/sparql11-query/>.

___4. Click **Run** button

___5. You will see some resources in the result screen (only showing the **p** and **o** columns due to space constraints).

p	o
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://open-services.net/ns/cm#ChangeRequest
http://purl.org/dc/terms/description	Unable to execute Apache Tomcat due to missing Java runtime environment (JRE). <http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral>
http://purl.org/dc/terms/title	Apache Tomcat requires JRE <http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral>
http://open-services.net/ns/core#shortTitle	Apache Tomcat requires JRE <http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral>
http://purl.org/dc/terms/modified	2014-04-17T06:39:14.631Z <http://www.w3.org/2001/XMLSchema#dateTime>
http://open-services.net/ns/cm#approved	false <http://www.w3.org/2001/XMLSchema#boolean>
http://open-services.net/ns/cm#inprogress	false <http://www.w3.org/2001/XMLSchema#boolean>
http://open-services.net/ns/cm#verified	false <http://www.w3.org/2001/XMLSchema#boolean>
http://open-services.net/ns/cm#fixed	false <http://www.w3.org/2001/XMLSchema#boolean>
http://open-services.net/ns/cm#status	Submitted
http://open-services.net/ns/cm#closed	false <http://www.w3.org/2001/XMLSchema#boolean>
http://open-services.net/ns/cm#reviewed	false <http://www.w3.org/2001/XMLSchema#boolean>
http://purl.org/dc/terms/type	Defect
http://purl.org/dc/terms/identifier	1
http://purl.org/dc/terms/created	2014-04-17T06:39:14.631Z <http://www.w3.org/2001/XMLSchema#dateTime>
http://open-services.net/ns/cm#severity	Unclassified

2.2 Summary

You learned how to connect the TRS Reference Application to the Lifecycle Query Engine to allow it to index the data. You also executed a simple query to verify the contents of the index. Feel free to attempt other SPARQL queries to perform more complex searches.

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, 2014.

2. About this Content

April 18, 2014

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
