

Innovate2010 Technical Workshop

OSLC Workshop

Consumer & Producer labs

Workshop Exercises



© Copyright International Business Machines Corporation, 2010. All rights reserved.

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

OVERVIEW	4	
		INTRODUCTION.....4
		ICONS.....4
LAB 1	SETTING UP FOR OSLC DEVELOPMENT	5
	1.1	DOWNLOAD AND UNZIP THE REQUIRED FILES FROM JAZZ.NET6
	1.2	SETUP A TOMCAT TEST SERVER.....8
	1.3	INSTALL THE REST CLIENT ADD-ON FOR FIREFOX.....9
	1.4	TEST THE REST CLIENT SETUP.....9
	1.5	TEST THE WTP AND TOMCAT SETUP11
	1.6	SETUP FOR THE REMAINING LABS21
LAB 2	AN INTRODUCTION TO THE OSLC APIS	22
	2.1	THE ROOT SERVICES DOCUMENT.....22
	2.2	THE REST CLIENT ADD-ON26
	2.3	OSLC SERVICES.....28
	2.4	SEARCH FOR SOME WORK ITEMS32
LAB 3	ACCESS OSLC APIS PROGRAMMATICALLY	35
	3.1	LOADING EXAMPLES.....35
	3.2	ACCESSING TO THE ROOT SERVICES DOCUMENT38
	3.3	RETRIEVE THE SERVICE PROVIDER CATALOG USING XPATH.....42
	3.4	JAZZ FORM-BASED AUTHENTICATION.....44
	3.5	WORK ITEM UPDATE.....49
LAB 4	IMPLEMENTING THE OSLC APIS IN A SERVICE PROVIDER	55
	4.1	LOADING EXAMPLES.....55
	4.2	SETTING UP THE SERVER RUNTIME ENVIRONMENT AND RUNNING THE SAMPLE SERVER.....58
	4.3	INTERACTING WITH THE SAMPLE PROVIDER.....60
	4.4	MODIFYING THE PROVIDER, ADDING ANOTHER DIALOG.....61
	4.5	TEST CHANGES64
APPENDIX A.	NOTICES	66
APPENDIX B.	TRADEMARKS AND COPYRIGHTS	68




Overview

Introduction

These labs will help guide you to leverage the Open Services for Lifecycle Collaboration (OSLC) standard interfaces for interoperating with IBM Rational Team Concert as well as other Jazz-based products. These labs will highlight key aspects by leveraging web browser access and programmatic access via Java client programs. The final lab will illustrate by an example how to write your own server using Java servlets. After you complete these labs, you will have a good foundation by which to leverage OSLC to implement in your interoperability project.

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 1 Setting up for OSLC Development



Lab Scenario

You have a new assignment on a team to integrate two Application Lifecycle Management (ALM) tools using OSLC. The first thing to do is to learn how to use OSLC services and how to provide your own OSLC services. This workshop will help you do just that.

Once you have completed this module, you will be ready to start developing OSLC integrations.

In order to complete and get the most out of this workshop, it is recommended that you are already familiar with RTC as a user. Of particular help would be familiarity with work items. In addition, you should have basic familiarity with Java programming and debugging using Eclipse. Note that OSLC can be used from any programming language that can invoke or provide web services and not just Java; however, the examples in this workshop are written in Java.



Note that these instructions are written specifically for RTC 2.0.0.2 on Windows. Please adjust accordingly for different operating systems (primarily the Eclipse client download and the file paths) and RTC versions (downloads).



Along with this lab document(s), you should have received or downloaded a file with a name of the form `ExtensionsAndIntegrationsLabCodeRepository-yyyymmdd.tar`. It is an exported RTC repository. You will import it at the end of this lab.



The repository will contain materials for other related workshops. Those other workshops also have initial labs like this one to set up the environment. You can run through more than one setup lab on the same RTC installation. Just note that some steps may be duplicated, such as the initial download steps or the final step to import the repository.

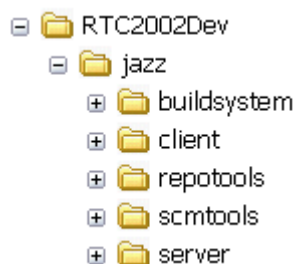
1.1 Download and Unzip the Required Files from jazz.net

- __1. Download the Express-C edition zip files.
 - __a. Go to the RTC 2.0.0.2 all downloads page at <https://jazz.net/downloads/rational-team-concert/releases/2.0.0.2?p=allDownloads>. There may be iFix releases available beyond 2.0.0.2. You can download one of those instead. The file sizes will vary from what is shown below.
 - __b. Scroll down to the **Express-C** section and download the highlighted file.


Express-C

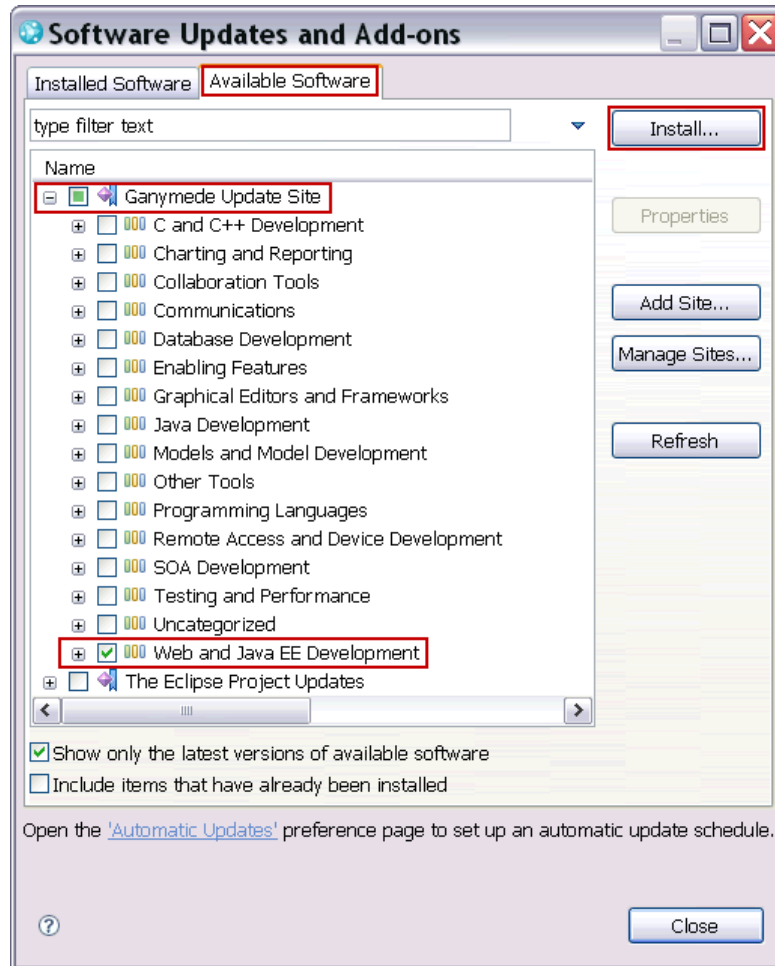
Description	Platform
ZIP	
Client for Eclipse IDE	Windows x86 (475.34 MB)
Client for Eclipse 3.5.x (p2 Install)	All (210.38 MB)
Client for Eclipse 3.5.x (Extension Install)	All (211.01 MB)
Client for Eclipse IDE and Server	Windows x86 (744.48 MB)
Server	Windows x86 (231.74 MB) Linux x86 (220.88 MB)
Build System Toolkit	Windows x86 (38.01 MB)
Plain Java Client Libraries	All (20.44 MB)

- __2. Unzip the product files. For this lab, our installation directory will be C:\RTC2002Dev.
 - __a. Unzip the Eclipse IDE and server download into C:\RTC2002Dev.
 - __b. Your C:\RTC2002Dev folder will look pretty standard at this point. Much like setting up a sandbox or demo environment.



- __3. Add the Eclipse Web Tools Platform (WTP) to the RTC client.
 - __a. Start the Eclipse client (C:\RTC2002Dev\jazz\client\eclipse\eclipse.exe).
 - __b. When prompted, select an Eclipse workspace. These instructions will use C:\RTC2002Dev\DevWS.

- ___c. Minimize the **Welcome** via this () button near the top of the window.
- ___d. From the menu bar, select **Help > Software Updates...**
- ___e. In the **Software Updates and Add-ons** dialog, switch to the **Available Software** tab, expand **Ganymede Update Site**, check the **Web and Java EE Development** entry and then click **Install...**



- ___f. Eclipse calculates the dependencies and shows you the features it will install. Click **Next** in the **Install** wizard.
- ___g. On the second page of the Install wizard, select **I accept the terms of the license agreements** (read them first if you wish) and then click **Finish**.
- ___h. When prompted to restart Eclipse, click **Yes**.
- ___i. Your C:\RTC2002Dev folder will still look the same. The WTP features and plug-ins were added to the features and plugins folders under C:\RTC2002Dev\jazz\client\eclipse.

1.2 Setup a Tomcat Test Server

- __1. This server will be used to run the OSLC provider sample. Download a Tomcat 5.5.28 Tomcat server from <http://tomcat.apache.org/download-55.cgi> or one of the mirrors.
- __2. Unzip the downloaded **apache-tomcat-5.5.28.zip** file. This workshop will assume the file is unzipped to the root of the C: drive. This will create the folder **C:\apache-tomcat-5.5.28** that contains the server.
- __3. Running Tomcat.
 - __a. If you have a J2SE 5 or higher JRE installed as your default JRE, Tomcat should be ready to run. From Windows explorer, simply double click the startup.bat and shutdown.bat files found in C:\apache-tomcat-5.5.28\bin.
 - __b. Alternatively, you can download a J2SE 5 or higher JRE (or JDK), unzip it and then set the JAVA_HOME variable before starting Tomcat. For example:
 - __i. Download a J2SE 5 JDK and unzip it to C:\J2SE5
 - __ii. Create a SetTomcatEnv.bat file with the single line:

```
set JAVA_HOME=C:\J2SE5
```
 - __iii. Open a command prompt and run this bat file and then start and stop Tomcat from that same command prompt. You can run any of the other Tomcat commands from the same prompt.
 - __c. Another alternative is to make a small addition to the startup and shutdown bat files found in C:\apache-tomcat-5.5.28\bin.
 - __i. Download a J2SE 5 JDK and unzip it to C:\J2SE5
 - __ii. In both the startup.bat and shutdown.bat files, add the following lines near the top of the files (just after the opening block comment).

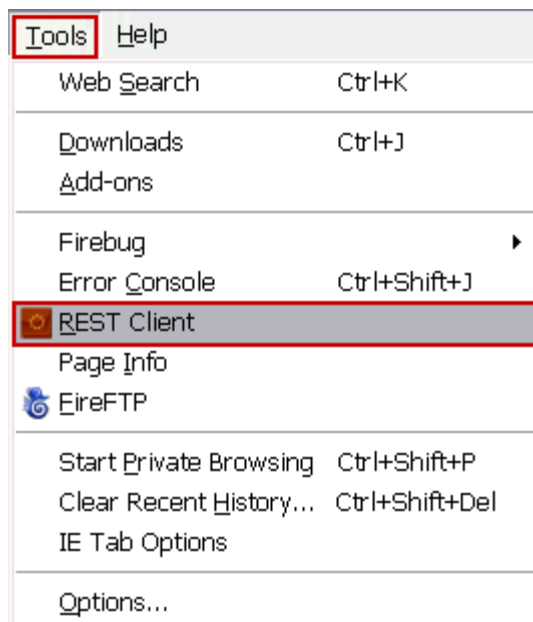
```
rem JAVA_HOME check
if not "%JAVA_HOME%" == "" goto gotJavaHome
set JAVA_HOME=C:\J2SE5
:gotJavaHome
```
 - __iii. From Windows explorer, simply double click the startup.bat and shutdown.bat files to start and stop Tomcat.
- __4. This workshop will assume that Tomcat can be started or stopped simply by double clicking the start or stop bat file from Windows explorer. That is, either 3.a or 3.c above is true.

1.3 Install the REST Client Add-on for Mozilla® Firefox®

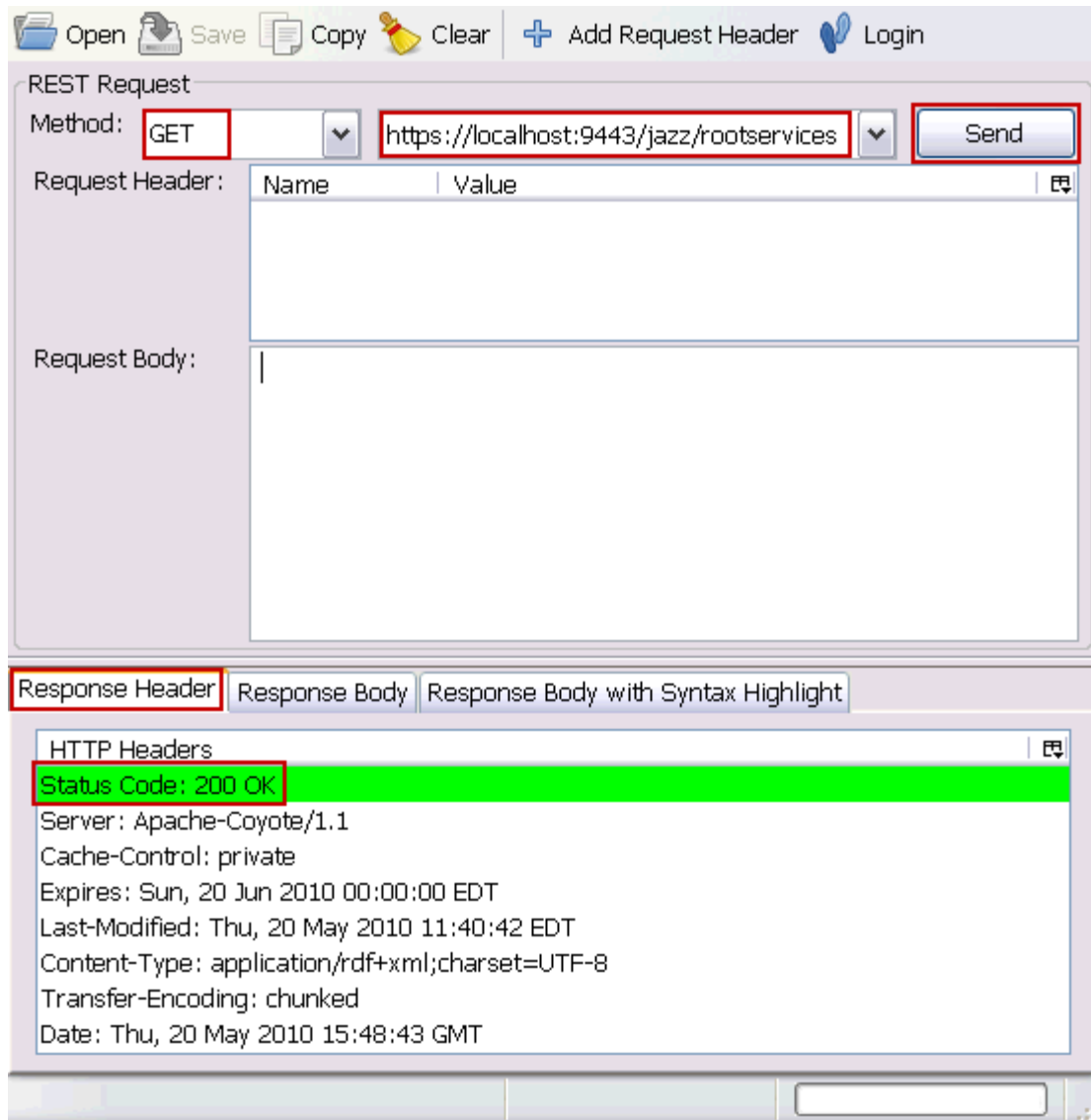
- __1. This workshop is written assuming Mozilla Firefox and the REST Client add-on. If you can not use Mozilla Firefox, alternatives would include the cURL command line tool (<http://curl.haxx.se/>) or a standalone application (<http://jamescrisp.org/2008/08/08/simple-rest-client/>). Add-ons for other browsers may also exist.
- __2. If you do not already have Mozilla Firefox installed, go to <http://www.mozilla.com/firefox/>, download version 3.5 or 3.6 and follow the installation instructions.
- __3. Start Mozilla Firefox, go to this page (<https://addons.mozilla.org/firefox/addon/9780/>), click the **Add to Firefox** button and follow the instructions, including the restart of Mozilla Firefox.

1.4 Test the REST Client Setup

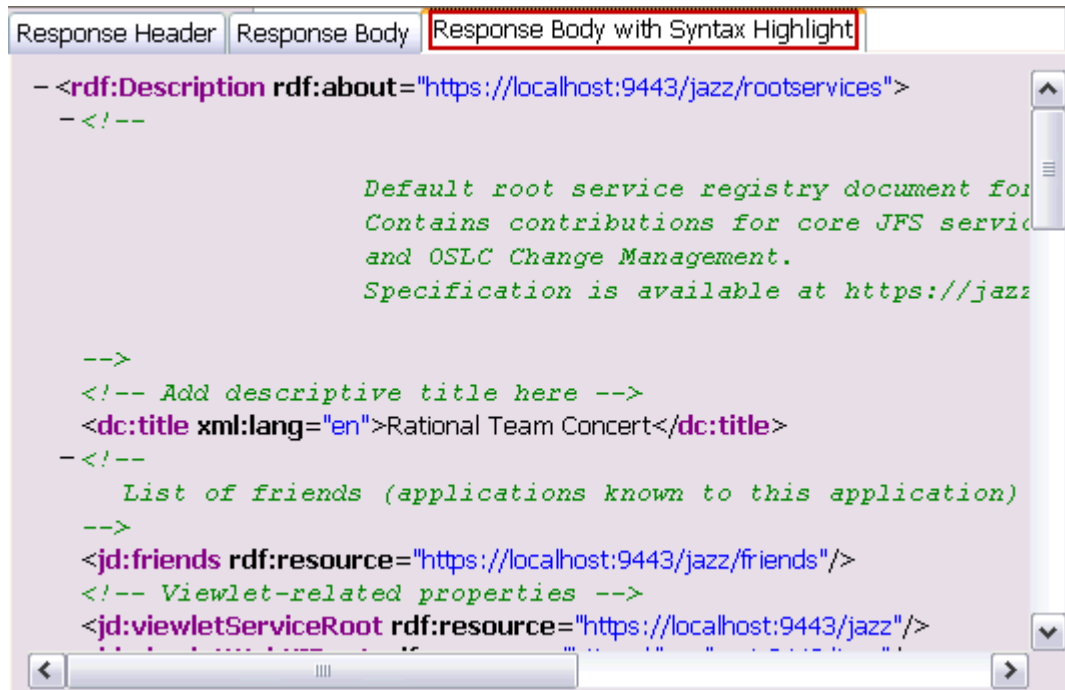
- __1. Start your Jazz Team Server. For a default installation, the public URI (root of all the URLs) used in later steps will be: <https://localhost:9443/jazz>
 - __a. In the Windows Explorer, navigate to the **C:\RTC2002Dev\jazz\server** folder.
 - __b. Double click the **server.startup.bat** file to start the server.
- __2. Open Mozilla Firefox and, from the menu bar, select **Tools > REST Client**. It will open a new **REST Client for Firefox** tab.



- 3. Select the **GET** method from the combo-box and type the URL: <https://localhost:9443/jazz/rootservices> and then press **Send**. The **Response Header** should show in green that the **Status Code** is "200 OK".



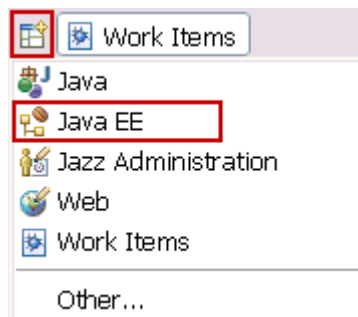
- __4. Switch to the **Response Body with Syntax Highlight** tab to see the content of the response. It will be an XML stream describing the root services of your Jazz Team Server.



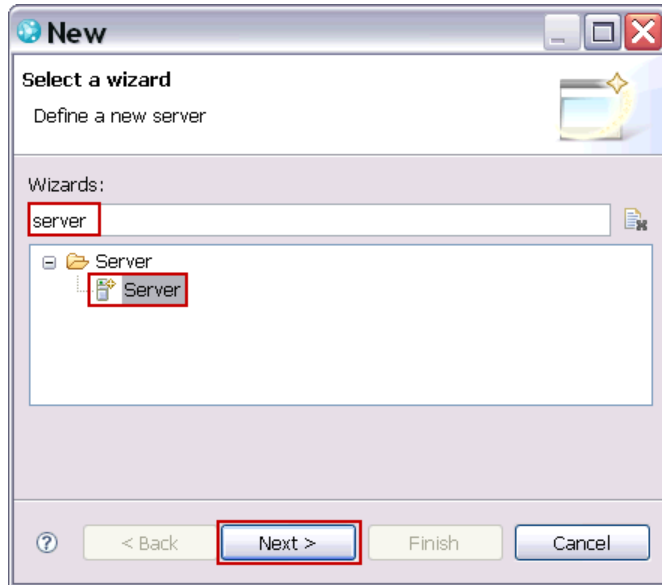
- __5. If you have reached this step then it means that the add-on is functioning correctly.

1.5 Test the WTP and Tomcat Setup

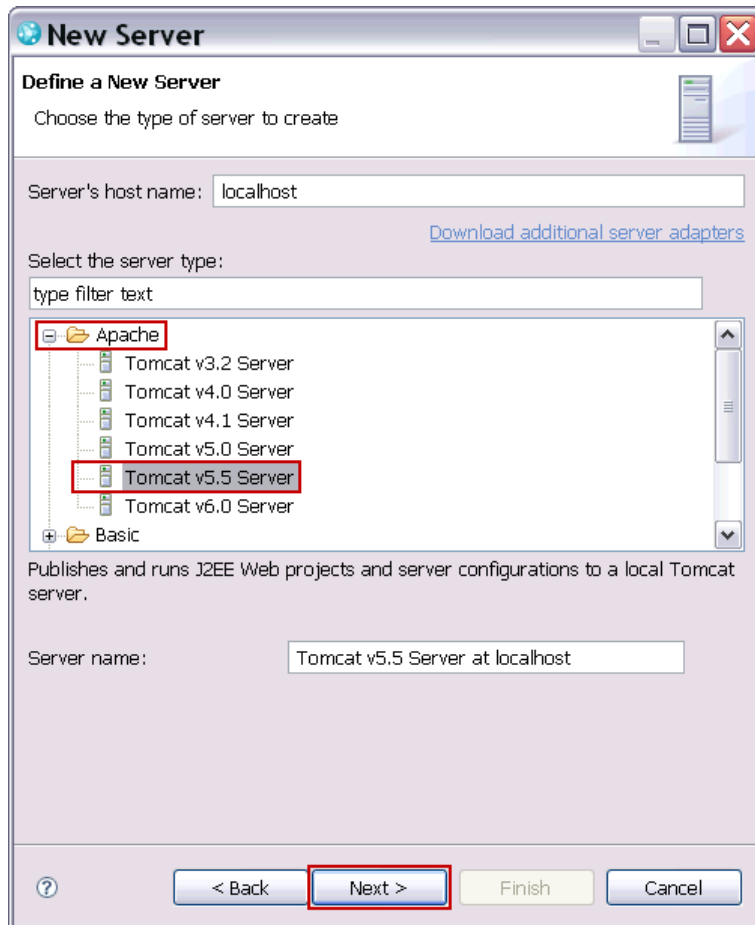
- __1. Create a Tomcat Server definition.
- For the OSLC Producer workshop you will use the Web Tool Platform and deploy your code against the Apache Tomcat application server. You will test the configuration now.
 - If the RTC client with WTP is not still running, start It now.
 - Open the Java EE perspective.



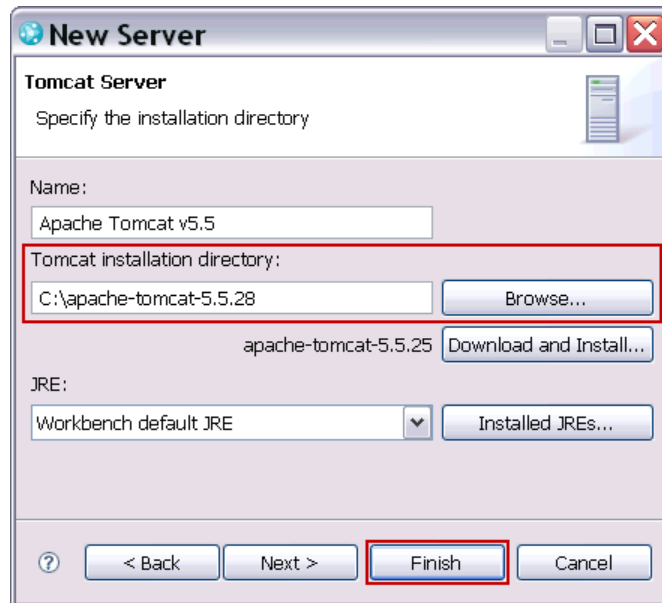
- ___d. From the menu bar select **File > New > Other...** then in the **New** wizard, type `server` in the filter, select **Server** from the list and then click **Next**.



- ___e. On the second page of the wizard, expand the **Apache** tree, select **Tomcat v5.5 Server** and then click **Next**.

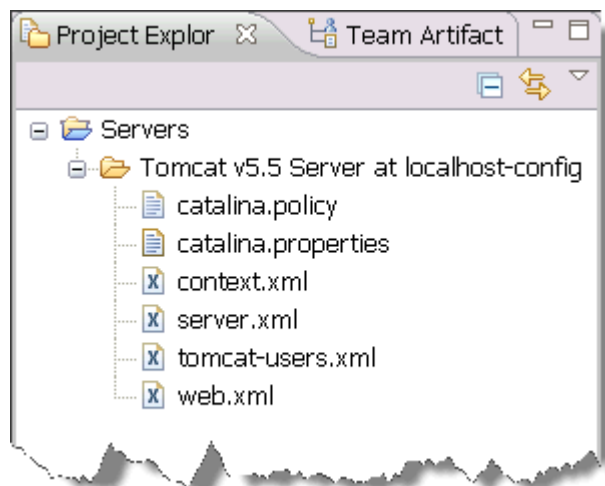


- ___f. On the final page of the wizard, specify where you have unzipped Tomcat (use the **Browse...** button) and press **Finish**.



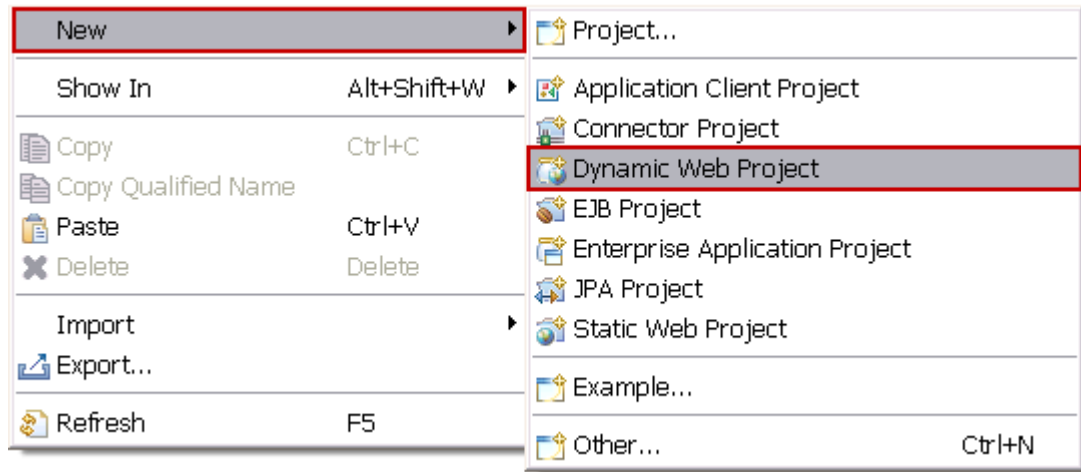
If you already have installed Tomcat in the Eclipse workspace then this wizard page will not show up.

- ___g. A project (named **Servers**) is created to contain your server definitions. It is visible in the **Project Explorer** view. The project contains a set of property files which will be used when run your servlets on the Tomcat server.

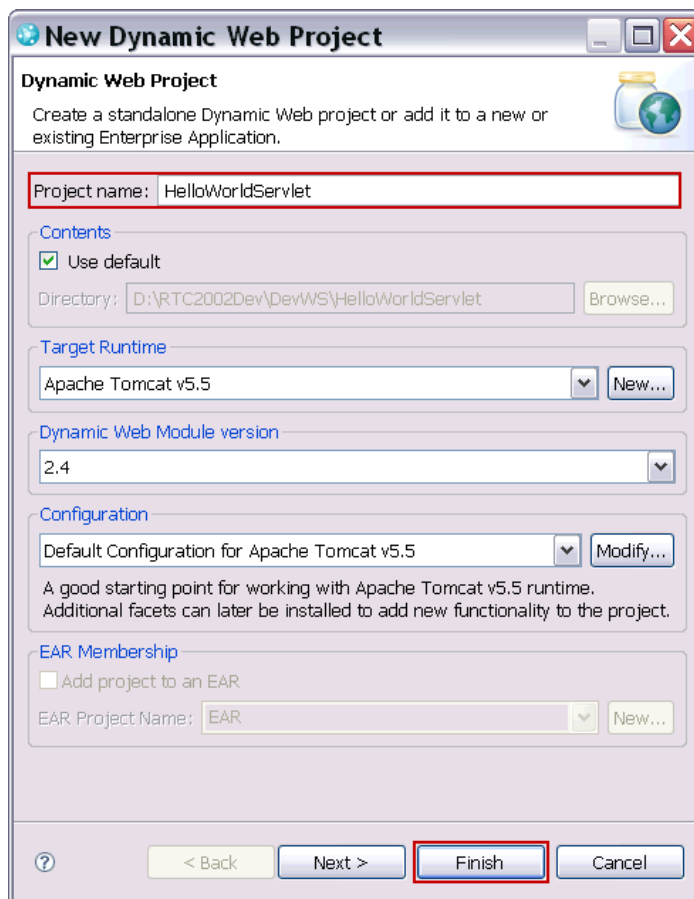


__2. Create a new web project to run on the Tomcat server.

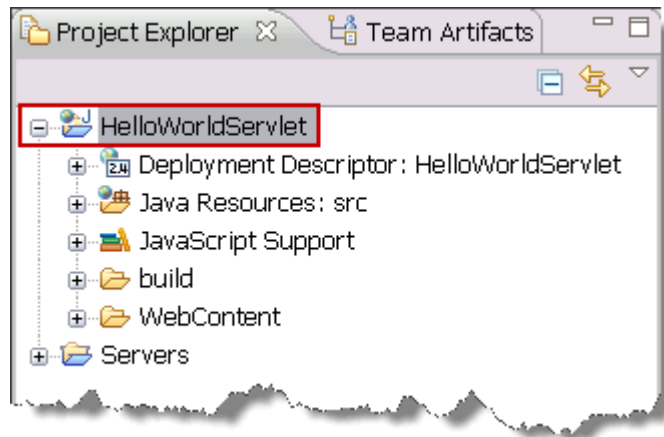
__a. Right click in empty space in the **Project Explorer** view, then select **New > Dynamic Web Project**.



__b. In the **Dynamic Web Project** wizard, type HelloWorldServlet into the Project name field and click **Finish**.

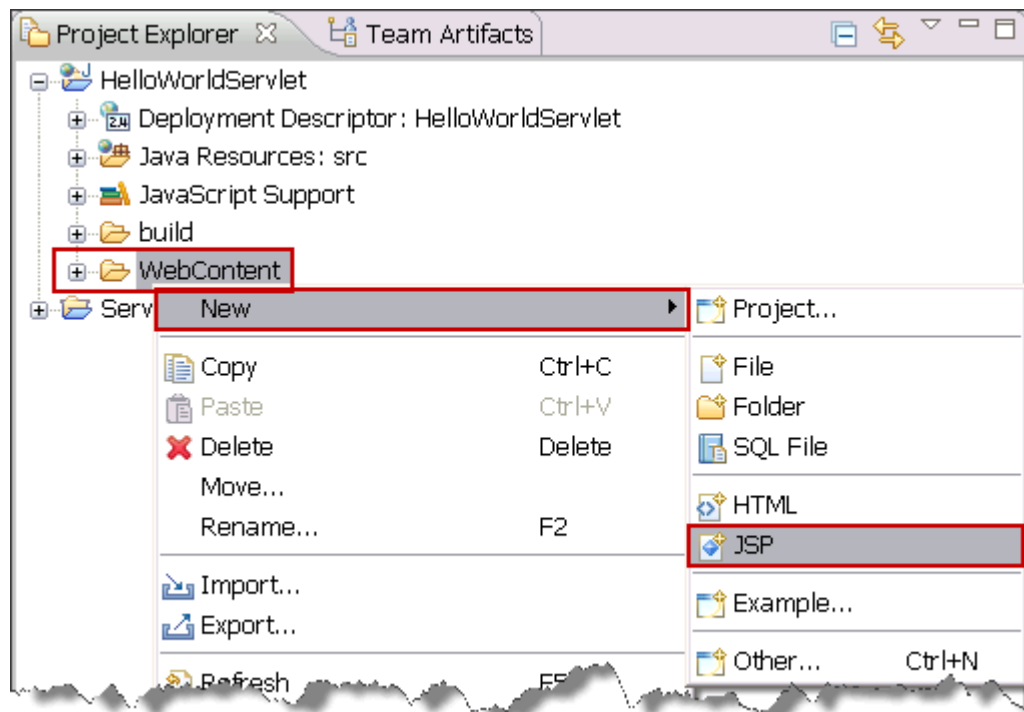


- __c. A web project is created in the Project Explorer.

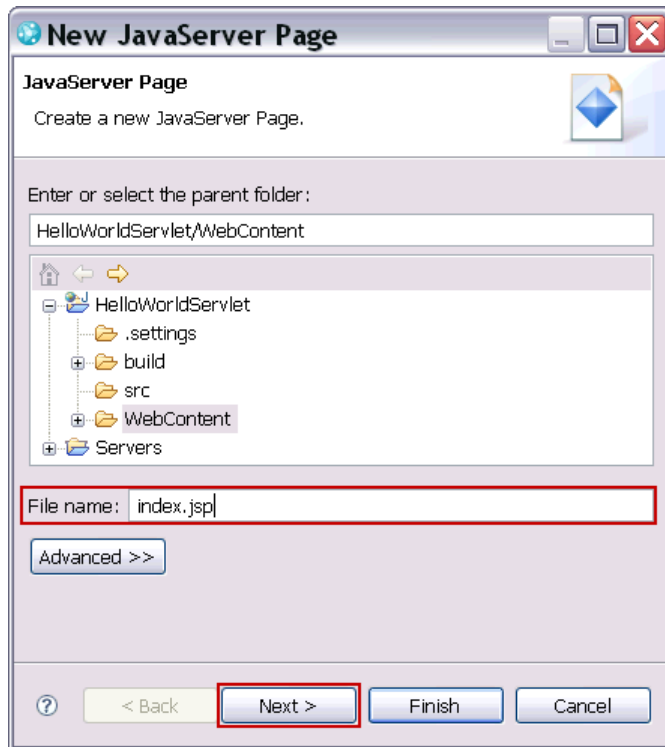


- __3. Create the default page for the web project.

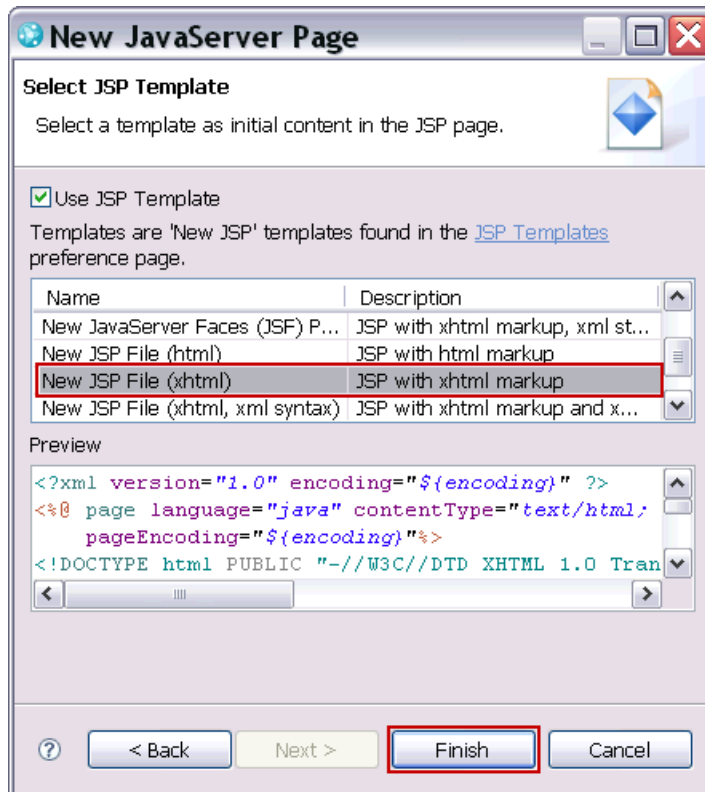
- __a. Right click the **WebContent** folder and then select **New > JSP**.



- __b. In the **New Java Server Page** wizard, type `index.jsp` into the **File name** field and then click **Next**.



- __c. In the second page of the wizard, **New JSP File (xhtml)** and click **Finish**.



- ___d. A file named **index.jsp** will be created under the **WebContent** folder and an editor will open for this new file.

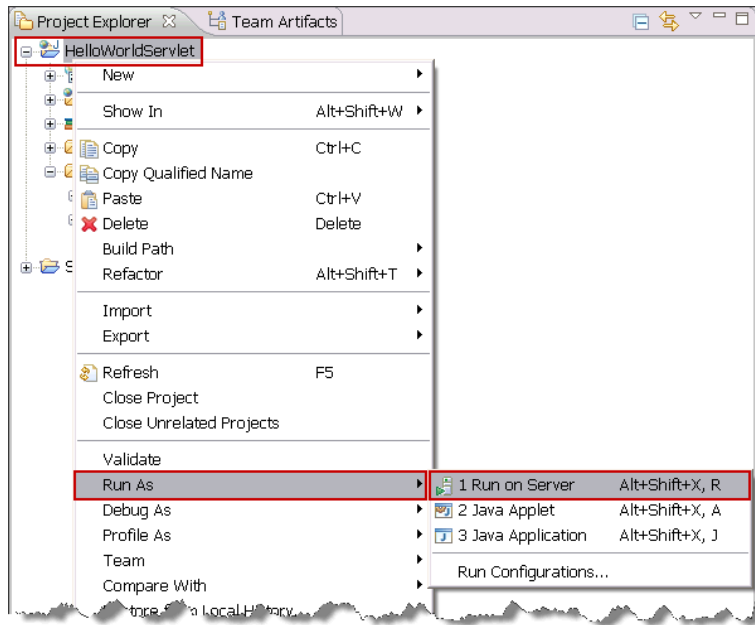


- ___e. Within the `<body>` element, type the following HTML.
Hello World from JSP, it is `<%= new Date().toString() %>`
- ___f. If after typing `Date`, you use code assist (`Ctrl+Space`), a list of possible classes will be presented. Select `java.util.date` from the list and the import of that class will be automatically added to your file as shown here.

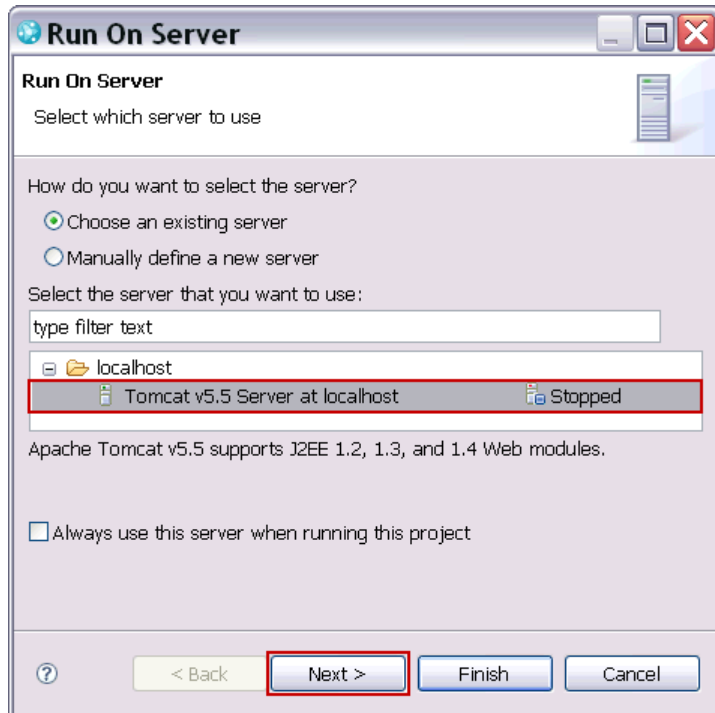


- ___g. Save your changes (`Ctrl+S`).

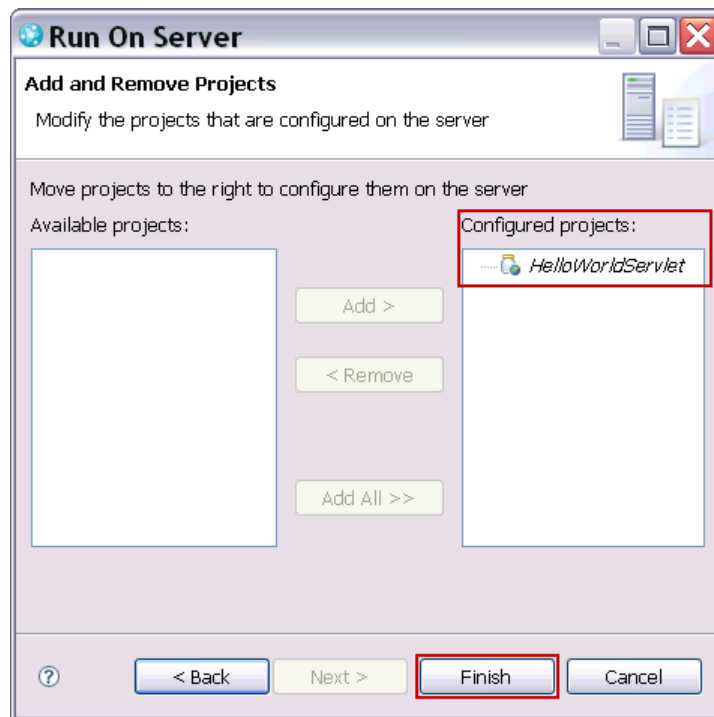
- __4. Run the web project on the Tomcat server.
 - __a. In the Project Explorer view, right click the HelloWorldServlet project then select **Run As > Run on Server** from the menu.



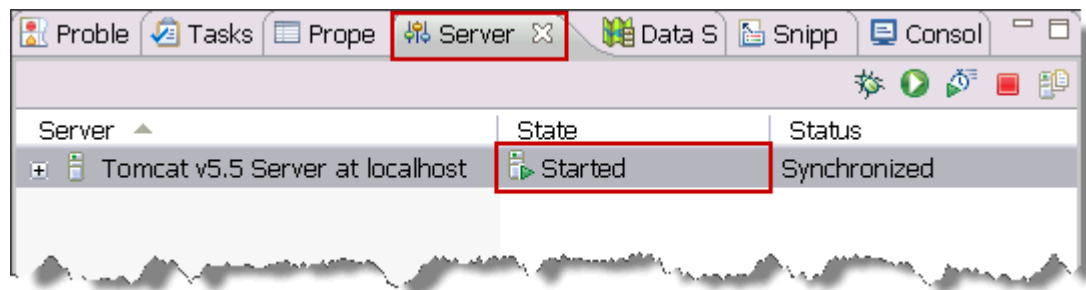
- __b. In the **Run On Server** wizard select your Tomcat server and press **Next**.



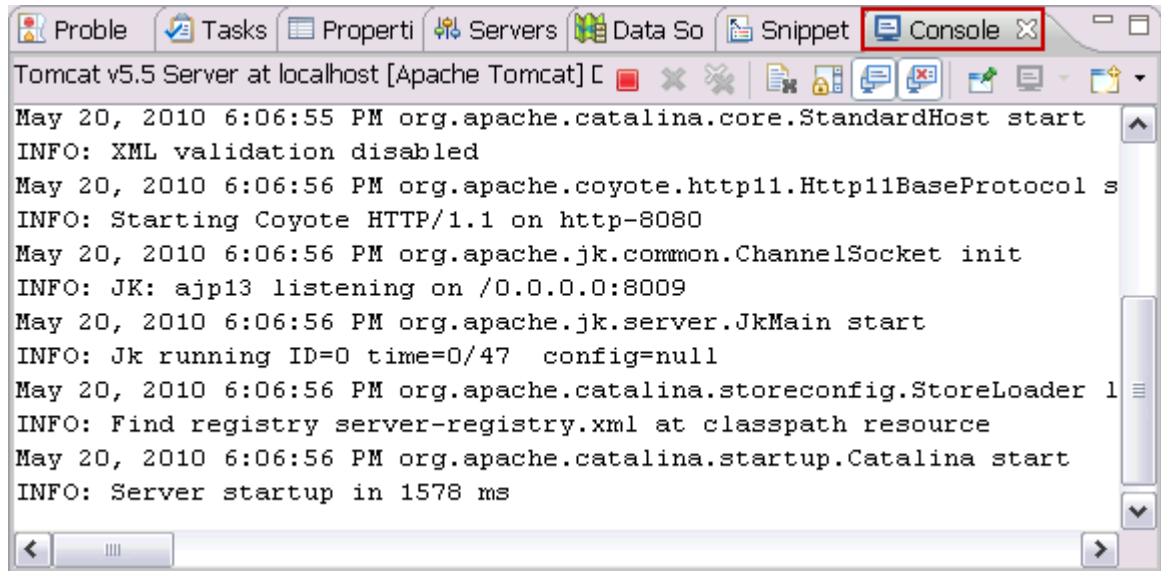
- ___c. On the second page verify that your servlet has been correctly added to the **Configured projects** list and then click **Finish**.



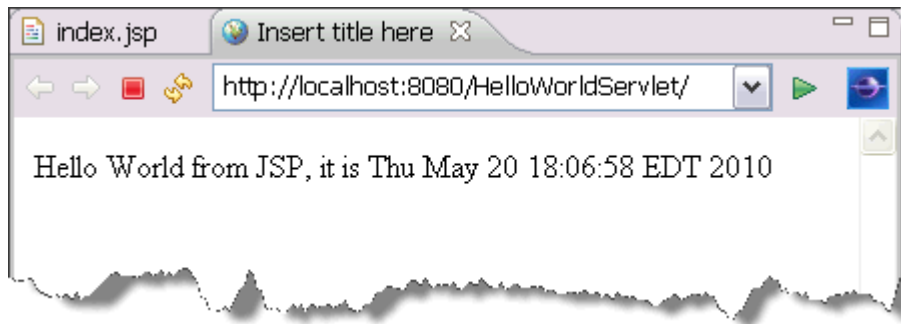
- ___d. Soon, the **Servers** view will show that the server has started...



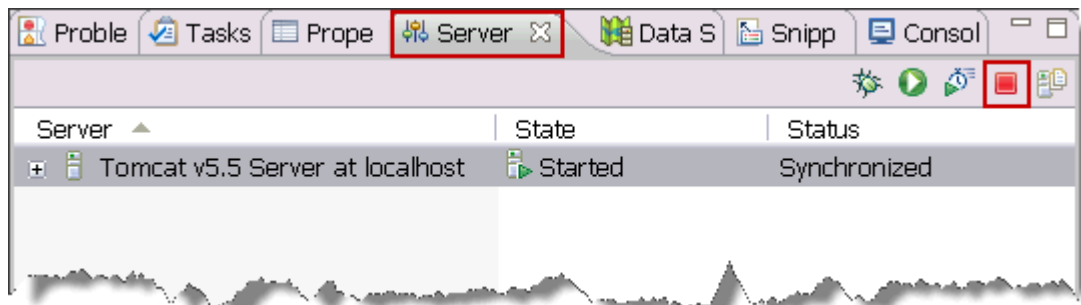
__e. The **Console** view will display some info about the running server...



__f. The Eclipse internal Web Browser view will open on the the URL of your servlet and display your JSP with the current timestamp.



__5. If you have reached this last step, it means that your Tomcat is correctly set up and the WTP features are properly installed. You can close the browser and any open editors. Shutdown the Tomcat server from the Servers view by clicking the **Stop the Server** icon.



1.6 Setup for the Remaining Labs

- __1. The `ExtensionsAndIntegrationsLabCodeRepository-yyyymmdd.tar` file you have contains an RTC repository. The repository has one project area, stream and workspace for this workshop (it may also contain artifacts for other workshops). The stream and workspace contain two components: one for the OSLC consumer labs and one for the OSLC provider labs. Right now, you are going to import this repository into the Tomcat server's database. In subsequent labs, you will connect to the project area from the RTC Eclipse workspace you have configured in this lab to obtain the code.
 - __a. From the Windows Explorer, copy the tar file to the `C:\RTC2002Dev\jazz\server` folder.
 - __b. Make sure the server is stopped. If it is not, double click the `server.shutdown.bat` file to stop the server.
 - __c. Open a command prompt in the `C:\RTC2002Dev\jazz\server` folder.
 - __d. Type the command `repotools -import fromFile=ExtensionsAndIntegrationsLabCodeRepository-yyyymmdd.tar` and then hit enter. Replace `yyyymmdd` with the actual date stamp in the file name.
 - __e. Type the command `repotools -rebuildTextIndices` and then hit enter.
 - __f. If you are continuing on to lab 2 right now, double click the `server.startup.bat` file to start the server.



You have completed lab 1. You now have a complete development environment for OSLC consumers and providers.

Lab 2 An introduction to the OSLC APIs



Lab Scenario

You will learn how to retrieve and use some default OSLC API directly from your favorite web browser.

If your RTC server is not running, start it now
(C:\RTC2002Dev\jazz\server\server.startup.bat).

2.1 The Root Services document

1. Open the **Mozilla Firefox** internet browser by double-clicking the **Mozilla Firefox** shortcut



on the **Windows Desktop**.

2. Enter the URL: <https://localhost:9443/jazz/rootservices>
This URL will return the **Root Services** document which is an XML informational resource that lists a set of REST services and capabilities.

```

Mozilla Firefox
File Edit View History Bookmarks Tools Help
localhost https://localhost:9443/jazz/rootservices
https://localhost:9443/jazz/rootservices
This XML file does not appear to have any style information associated with it. The document tree is shown below.
- <rdf:Description rdf:about="https://localhost:9443/jazz/rootservices">
  - <!--
    Default root service registry document for an RTC server.
    Contains contributions for core JFS services and components
    and OSLC Change Management.
    Specification is available at https://jazz.net/wiki/bin/view/Main/RootServicesSpec
  -->
  <!-- Add descriptive title here -->
  <dc:title xml:lang="en">Rational Team Concert</dc:title>
  - <!--
    List of friends (applications known to this application)
  -->
  <jd:friends rdf:resource="https://localhost:9443/jazz/friends"/>
  <!-- Viewlet-related properties -->
  <jd:viewletServiceRoot rdf:resource="https://localhost:9443/jazz"/>
  <jd:viewletWebUIRoot rdf:resource="https://localhost:9443/jazz"/>
  - <!--
    Default root service registry document for a JFS server.
  -->
  <jfs:oauthRequestTokenUrl rdf:resource="https://localhost:9443/jazz/oauth-request-token"/>
  <jfs:oauthAccessTokenUrl rdf:resource="https://localhost:9443/jazz/oauth-access-token"/>
  <jfs:oauthRealmName>Jazz</jfs:oauthRealmName>
  <jfs:oauthDomain>https://localhost:9443/jazz</jfs:oauthDomain>
  <jfs:oauthUserAuthorizationUrl rdf:resource="https://localhost:9443/jazz/oauth-authorize"/>
  <jfs:oauthRequestConsumerKeyUrl rdf:resource="https://localhost:9443/jazz/oauth-request-consumer"/>
  
```

Root Services



The Root Services URL locates a REST API for discovering the Jazz Team Server's various services and specific capabilities, such as a way to discover a web UI presentation for particular kinds of resources.

This Service is an open-ended mechanism that can be used to keep track of and share important information about the Jazz Team Server. There are backstage mechanisms for the various services to register their important information with the Discovery Service.

The Discovery Service is also used by all services to discover the whereabouts of other services provided by the other tools affiliated with this JTS (Jazz Team Server).

- __3. One of these services is the “**whoiam**” service. Look for a tag named **jfs:currentUser**. The URL associated to the unique attribute **rdf:resource** is the URL to access to the “**whoiam**” service.

```

<!-- service for finding users and getting user info -->
<jfs:users rdf:resource="https://localhost:9443/jazz/users"/>
- <!--
  Service to redirect to the resource that represents the authenticated user
-->
<jfs:currentUser rdf:resource="https://localhost:9443/jazz/whoami"/>
<!-- JFS storage service -->
<jfs:storage rdf:resource="https://localhost:9443/jazz/storage"/>
<!-- JFS SPARQL query service -->
<jfs:query rdf:resource="https://localhost:9443/jazz/query"/>

```

- __4. Copy this URL (<https://localhost:9443/jazz/whoami>) and paste it into the navigation field of your web browser and press Enter.

Login / Protected resources

During the following lab examples, you might face a situation where you get a 200 OK status code but not the expected result:

The screenshot shows an HTTP response header with the following fields:

- Status Code: 200 OK
- Server: Apache-Coyote/1.1
- Cache-Control: private
- Expires: Thu, 01 Jan 1970 01:00:00 CET
- X-com-ibm-team-repository-web-auth-msg: authrequired
- Last-Modified: Thu, 20 May 2010 16:25:37 GMT
- Content-Type: text/html;charset=utf-8
- Content-Length: 2358
- Date: Thu, 20 May 2010 16:25:47 GMT

In this case, please check out if the HTTP headers you got doesn't have a key/value field set to

X-com-ibm-team-repository-web-auth-msg: authrequired

Then it means that you try to access to a protected resource and you need to login first.

In such case, follow these steps to fix this issue:

1. Open a new tab on your current web browser
2. Login to the server entering the following URL in your navigation field: <https://localhost:9443/jazz/web>

The login dialog Web UI will show up:

The screenshot shows the Jazz Team Server login dialog with the following fields and options:

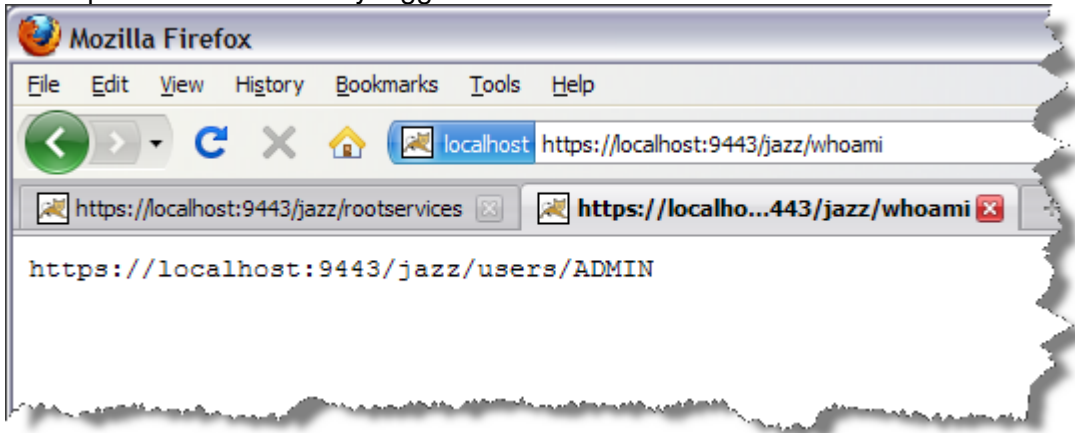
- User ID: ADMIN
- Password: [masked]
- Remember my User ID
- Log In button

At the bottom, there is a copyright notice: "Licensed Material - Property of IBM Corp. © Copyright IBM Corp. and its licensors 2008, 2009. All Rights Reserved. IBM, the IBM logo, Jazz, and Rational are trademarks of IBM Corporation, in the United States, other countries, or both. Built on Eclipse is a trademark of Eclipse Foundation, Inc. Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both." The IBM logo and Rational software logo are also present.

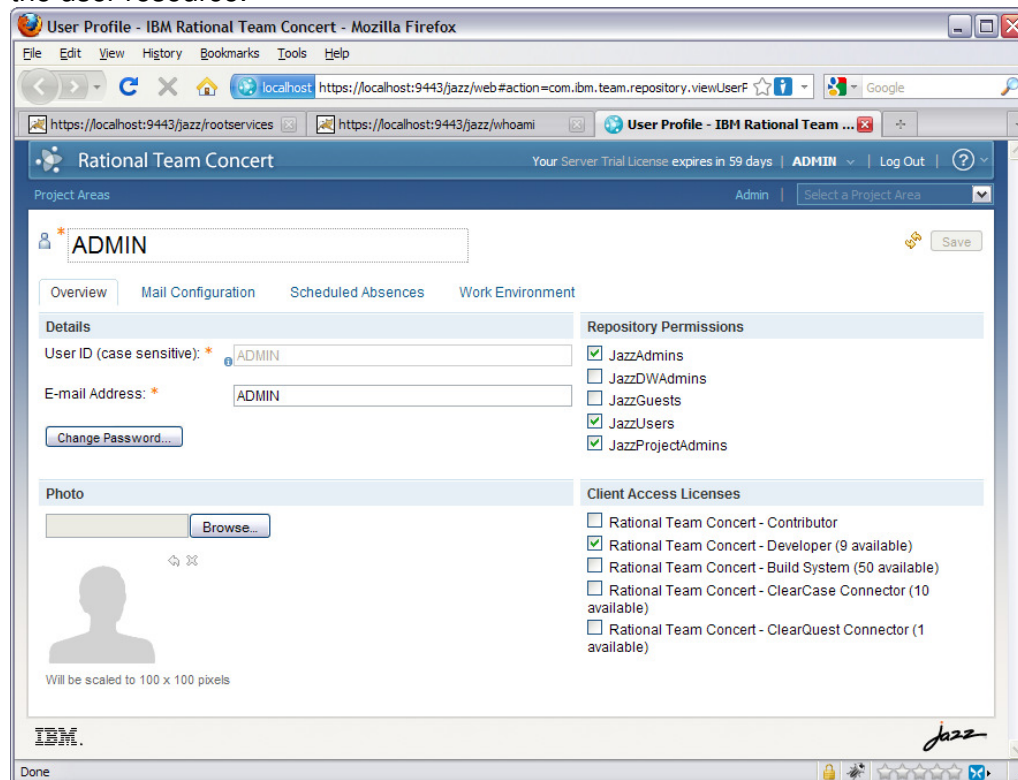
3. Login with ADMIN as both **User ID** and **Password**
4. Once you are logged, switch back to the REST client and retry to run the described sample. It should work as expected.



- __5. The REST service provides a single resource that returns the URL of a user resource that corresponds to the currently logged on user:



- __a. Copy this URL and paste it in the navigation field of your web browser to directly access the user resource.



Because we didn't mention the format we would like to access this resource with, the Jazz Team Server redirected us to the **User Editor**.

2.2 The REST Client add-on



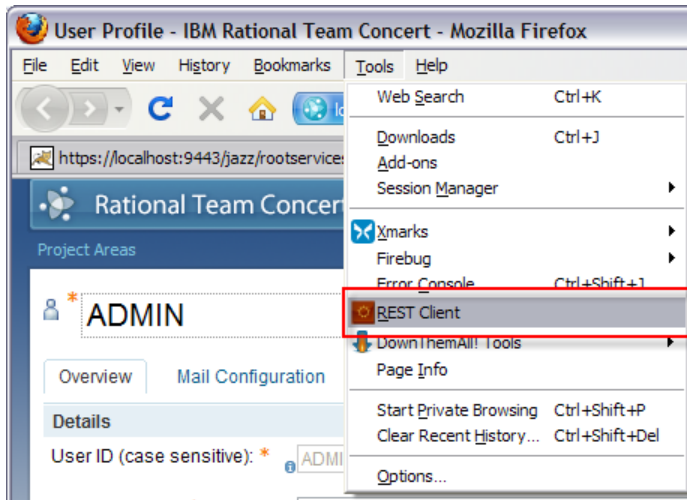
REST Client add-on

To be able to properly handle the response of a REST service, we need to use a REST client application which will let us specify all of the parameters required for an HTTP method.

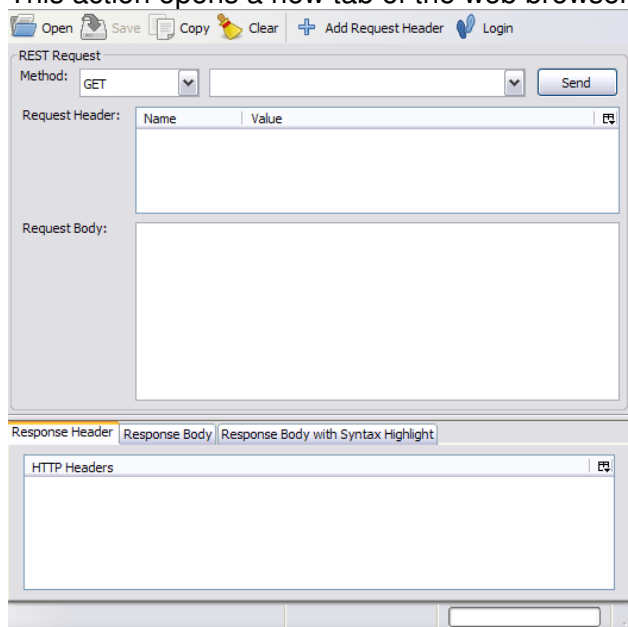
There are several REST clients available on the web. We have chosen to pick up one which can be installed directly on a Firefox web browser.

This REST client is a Firefox add-on named “**REST Client**” (<https://addons.mozilla.org/en-US/firefox/addon/9780>).

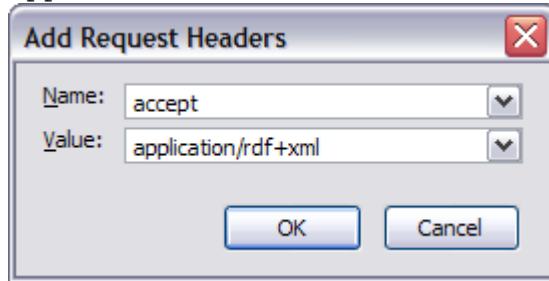
1. From the Tools menu of your web browser, select the item **REST client** to open the REST client UI.



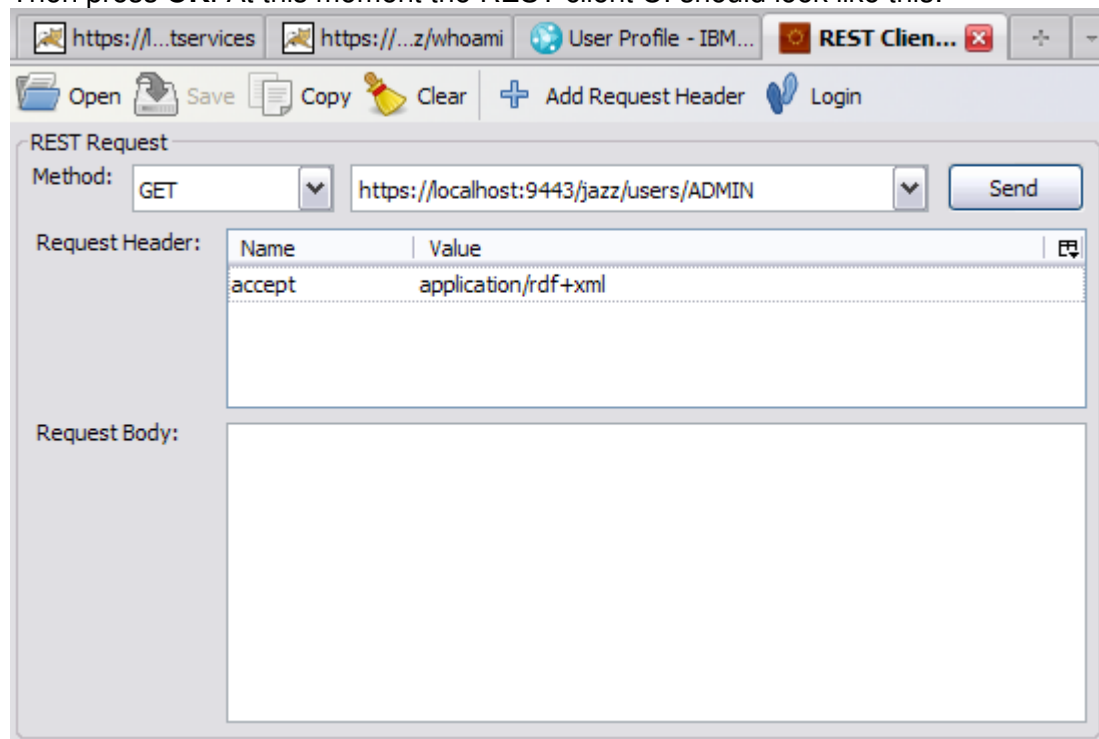
This action opens a new tab of the web browser titled **REST Client**.



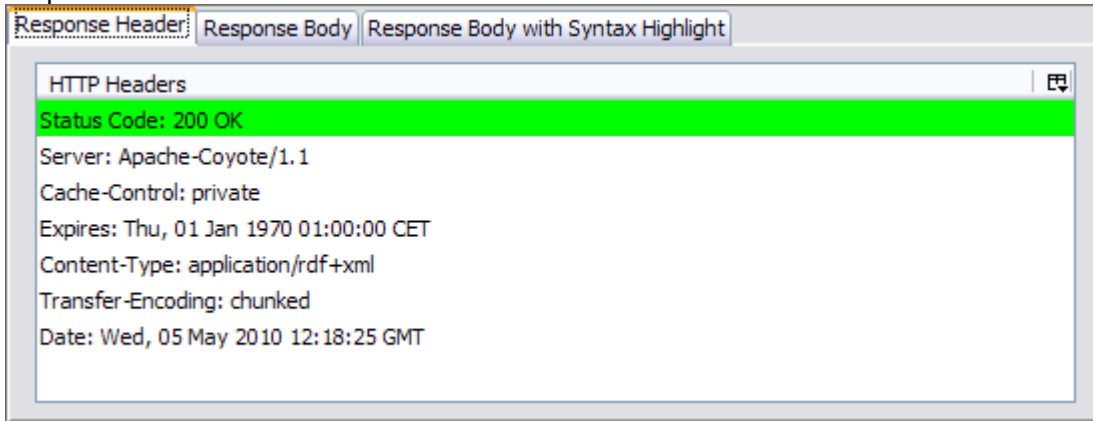
- __2. Now we should be able to specify the correct parameters to access the XML representation of the user resource.
- __a. Choose the `GET` method from the **Method** combo-box
 - __b. In the URL field, copy/paste the URL of the current user that the “whoiam” service has revealed to you (i.e. <https://localhost:9443/jazz/users/ADMIN>).
 - __c. Press the **Add Request Header** button and add the header key `accept` and the value `application/rdf+xml`.



- __d. Then press **OK**. At this moment the REST client UI should look like this:

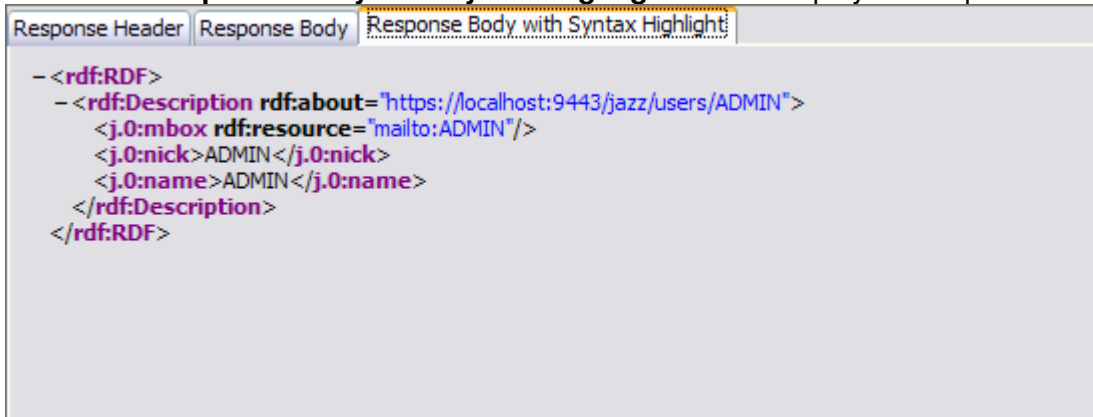


- __3. Press the **Send** button. The REST service **Response Header** displays the headers of the HTTP response:



If the **Status Code** has the value “200 OK” it means that the HTTP request was handled properly. If not, please check out the URL and the header you have provided.

- __4. Press the “**Response Body with Syntax Highlight**” tab to display the response body.



The representation of a user resource is based on the **Friend of a Friend (FOAF) RDF vocabulary** (<http://xmlns.com/foaf/spec/>).

2.3 OSLC services

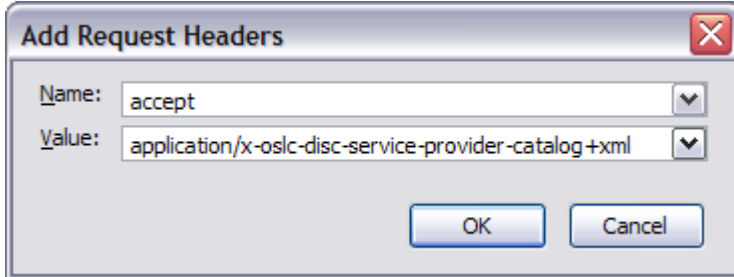
Until now we were calling Jazz Foundation REST services which are specific to the Jazz Team Servers. It is time now to call specifically the OSLC REST services implemented in the Jazz Team Server. The specifications of these APIs are defined on the web site of the Open Services for Lifecycle Collaboration community (<http://open-services.net>).

- __1. From the Root Services document, extract the Change Management Catalog URL (pointed to by **rdf:resource**) of the element **oslc_cm:cmServiceProviders**.

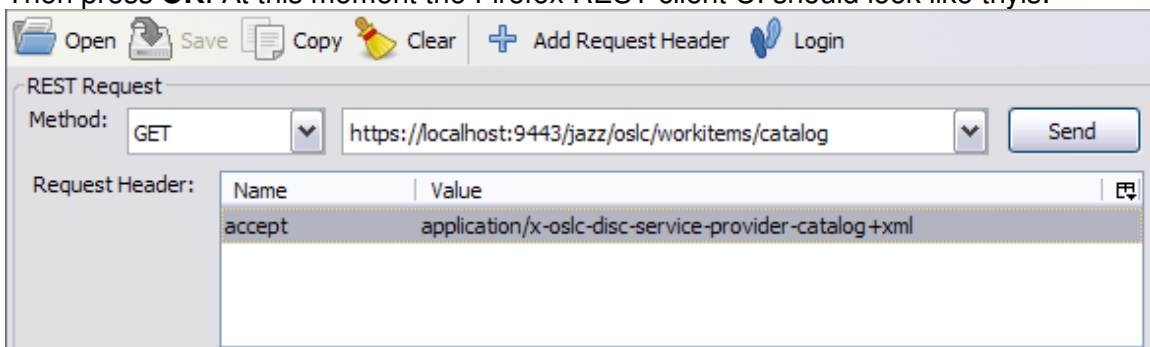
```
<!-- Change Management service catalog -->
<oslc_cm:cmServiceProviders rdf:resource="https://localhost:9443/jazz/oslc/workitems/catalog"/>
```

- __2. Copy this URL (<https://localhost:9443/jazz/oslc/workitems/catalog>) and paste it into the URL field of REST client.

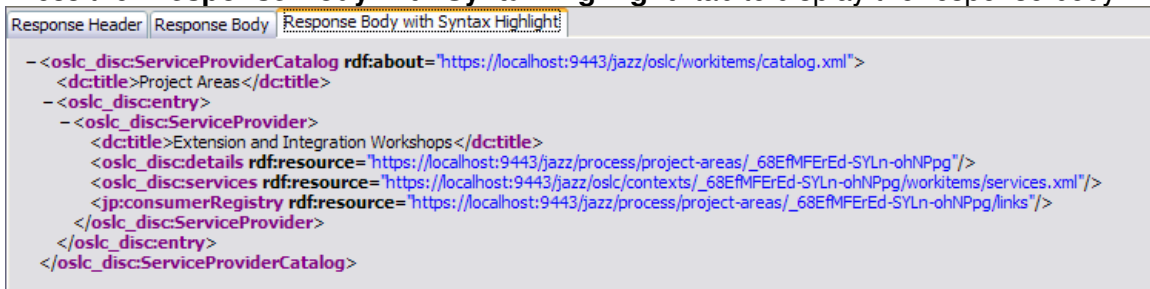
- ___3. If you already have any request header setup, select it then press the **Del** key to remove the existing header.
- ___4. Press the **Add Request Header** button and add the header key `accept` with the value `application/x-oslc-disc-service-provider-catalog+xml`.



- ___5. Then press **OK**. At this moment the Firefox REST client UI should look like this:



- ___6. Press the **Send** button. The REST service response header will be displayed at on bottom part of the UI. If the Status Code doesn't show the value "200 OK", please check out the URL and the header you have provided.
- ___7. Press the "**Response Body with Syntax Highlight**" tab to display the response body.

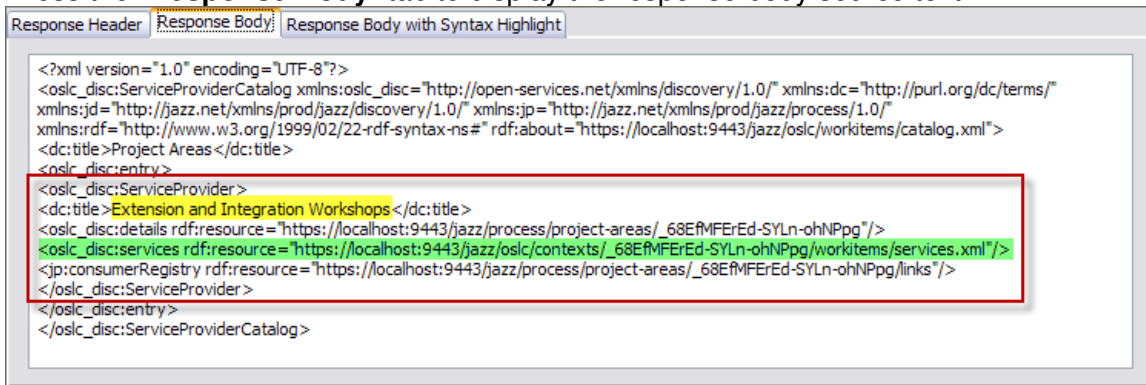


The resulting document contains a list of **ServiceProvider** elements that point to the documents which contain the actual service descriptions.

In the case of RTC, there is one **ServiceProvider** element for each **Project Area**.

Typically, an application would use the title of this element to allow the user to choose between the project areas.

8. Press the “**Response Body**” tab to display the response body source text.

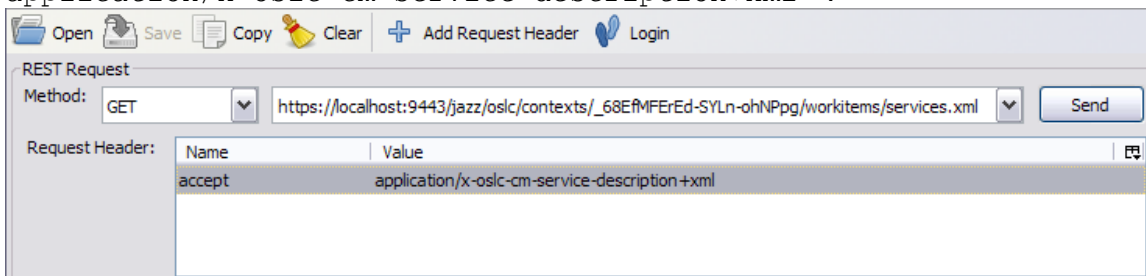


9. Retrieve the Service Provider element for the **Extension and Integration Workshops** Project Area.

10. Copy the URL associated to the attribute **rdf:resource** defined in the sub-element **oslc_disc:services**.

11. Paste the URL in the URL field of the REST client

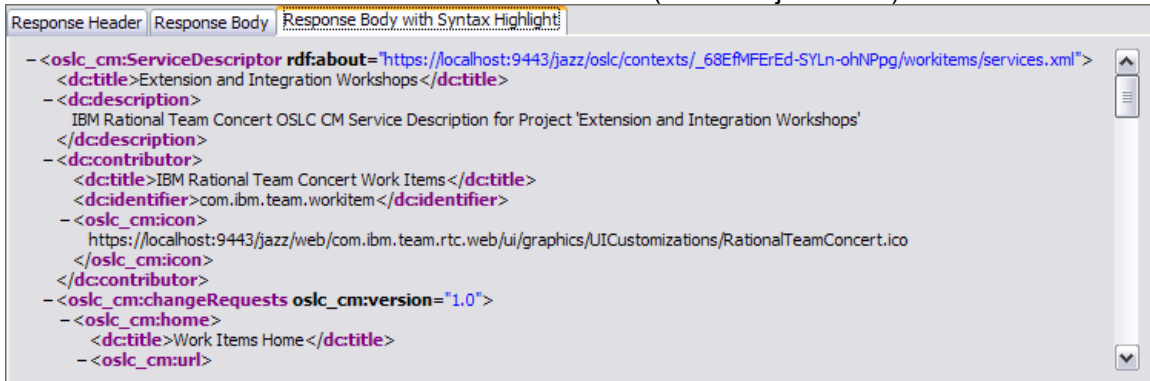
12. Replace the accept header by the following media-type:
application/x-oslc-cm-service-description+xml¹.



¹ The list of the Media Types available for the OSLC-CM is available at this link: http://open-services.net/bin/view/Main/CmRestApiV1#Media_Types_Used

Media Type	Resource
application/x-oslc-cm-change-request+xml	Change Request Resource
application/x-oslc-cm-service-description+xml	Service Descriptor Document
application/x-oslc-disc-service-provider-catalog+xml	Service Provider Catalog
application/atom+xml	Resource Collection
application/json	General JSON format request/response
application/xml	General XML format request/response
text/xml	

- __13. Press the **Send** button. The response body will display Service Provider document listing all the REST services available for this Service Provider (alias Project Area):



```

Response Header | Response Body | Response Body with Syntax Highlighti
- <oslc_cm:ServiceDescriptor rdf:about="https://localhost:9443/jazz/oslc/contexts/_68EfMFErEd-SYLn-ohNPpg/workitems/services.xml">
  <dc:title>Extension and Integration Workshops</dc:title>
  - <dc:description>
    IBM Rational Team Concert OSLC CM Service Description for Project 'Extension and Integration Workshops'
  </dc:description>
  - <dc:contributor>
    <dc:title>IBM Rational Team Concert Work Items</dc:title>
    <dc:identifier>com.ibm.team.workitem</dc:identifier>
    - <oslc_cm:icon>
      https://localhost:9443/web/com.ibm.team.rtc.web/ui/graphics/UICustomizations/RationalTeamConcert.ico
    </oslc_cm:icon>
  </dc:contributor>
  - <oslc_cm:changeRequests oslc_cm:version="1.0">
    - <oslc_cm:home>
      <dc:title>Work Items Home</dc:title>
    - <oslc_cm:url>

```

- links to the dialog modules,

```

- <oslc_cm:creationDialog oslc_cm:default="true" calm:id="defect" oslc_cm:hintWidth="740px" oslc_cm:hintHeight="510px">
  <dc:title>New Defect</dc:title>
  - <oslc_cm:url>
    https://localhost:9443/jazz/_ajax-modules
    /com.ibm.team.workitem.WICreationDialog?projectAreaName=Extension%20and%20Integration%20Workshops&
    dc%3Atype=defect
  </oslc_cm:url>
</oslc_cm:creationDialog>
- <oslc_cm:creationDialog calm:id="planItem" oslc_cm:hintWidth="740px" oslc_cm:hintHeight="510px">
  <dc:title>New Plan Item</dc:title>
  - <oslc_cm:url>
    https://localhost:9443/jazz/_ajax-modules
    /com.ibm.team.workitem.WICreationDialog?projectAreaName=Extension%20and%20Integration%20Workshops&
    dc%3Atype=com.ibm.team.appt.workItemType.story
  </oslc_cm:url>
</oslc_cm:creationDialog>

```

application/xhtml+xml	XHTML presentation format for Change Request Resource
text/html	HTML presentation format for Change Request Resource

- link to the Change Request factory URL to create new work items
 - `<oslc_cm:factory oslc_cm:default="true">`
 - `<dc:title>Default location for creation of change requests</dc:title>`
 - `<oslc_cm:url>`
 - `https://localhost:9443/jazz/oslc/contexts/_68EfMFErEd-SYLn-ohNPpg/workitems`
 - `</oslc_cm:url>`
 - `</oslc_cm:factory>`
 - `<oslc_cm:factory calm:id="drafts">`
 - `<dc:title>Location for creation of draft change requests</dc:title>`
 - `<oslc_cm:url>`
 - `https://localhost:9443/jazz/oslc/contexts/_68EfMFErEd-SYLn-ohNPpg/drafts/workitems`
 - `</oslc_cm:url>`
 - `</oslc_cm:factory>`

- and the REST service to query work items.
 - `<oslc_cm:simpleQuery>`
 - `<dc:title>CQL based change request queries</dc:title>`
 - `<oslc_cm:url>`
 - `https://localhost:9443/jazz/oslc/contexts/_68EfMFErEd-SYLn-ohNPpg/workitems`
 - `</oslc_cm:url>`
 - `</oslc_cm:simpleQuery>`

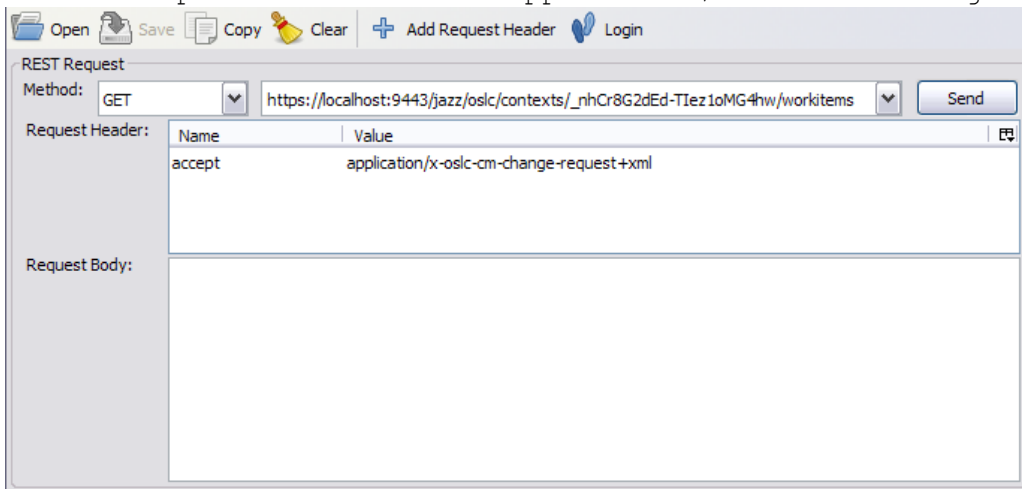
2.4 Search for some Work Items

In this paragraph, we will describe how we can query Work Items using the corresponding OSLC REST service.

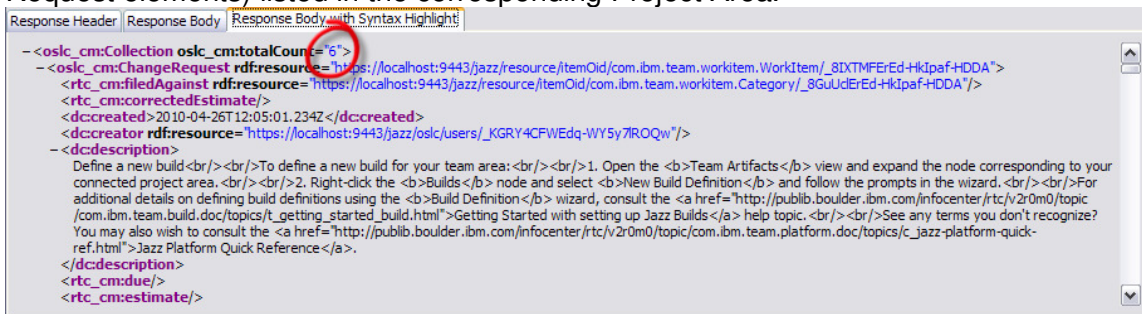
1. From the Service Provider services document, look for the `oslc_cm:url` element into the `oslc_cm:simpleQuery` element.
 - `<oslc_cm:simpleQuery>`
 - `<dc:title>CQL based change request queries</dc:title>`
 - `<oslc_cm:url>`
 - `https://localhost:9443/jazz/oslc/contexts/_68EfMFErEd-SYLn-ohNPpg/workitems`
 - `</oslc_cm:url>`
 - `</oslc_cm:simpleQuery>`

2. Copy this URL, also named Simple Query URL) and paste it into your REST client.

3. Set the Accept header with the value `application/x-oslcm-change-request+xml`.



4. Now, if you press the **Send** button, the response body will display **ALL** the work items (Change Request elements) listed in the corresponding Project Area.

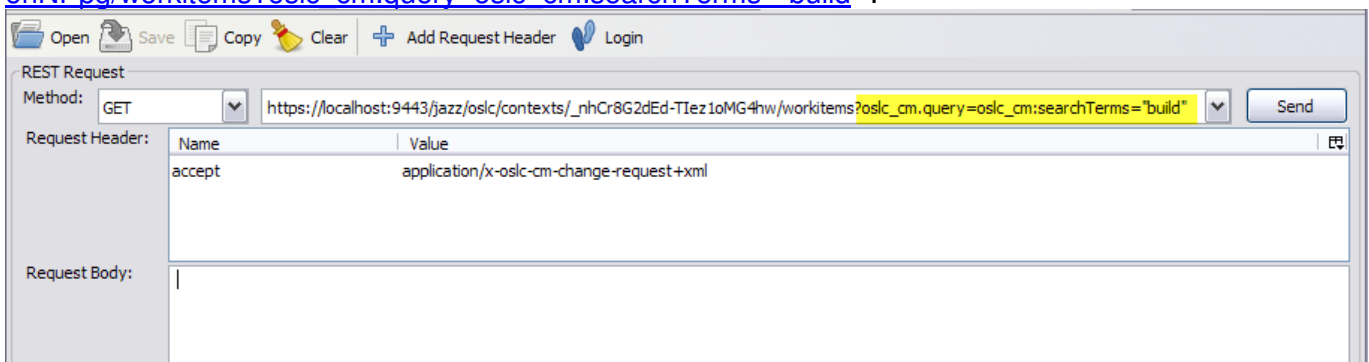


5. Actually, this REST service offers the possibility to filter the work items you are looking for and, for the selected work items, to specify which attributes you are interested in. First let try to retrieve all the work items which contain the word "build" in it.
To do so, complete the Simple Query URL with the following parameter:

```
?oslcm.query=oslcm:searchTerms="build"
```

So the URL should look like this:

[https://localhost:9443/jazz/oslcm/contexts/_68EfMFErEd-SYLn-ohNPpg/workitems?oslcm.query=oslcm:searchTerms="build"](https://localhost:9443/jazz/oslcm/contexts/_68EfMFErEd-SYLn-ohNPpg/workitems?oslcm.query=oslcm:searchTerms=) :



- Press the Send button. the response body will display a subset of work items:

- With this OSLC service, it is also possible to specify the subset of attributes you want to fetch from the server. For example, let say you are only interested in the work item ID (dc:identifier) and the work item title (dc:title). In this case, complete the previous URL with the following expression:

```
&oslc_cm.properties=dc:identifier,dc:title
```

- Press the Send button. The REST client will only display the requested attributes:



You might have noticed that the last query did return only the requested attributes, it also returned the **oslc_cm:score** attribute.

When `oslc_cm:searchTerms` is used in the request, each entry (hit) in the response contains a **oslc_cm:score** property which is a non-negative number and in the range from 0-100. This score should help to order the entries based on the largest `oslc_cm:score`.



Conclusion

You have completed lab 3. You now have an initial understanding of the OSLC APIs.

In the next lab you will learn how to programmatically access this API.

Lab 3 Access OSLC APIs programmatically



Lab Scenario

You will learn how to access OSLC APIs programmatically and you will build your first OSLC Consumer.

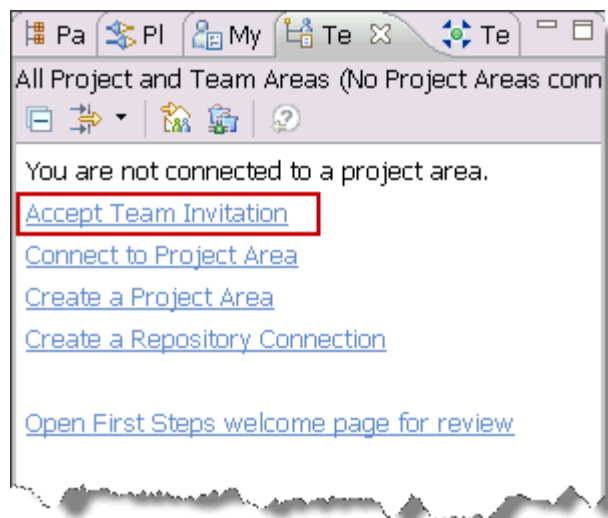
If your RTC server is not running, start it now

(C:\RTC2002Dev\jazz\server\server.startup.bat).

If your RTC development environment is not open, navigate to C:\RTC2002Dev\jazz\client\eclipse in the Windows explorer and double click **eclipse.exe**. If prompted to select an Eclipse workspace, create a new one that you should name OSLC-Consumer. If you are in a classroom environment where lab one was done for you, select the Eclipse workspace as directed by your instructor.

3.1 Loading Examples

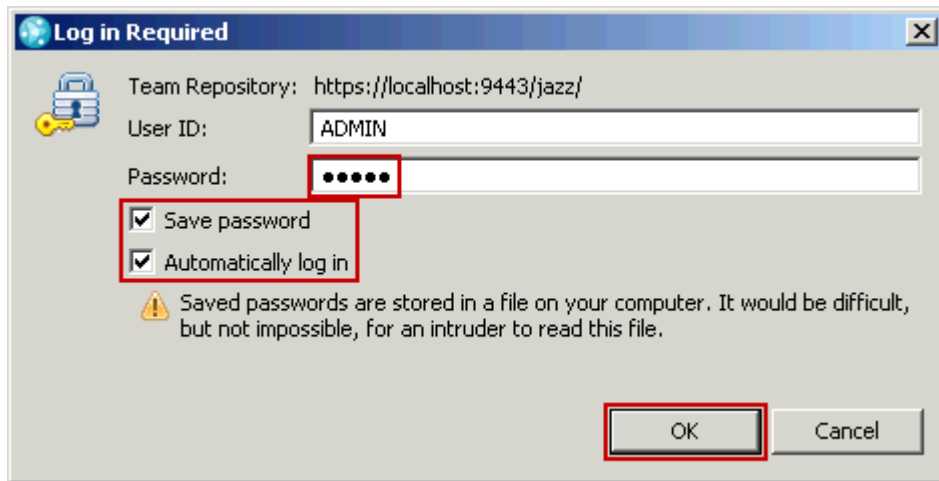
- __1. If the **Java** perspective is not open, open it now by selecting **Window > Open Perspective > Other... > Java** from the menu bar.
- __2. Load the lab code.
 - __a. On the left, switch to the **Team Artifacts** view and click the **Accept Team Invitation** link.



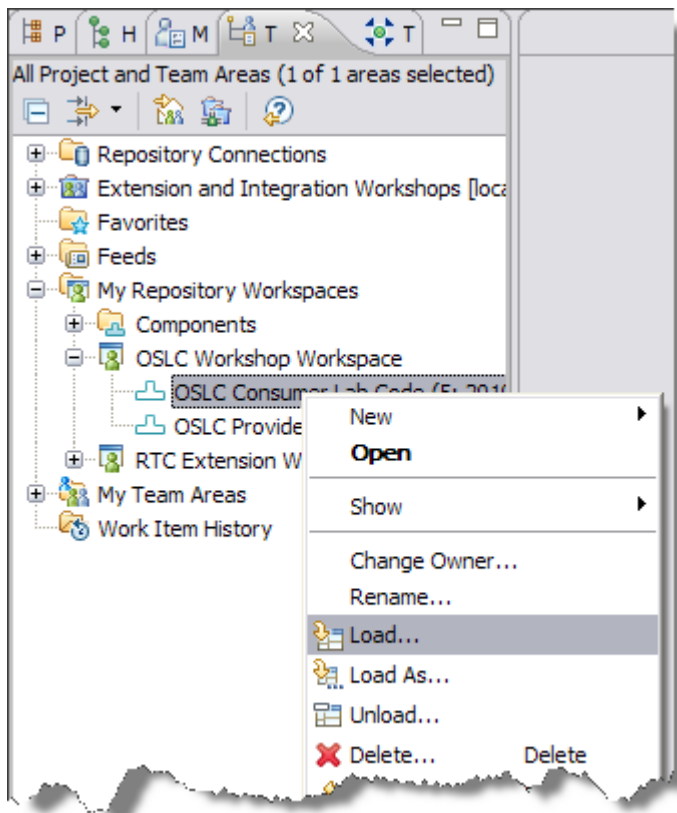
- __b. In the **Accept Team Invitation** wizard, enter the following in the text field and then click **Finish**.

```
teamRepository=https://localhost:9443/jazz
userId=ADMIN
userName=ADMIN
projectAreaName=Extension and Integration Workshops
```

- ___c. When prompted for a password, enter ADMIN. Also, check the **Save password** and **Automatically log in** check boxes. Then click **OK**.

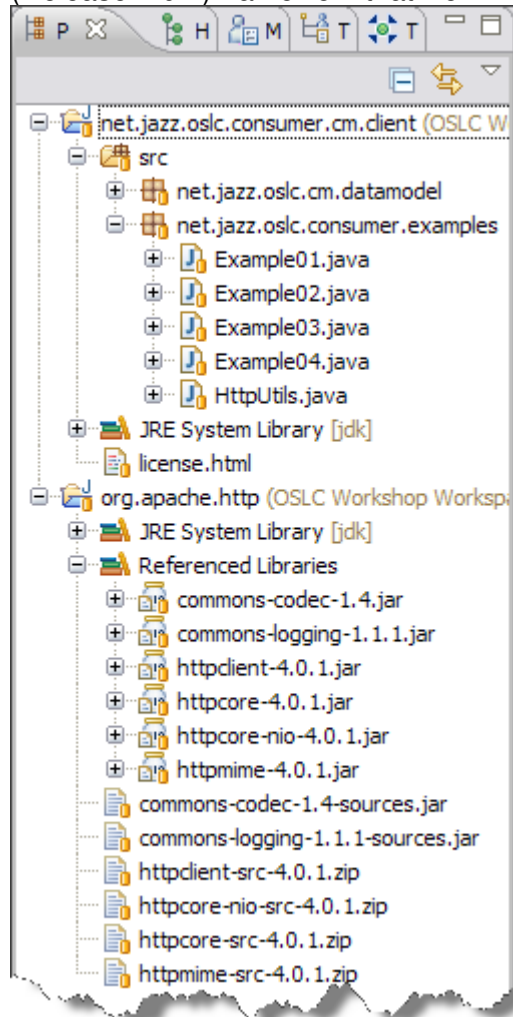


- ___d. If prompted with a **Repository Connection Certificate Problem**, select the **Accept this certificate permanently** radio button and then click **OK**.
- ___e. Close the project area editor that opens.
- ___f. In the **Team Artifacts** view, expand the **My Repository Workspaces** node, expand the **OSLC Workshop Workspace**, right click on the **OSLC Consumer Lab Code** component and then select the **Load...** action from then context menu.



In the **Load Repository Workspace** wizard, click **Finish**.

- __g. Verify that there are now at least 2 Eclipse projects in your **Package Explorer** view. These projects contain the code for the **OSLC consumer lab**. In this lab we will mainly work with the `net.jazz.oslc.consumer.cm.client` Eclipse project. This project contains a set of samples we will explain and run during this lab. The other project `org.apache.http` contains the jars of the [Apache HTTP Client](#) (Release 4.0.1) framework that we will use to reach the REST API.



Apache HTTP Client

All our examples are based on Apache HTTP Client 4.0.1 API. This is not a prerequisite. Feel free to adopt the HTTP Client framework you prefer. We found this framework pretty convenient for our own needs.

3.2 Accessing to the Root Services document

This first example describes how to fetch the content of a URL, and more particularly, how to fetch the Root Services document using the Apache HTTP Client API.

1. In the **Package Explorer** view, expand the **src/net.jazz.oslc.consumer.examples** source package of the **net.jazz.oslc.consumer.cm.client** Eclipse project and then double click the **Example01.java** file. The following snippet of code is extracted from the `main` method.

```
// Setup the HttpClient
HttpClient httpClient = new DefaultHttpClient();

// Disabling SSL Certificate Validation
HttpUtils.setupLazySSLSupport(httpClient);

// Setup the HTTP GET method
HttpGet httpget = new HttpGet("https://localhost:9443/jazz/rootservices");

HttpResponse response;
try {
    // Execute the request
    response = httpClient.execute(httpget);
    System.out.println(">> HTTP Status code: " + response.getStatusLine());

    if (response.getStatusLine().getStatusCode() == 200) {
        System.out.println(">> HTTP Response Headers: ");
        HttpUtils.printResponseHeaders(response);

        System.out.println(">> HTTP Response Body: ");
        HttpUtils.printResponseBody(response);
    } else {
        // Release allocated resources
        response.getEntity().consumeContent();
    }
} catch (ClientProtocolException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} finally {
    // Shutdown the HTTP connection
    httpClient.getConnectionManager().shutdown();
}
```

- __2. To get access to the **Apache HTTP Client API**, for executing an HTTP method, we need to create an instance of **org.apache.http.impl.client.DefaultHttpClient**:

```
// Setup the HttpClient
HttpClient httpClient = new DefaultHttpClient();
```

- __3. The Jazz Team Server uses the SSL (Secure Socket Layer) protocol. If we try to access any HTTPS URL, we will get an SSL certificate exception. It is for this reason that the client needs to specify how he wants to handle the certificates.
For the purpose of the demo, we have defined some code which disables the certificate validation by overwriting the default behavior to trust any certificate.

```
// Disabling SSL Certificate Validation
HttpUtils.setupLazySSLSupport(httpclient);
```

This behavior is implemented by the **HttpUtils.setupLazySSLSupport** static method.

- __4. The next line creates an instance of **org.apache.http.client.methods.HttpGet** which refines the call to the HTTP GET method. The call is initialized with the URI of the Root Services document.

```
// Setup the HTTP GET method
HttpGet httpget = new HttpGet("https://localhost:9443/jazz/rootservices");
```

- __5. The next line sends/executes the GET. The response of the http method is returned in an instance of **org.apache.http.HttpResponse**.

```
// Execute the request
HttpResponse response = httpClient.execute(httpget);
```

- __6. The next line prints out the **status code** of the HTTP response

```
System.out.println(">> HTTP Status code: " + response.getStatusLine());
```

- __7. If the response is OK (status code = 200) then the code prints the response headers (**HttpUtils.printResponseHeaders**) and the response body (**HttpUtils.printResponseBody**).

```
System.out.println(">> HTTP Response Headers: ");
HttpUtils.printResponseHeaders(response);

System.out.println(">> HTTP Response Body: ");
HttpUtils.printResponseBody(response);
```

- __8. If the response is an error then the code releases any created resources:

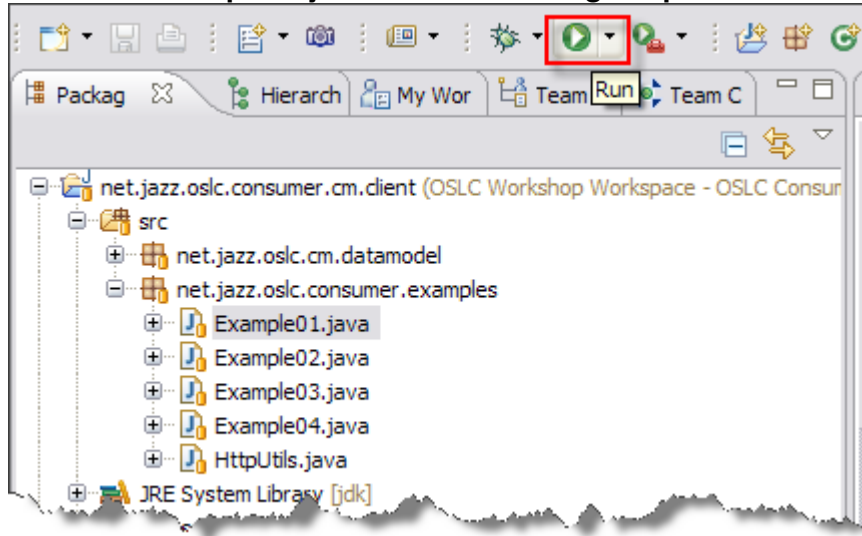
```
// Release allocated resources
response.getEntity().consumeContent();
```

- __9. Finally, the last line shuts down the HTTP client by releasing the connections.

```
// Shutdown the HTTP connection
httpClient.getConnectionManager().shutdown();
```

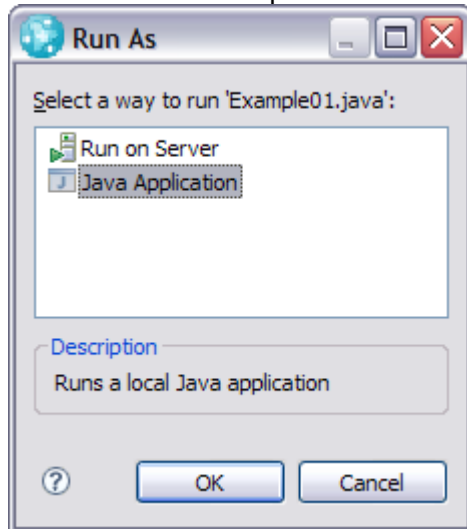
__10. Now that we have a good understanding of the code, lets run it.

__a. Select the **Example01.java** file in the **Package Explorer**:

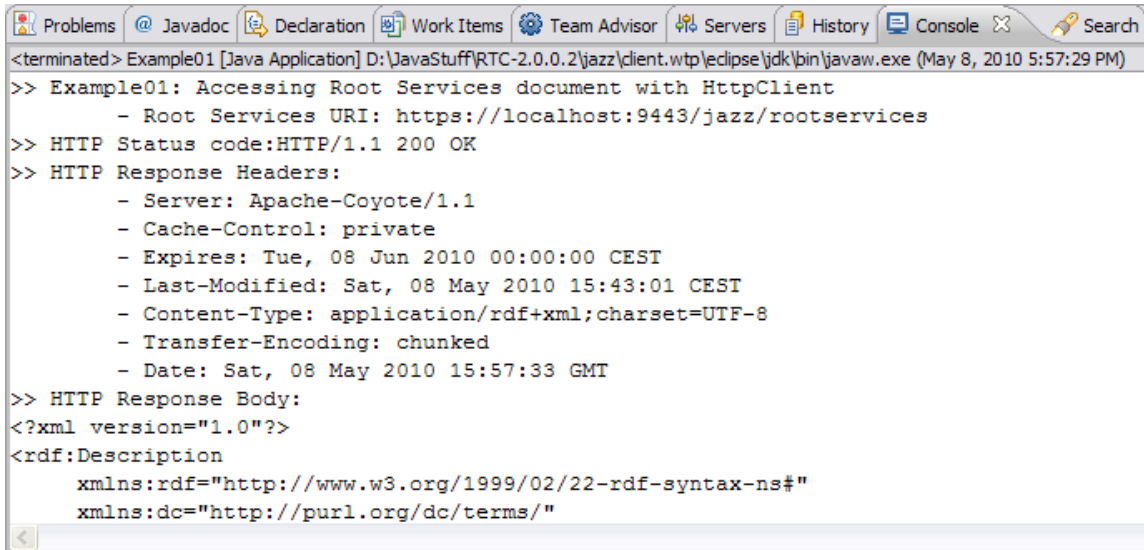


__b. Press the **Run as...** button located on toolbar

__c. Select run the example as a **Java Application** and press **OK**.



- ___d. The **Console view** will appear in the bottom part of the workbench displaying the example print out.



```
<terminated> Example01 [Java Application] D:\JavaStuff\RTC-2.0.0.2\jazz\client.wtp\eclipse\jdk\bin\javaw.exe (May 8, 2010 5:57:29 PM)
>> Example01: Accessing Root Services document with HttpClient
- Root Services URI: https://localhost:9443/jazz/rootservices
>> HTTP Status code:HTTP/1.1 200 OK
>> HTTP Response Headers:
- Server: Apache-Coyote/1.1
- Cache-Control: private
- Expires: Tue, 08 Jun 2010 00:00:00 CEST
- Last-Modified: Sat, 08 May 2010 15:43:01 CEST
- Content-Type: application/rdf+xml;charset=UTF-8
- Transfer-Encoding: chunked
- Date: Sat, 08 May 2010 15:57:33 GMT
>> HTTP Response Body:
<?xml version="1.0"?>
<rdf:Description
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/terms/"
```

The log should looklike this:

```
>> Example01: Accessing Root Services document with HttpClient
- Root Services URI: https://localhost:9443/jazz/rootservices
>> HTTP Status code: HTTP/1.1 200 OK
>> HTTP Response Headers:
- Server: Apache-Coyote/1.1
- Cache-Control: private
- Expires: Tue, 08 Jun 2010 00:00:00 CEST
- Last-Modified: Sat, 08 May 2010 15:43:01 CEST
- Content-Type: application/rdf+xml;charset=UTF-8
- Transfer-Encoding: chunked
- Date: Sat, 08 May 2010 15:04:03 GMT
>> HTTP Response Body:
<?xml version="1.0"?>
<rdf:Description
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/terms/"
  xmlns:jfs="http://jazz.net/xmlns/prod/jazz/jfs/1.0/"
  xmlns:jd="http://jazz.net/xmlns/prod/jazz/discovery/1.0/"
  xmlns:jp06="http://jazz.net/xmlns/prod/jazz/process/0.6/"
  rdf:about="https://localhost:9443/jazz/rootservices">
```

...

```
</rdf:Description>
```

3.3 Retrieve the Service Provider catalog using XPath

This new example shows how an OSLC consumer can retrieve an element or an attribute of an element in an XML representation, such the Root Services document.

Actually, this example uses the XPath language to retrieve the Service Provider catalog listed by the attribute `rdf:resource` of the element `oslc_cm:cmServiceProviders`.

The W3C XPath language (<http://www.w3.org/TR/xpath/>) has been defined for querying XML documents to select any node (element or attribute) or list of nodes. Here are few XPath expression examples:

Expression	Description
<code>foo</code>	Selects all the child nodes named <code>foo</code> .
<code>/foo</code>	Selects from the root node the nodes named <code>foo</code> .
<code>//foo</code>	Selects nodes named <code>foo</code> no matter where they are in the document
<code>@att</code>	Selects the attribute node named <code>att</code> .
<code>foo/bar</code>	Selects all the nodes named <code>bar</code> having a parent node named <code>foo</code> .
<code>//foo[@att]</code>	Select all the nodes named <code>foo</code> no matter where they are having an attribute named <code>att</code> .
<code>//foo [@att="val"]</code>	Select all the nodes named <code>foo</code> no matter where they are having an attribute named <code>att</code> with the value <code>val</code> .

For example, knowing that the Root Services document has the following tag structure:

```
<?xml version="1.0"?>
<rdf:Description ...>
.../...
<!-- Change Management service catalog -->
<oslc_cm:cmServiceProviders
  xmlns:oslc_cm="http://open-services.net/xmlns/cm/1.0/"
  rdf:resource="https://localhost:9443/jazz/oslc/workitems/catalog" />
.../...
</rdf:Description>
```

The XPath expression to retrieve the node defining the Service Provider catalog will be:

```
/rdf:Description/oslc_cm:cmServiceProviders/@rdf:resource
```

This expression means: “Select the attribute node named `rdf:resource` from the element node named `oslc_cm:cmServiceProviders`, child of the element named `rdf:Description`.”

Let see the code for the next example now...

__1. In the **Package Explorer** view, open the **Example02.java** file and look at the `main` method:

```
// Define the XPath evaluation environment for RTC XML documents
XPath xpath = HttpUtils.getRTCXPath();

// Parse the response body
InputStream source = new InputStream(response.getEntity().getContent());
Node attribute = (Node) (xpath.evaluate(
    "/rdf:Description/oslc_cm:cmServiceProviders/@rdf:resource",
    source, XPathConstants.NODE));

// Print out the Service Provider catalog URI
System.out.println(">> Catalog URI: " + attribute.getTextContent());
```

__2. The first line creates an instance of an XPath evaluation environment. This environment is set up to be able to parse and understand nodes using RTC namespaces.

```
// Define the XPath evaluation environment for RTC XML documents
XPath xpath = HttpUtils.getRTCXPath();
```

The **getRTCXPath** method is implemented in the **HttpUtils** class. It mainly consists of mapping the name space ID to their corresponding URL.

__3. The next lines parse the response body (`response.getEntity().getContent()`) using the `XPath.evaluate` method. This method takes 3 arguments:

```
// Parse the response body
InputStream source = new InputStream(response.getEntity().getContent());
Node attribute = (Node) (xpath.evaluate(
    "/rdf:Description/oslc_cm:cmServiceProviders/@rdf:resource",
    source, XPathConstants.NODE));
```

__a. The XPath expression to evaluate, describing the node(s) to select. In the example code, we provide the XPath to retrieve the URI of the Service Provider catalog:

```
"/rdf:Description/oslc_cm:cmServiceProviders/@rdf:resource"
```

__b. The source to parse.



This source can be either an `org.xml.sax.InputStream` or directly a DOM structure like an `org.w3c.dom.Document` or an `org.w3c.dom.Element`.

If you know that you will have to parse the same document several times, we recommend creating the DOM structure using your favorite SAX parser then reuse the DOM document each time you need it.

__c.

__d. The expected return type.



This return type can take 2 values:

- ❖ XPathConstants.NODE then the method will return the first Node found.
- ❖ XPathConstants.NODESET then the method will return a NodeList of all the nodes found.

In the example code, we are looking for the first attribute found.

__4. The last line prints out the value associated the found attribute. This value should be the Service Provider catalog URI:

```
// Print out the Service Provider catalog URI
System.out.println(">> Catalog URI: " + attribute.getTextContent());
```

__5. Now that we have a good understanding of the code, lets run it.

__a. Select the **Example02.java** file in the **Package Explorer**:

__b. Run the sample as a Java application

__c. The Console view will print out the catalog URI:.

```
<terminated> Example02 [Java Application] D:\JavaStuff\RTC-2.0.0.2\jazz\client.wtp\ eclipse\jdk\bin\javaw.exe (May 8, 2010 7:13:28 PM)
>> Example02: Retrieving the Service Provider catalog URI with XPath
- Root Services URI: https://localhost:9443/jazz/rootservices
- Service Provider XPath expression: /rdf:Description/oslc_cm:cmServiceProviders/@rdf:resource
>> HTTP Status code:HTTP/1.1 200 OK
>> Catalog URI: https://localhost:9443/jazz/oslc/workitems/catalog
```

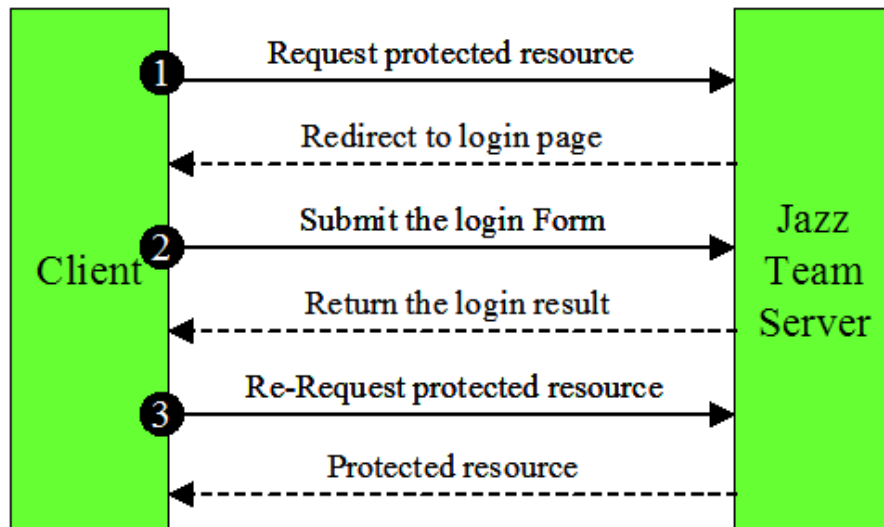
3.4 Jazz Form-based Authentication

In the next example we will see how to authenticate and pass the Jazz Team Server (JTS) security mechanisms defined by the foundation core services. Then we should be able to reach any protected document like the Service Provider catalog document.

This example prints out the titles of each Service Provider (alias Project Area) stored in the JTS we are connected to.

Contrary to the Root Services document, the Service Provider catalog document is a protected document, so the client needs to authenticate with the JTS to be able to access it.

JTS uses a **Form-Based Authentication**. This authentication has to go thru three steps:



- __1. The client requests a protected resource.
- __2. If the client is not authenticated, the server responds a redirect to the login page, and the client has to fill the form and submit it to the server.
- __3. If the login has succeeded, the client submits a request the protected resource again and should get it back.

This behavior is implemented by the `sendGetForSecureDocument` method stored in the **HttpUtils** class. If you don't want to dig into this implementation, feel free to directly skip to step (7).

- __4. In the **Package Explorer** view, open the **HttpUtils.java** file and scroll to the `sendGetForSecureDocument` method. To simplify the code, we have removed all the printouts from the snipped code.

```

HttpGet documentGet = new HttpGet(protectedResource);
documentGet.addHeader("accept", mediaType);

// Step (1): Request the protected resource
HttpResponse documentResponse = httpClient.execute(documentGet);

if (documentResponse.getStatusLine().getStatusCode() == 200) {
    Header header = documentResponse.getFirstHeader(
        "x-com-ibm-team-repository-web-auth-msg");

    if ((header!=null) && ("authrequired".equals(header.getValue()))) {
        documentResponse.getEntity().consumeContent();
        // The server requires an authentication: Create the login form
        HttpPost formPost = new HttpPost(serverURI+"/j_security_check");
        List<NameValuePair> nvps = new ArrayList<NameValuePair>();
        nvps.add(new BasicNameValuePair("j_username", login));
        nvps.add(new BasicNameValuePair("j_password", password));
        formPost.setEntity(new UrlEncodedFormEntity(nvps, HTTP.UTF_8));
    }
}
  
```

```

// Step (2): The client submits the login form
HttpResponse formResponse = httpClient.execute(formPost);

header = formResponse.getFirstHeader(AUTHREQUIRED);
if ((header!=null) && ("authfailed".equals(header.getValue())) {
    // The login failed
    throw new InvalidCredentialsException("Authentication failed");
} else {
    formResponse.getEntity().consumeContent();
    // The login succeed

    // Step (3): Request again the protected resource
    HttpGet documentGet2 = new HttpGet(protectedResource);
    documentGet2.addHeader("accept", mediaType);
    return httpClient.execute(documentGet2);
}
}
}
return documentResponse;

```

__5. For the first step, as for any other document, the client tries to reach the document:

```

HttpGet documentGet = new HttpGet(protectedResource);
documentGet.addHeader("accept", mediaType);

```

// Step (1): Request the protected resource

```

HttpResponse documentResponse = httpClient.execute(documentGet);

```

__6. If the request didn't return any error, the client checks out if an authentication is required. This check will consist in verifying the presence of the `x-com-ibm-team-repository-web-auth-msg` HTTP response header. If the value of this header is `authrequired` then the client must submit a form-based login (https://jazz.net/wiki/bin/view/Main/JFSCoreSecurity#User_Authentication). if the authentication is not required, the client returns the HTTP response.

```

if (documentResponse.getStatusLine().getStatusCode() == 200) {
    Header header
        = documentResponse.getFirstHeader("x-com-ibm-team-repository-web-auth-msg");

    if ((header!=null) && ("authrequired".equals(header.getValue())) {
        ...
    } else {
        return documentResponse;
    }
}

```

__7. The next step consists of filling in and POSTing the authentication form:

```

// The server requires an authentication: Create the login form
HttpPost formPost = new HttpPost(serverURI+"/j_security_check");
List<NameValuePair> nvps = new ArrayList<NameValuePair>();
nvps.add(new BasicNameValuePair("j_username", login));

```

```

nvps.add(new BasicNameValuePair("j_password", password));
formPost.setEntity(new UrlEncodedFormEntity(nvps, HTTP.UTF_8));
// Step (2): The client submits the login form
HttpResponse formResponse = httpClient.execute(formPost);

```

- __8. Then the client needs to check out the result of the login. If the login failed then the client should throw an exception:

```

header = formResponse.getFirstHeader("x-com-ibm-team-repository-web-auth-msg ");
if ((header!=null) && ("authfailed".equals(header.getValue()))) {
    // The login failed
    throw new InvalidCredentialsException("Authentication failed");
}

```

- __9. If the login didn't fail, then the client can request the protected document a second time, and should receive the expected response:

```

// Step (3): Request again the protected resource
HttpGet documentGet2 = new HttpGet(protectedResource);
documentGet.addHeader("accept", mediaType);
return httpClient.execute(documentGet2);

```

At this point, we should be able to understand the third example.

- __10. In the **Package Explorer** view, open the **Example03.java** file and look at the `main` method:

```

// Access to the Service Provider catalog
HttpResponse catalogResponse
    = HttpUtils.sendGetForSecureDocument(
        server, serviceProvidersCatalog,
        "application/x-oslc-disc-service-provider-catalog+xml",
        login, password, httpClient);

if (catalogResponse.getStatusLine().getStatusCode() == 200) {
    source = new InputSource(catalogResponse.getEntity().getContent());
    NodeList titleNodes = (NodeList) (xpath.evaluate(
        serviceProviderTitleXPath,
        source, XPathConstants.NODESET));

    // Print out the title of each Service Provider
    int length = titleNodes.getLength();
    System.out.println(">> Project Areas:");
    for (int i = 0; i < length; i++) {
        System.out.println(">> \t - "+ titleNodes.item(i).getTextContent());
    }
}

```

- __a. Once the client has retrieved the URI of the Service Provider catalog (`serviceProvidersCatalog`), it can fetch the catalog. Because the catalog is a protected document, the client uses the Form Based authentication code we have previously described. The associated Media Type is `application/x-oslc-disc-service-provider-catalog+xml`².

```
// Access to the Service Provider catalog
HttpResponse catalogResponse
    = HttpUtils.sendGetForSecureDocument(
        server, serviceProvidersCatalog,
        "application/x-oslc-disc-service-provider-catalog+xml",
        login, password, httpClient);
```

- __b. If the server didn't return an error, then the client can parse the response body and extract from the Service Provider catalog document the title nodes of each Service Provider (alias Project Area) and print out the Service Provider title:

```
source = new InputSource(catalogResponse.getEntity().getContent());
NodeList titleNodes = (NodeList) (xpath.evaluate( "//*[oslc_disc:ServiceProvider/dc:title",
        source, XPathConstants.NODESET));

// Print out the title of each Service Provider
int length = titleNodes.getLength();
System.out.println(">> Project Areas:");
for (int i = 0; i < length; i++) {
    System.out.println(">> \t - " + titleNodes.item(i).getTextContent());
}
```

The developer knows that the Service Provider catalog document has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<oslc_disc:ServiceProviderCatalog ...>
  <dc:title>Project Areas</dc:title>
  <oslc_disc:entry>
    <oslc_disc:ServiceProvider>
      <dc:title>Extension and Integration Workshops</dc:title>
      <oslc_disc:details rdf:resource="..."/>
      <oslc_disc:services rdf:resource=".../workitems/services.xml"/>
      <jp:consumerRegistry rdf:resource=".../links"/>
    </oslc_disc:ServiceProvider>
  </oslc_disc:entry>
</oslc_disc:ServiceProviderCatalog>
```

Therefore the XPath expression to retrieve the `dc:title` nodes of the Service Provider could be:

```
/oslc_disc:ServiceProviderCatalog/oslc_disc:entry/oslc_disc:ServiceProvider/dc:title
```

² http://open-services.net/bin/view/Main/CmRestApiV1#Media_Types_Used

We could also simplify this expression with the following XPath expression:

```
//oslc_disc:ServiceProvider/dc:title
```

This expression means: select all the `oslc_disc:ServiceProvider` nodes, no matter where they are, then select their `dc:title` child node. In this particular case, we could even simplify to the following XPath expression:

```
//dc:title
```

Actually, the above expression means: select all the `dc:title` nodes no matter where they are.

- __11. Now that we have a good understanding of the code, lets run it.
 - __a. Select the **Example03.java** file in the **Package Explorer**:
 - __b. Run it as Java application.
 - __c. The Console view will print out the titles of the Project Areas currently stored in the Jazz Team Server:

```
<terminated> Example03 [Java Application] D:\JavaStuff\RTC-2.0.0.2\jazz\client.wtp\eclipse\jdk\bin\javaw.exe (May 9, 2010 3:07:42 PM)
>> Example03: Print out the content of the Service Providers catalog
- Root Services URI: https://localhost:9443/jazz/rootservices
- Service Providers catalog XPath expression: /rdf:Description/oslc_cm:cmService
- Service Provider title XPath expression: //oslc_disc:ServiceProvider/dc:title
- Login: ADMIN
- Password: ADMIN
>> GET (1) https://localhost:9443/jazz/oslc/workitems/catalog
>> Response Headers:
- Server: Apache-Coyote/1.1
- Cache-Control: private
- Expires: Thu, 01 Jan 1970 01:00:00 CET
- Set-Cookie: JazzFormAuth=Form; Path=/jazz
- X-com-ibm-team-repository-web-auth-msg: authrequired
- Last-Modified: Sun, 09 May 2010 13:07:46 GMT
- Content-Type: text/html;charset=utf-8
- Content-Length: 2358
- Date: Sun, 09 May 2010 13:07:46 GMT
>> POST https://localhost:9443/jazz/j_security_check
- Server: Apache-Coyote/1.1
- Location: https://localhost:9443/jazz/authenticated/identity?redirectPath=%2F
- Content-Length: 0
- Date: Sun, 09 May 2010 13:07:48 GMT
>> GET (2) https://localhost:9443/jazz/oslc/workitems/catalog
>> Project Areas:
>> - Extension and Integration Workshops
```

3.5 Work Item update

This last example describes how to retrieve an existing Change Request (alias Work Item), modify it and store it back in the server.

- __1. In the **Package Explorer** view, open the **Example04.java** file and scroll to the `run` method. To simplify the reading of the code, we have split the code into a set of methods describing all of the steps to retrieve a Change Request, fetch it, modify it, and finally store it back.

```

// Step (1) : Retrieve the Service Provider catalog
String catalogURI = getServiceProviderCatalog();
System.out.println(">> Service Provider Catalog: "+catalogURI);

// Step (2) : Retrieve the designated Service Provider (Project Area)
String paName = "Extension and Integration Workshops";
String projectAreaURI = getServiceProvider(catalogURI, paName);
System.out.println(">> Project Area ["+paName+"]: "+projectAreaURI);

// Step (3) : Retrieve the Change Request Simple Query for the current Service Provider
String simpleQueryURI = getSimpleQueryURI(projectAreaURI);
System.out.println(">> Simple Query URL: "+simpleQueryURI);

// Step (4) : Retrieve the designated Change Request (Work Item)
String wiID = "1";
ChangeRequest cr = getChangeRequest(simpleQueryURI, wiID);
System.out.println(">> Change Request URL for ["+wiID+"]: "+cr.getUri());

// Step (5) : Apply modification to the current Change Request
cr.setDcDescription(cr.getDcDescription()+" - " + new Date().toString());

// Step (6) : Update the Change Request on the server
HttpResponse response = updateChangeRequest(cr);

// Step (7) : Print out the HTTP PUT method response
System.out.println(">> Update Response Status code:" + response.getStatusLine());
System.out.println(">> Update Response Headers:");
HttpUtils.printResponseHeaders(response);
System.out.println(">> Update Response Body:");
HttpUtils.printResponseBody(response);

```

- __2. Step (4): We will not spend time on 3 first steps which have been explained during the previous examples. The step (4) is interesting because the code not only fetches a Change Request resource but it also maps the XML representation to a Java Object representation, which is an instance of the `net.jazz.oslc.cm.datamodel.ChangeRequest`.
- __a. Actually, this method queries the designated Change Request, fetching only the subset of properties supported by the ChangeRequest implementation (`dc:title`, `dc:identifier`, `dc:type`, `dc:description`, `dc:subject`, `dc:creator`, `dc:modified`):

```

// Build the query requesting a change request with a specific dc:identifier
// Fetch only a subset of its properties
String queryWIs
    = simpleQueryURI
      + "?oslc_cm.query="
        + URLEncoder.encode("dc:identifier=\""+wiID+"\"", HTTP.UTF_8)
      + "&oslc_cm.properties="
        + "dc:title,dc:identifier,dc:type,dc:description, dc:subject,dc:creator,dc:modified";

```

```
HttpResponse response = HttpUtils.sendGetForSecureDocument(
    server, queryWIs, "application/xml",
    login, password, httpClient);
```

- ___b. Then the client extracts from the HTTP response the `org.w3c.dom.Node` representing the Change Request

```
// Extract the Change Request DOM node
String wiXPath = "//oslc_cm:ChangeRequest";
InputSource source = new InputSource(response.getEntity().getContent());
Element wiNode = (Element)(xpath.evaluate(wiXPath, source, XPathConstants.NODE));
```

- ___c. Finally, the client instantiates a new Change Request based on the content of the Node, and returns the resulting ChangeRequest instance.

```
// Create the corresponding ChangeRequest instance
String wiURI = wiNode.getAttribute("rdf:resource");
return new ChangeRequest(wiURI, wiNode);
```

- ___3. Step (5): During this step, the client modifies the Change Request using the provided API. Actually, it concatenates the current timestamp at the end of the description.

```
// Step (5) : Apply modification to the current Change Request
cr.setDcDescription(cr.getDcDescription()+" - " + new Date().toString());
```

- ___4. Step (6): In this step, the client uses the Update Change Request OSLC-CM API to update the modified Change Request. This API is described by the specs like this ([Change Management REST API - Update a change request](#)):

Update a change request

PUT {CR URI}

Update the referenced change request resource with the **request's body**. The service MUST support **application/x-oslc-cm-change-request+xml** and **application/x-oslc-cm-change-request+json** for the request's content body.

A service provider MUST support at least the following HTTP response status codes:

Status Code	Response Content	Description
200 OK	<HTTP-Header> Location: {CR URI} Content-body: updated CR	Once the change request is updated with the supplied changed, the resulting change request representation is returned

Which means that to update the Change Request, the client needs to send an **HTTP PUT** message with the Change Request URI, the `content-type` header must be set to `application/x-oslc-cm-change-request+xml` and the request's body must contain the XML representation of the modified Change Request.

This behavior is implemented by the `updateChangeRequest`:

```
// How to fill the request body (Content Producer)
ContentProducer cp = new ContentProducer() {
    public void writeTo(OutputStream outstream) throws IOException {
        Writer writer = new OutputStreamWriter(outstream, HTTP.UTF_8);
        cr.writeXML(writer);
        writer.flush();
    }
};
// Call the PUT method against the Change Request URI
return HttpUtils.sendPutForSecureDocument(
    server, cr.getUri(),
    cp, "application/x-oslc-cm-change-request+xml",
    login, password, httpClient);
```

If you dig into the `HttpUtils.sendPutForSecureDocument` method, you will notice that this method implements the same form-based authentication pattern as for the `sendGetForSecureDocument` method previously described.

- __5. Step (7): This step prints out the response of the HTTP PUT. So, let run the example and check out the results.
 - __a. Select the **Example04.java** file in the **Package Explorer**:
 - __b. Run the example as a Java application.
 - __c. The Console view should provide the following output:

```
<terminated> Example04 [Java Application] D:\JavaStuff\RTC-2.0.0.2\jazz\client.wtp\ eclipse\jdk\bin\javaw.exe (May 11, 2010 6:37:03 PM)
>> Example04: Update a Change Request
  - Server: https://localhost:9443/jazz
  - Login: ADMIN
  - Password: ADMIN
>> Service Provider Catalog: https://localhost:9443/jazz/oslc/workitems/catalog
>> Project Area [Extension and Integration Workshops]: https://localhost:9443/jazz/oslc/contexts/_68EfMFEd-SYLn-ohNfPg/workitems/ser
>> Simple Query URL: https://localhost:9443/jazz/oslc/contexts/_68EfMFEd-SYLn-ohNfPg/workitems
>> Change Request URL for [1]: https://localhost:9443/jazz/resource/itemOid/com.ibm.team.workitem.WorkItem/_8IXTMFErEd-HkIpaf-HDDR
>> Update Response Status code:HTTP/1.1 204 No Content
>> Update Response Headers:
  - Server: Apache-Coyote/1.1
  - Cache-Control: max-age=0, must-revalidate
  - Expires: Tue, 11 May 2010 16:37:13 GMT
  - Date: Tue, 11 May 2010 16:37:13 GMT
>> Update Response Body:
```

Don't be surprised by the Status Line - 204 No Content - and the empty response body. Actually, based on the specs, we should expect a status line set to "200 OK" and a response body with the update Change Request. This issue has been identified and fixed in the release 3.0 M5: [Work Item #108069](#).

- __6. Copy the URL of the Work Item from the Console view. It should located after the label: `Change Request URL for [1]`.



- ___7. Open the **Firefox** internet browser by double-clicking the **Mozilla Firefox** shortcut on the **Windows Desktop**.
- ___8. Paste the copied URL into the navigation field of your browser and Press Enter.
- ___9. If you were not already logged in, the web UI will display the login dialog. Login with `ADMIN` as both **User ID** and **Password**.

- ___10. After the login, the Work Item WebUI editor will appear and you should be able to check out that the description has been changed with a timestamp at the end:



Conclusion

This last example concludes our two labs on how consuming OSLC-CM API. We hope this will help you feel more comfortable with the basics of OSLC, and encourage you to look at some of the advanced features.

Don't hesitate to join the <http://open-services.net> community and follow the different specification activities...

Lab 4 Implementing the OSLC APIs in a service provider



Lab Scenario

You have an assignment to extend an OSLC service provider to have additional capabilities. You will learn how to modify the OSLC provider implementation to provide additional dialogs and capabilities



If you have not done the setup, see OSLC Lab 1, “Setting up for OSLC Development”.

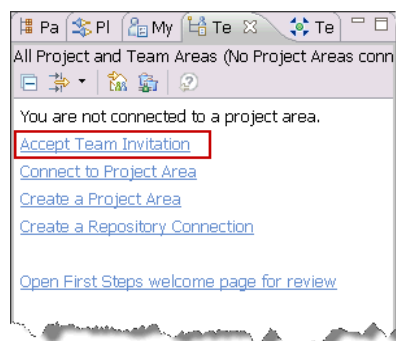
It is recommended that you complete OSLC Lab 2, “An introduction to the OSLC APIs” prior to this lab.



In order to complete and get the most out of this workshop, it is recommended that you are already familiar with RTC as a user. Of particular help would be familiarity with work items. In addition, you should have basic familiarity with Java programming and debugging using Eclipse. Note that OSLC can be used from any programming language that can invoke or provide web services and not just Java; however, the examples in this workshop are written in Java.

4.1 Loading Examples

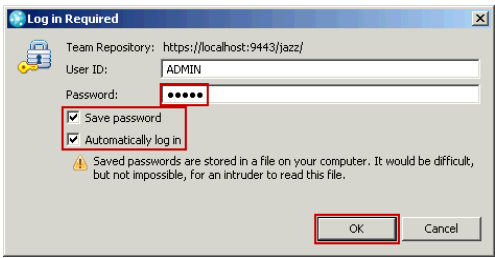
- __1. If the **Java** perspective is not open, open it now by selecting **Window > Open Perspective > Other... > Java** from the menu bar.
- __2. Load the lab code.
 - __a. On the left, switch to the **Team Artifacts** view and click the **Accept Team Invitation** link.



- ___ b. In the **Accept Team Invitation** wizard, enter the following in the text field and then click **Finish**.

```
teamRepository=https://localhost:9443/jazz  
userId=ADMIN  
userName=ADMIN  
projectAreaName=Extension and Integration Workshops
```

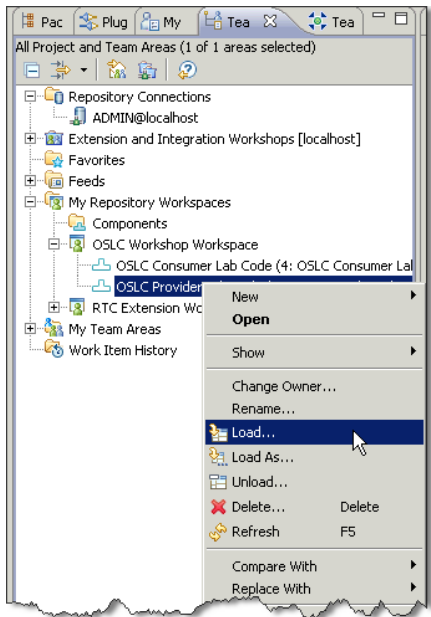
- ___ c. When prompted for a password, enter `ADMIN`. Also, check the **Save password** and **Automatically log in** check boxes. Then click **OK**.



- ___ d. If prompted with a **Repository Connection Certificate Problem**, select the **Accept this certificate permanently** radio button and then click **OK**.

- ___ e. Close the project area editor that opens.

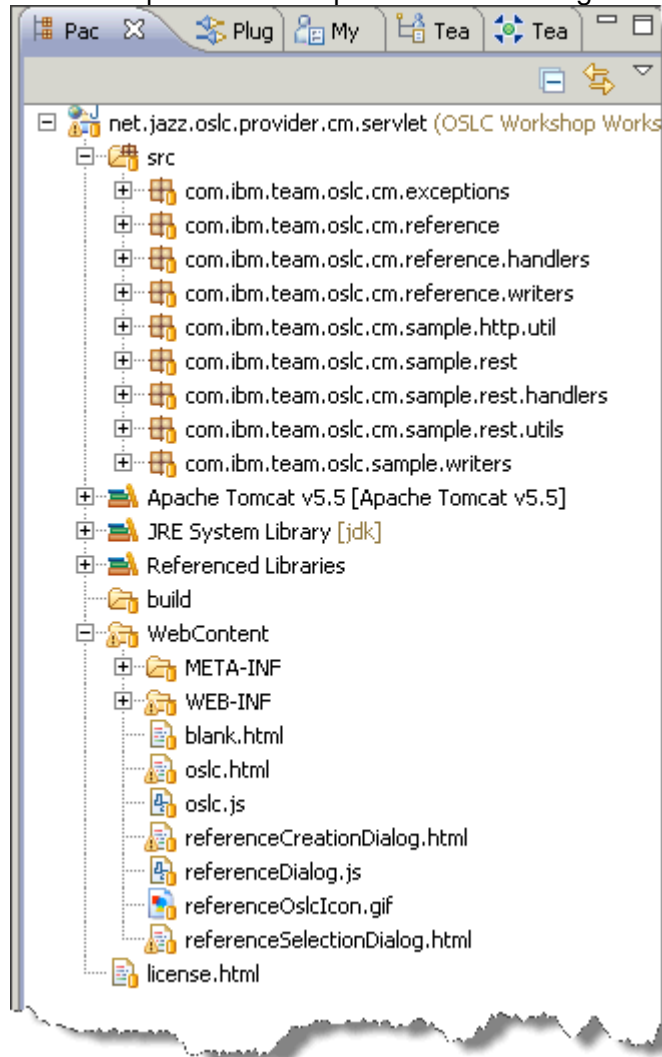
- ___ f. In the **Team Artifacts** view, expand the **My Repository Workspaces** node, expand the **OSLC Workshop Workspace**, right click on the **OSLC Provider Lab Code** component and then select the **Load...** action from then context menu.



Leave the default selection option **Find and load Eclipse projects**

In the **Load Repository Workspace** wizard, click **Finish**.

- __g. Verify that there are now at least 2 Eclipse projects in your **Package Explorer** view. These projects contain the code for the **OSLC Provider Lab**. In this lab we will mainly work with the `net.jazz.oslc.provider.cm.servlet` Eclipse project. This project contains a set of samples we will explain and run during this lab.

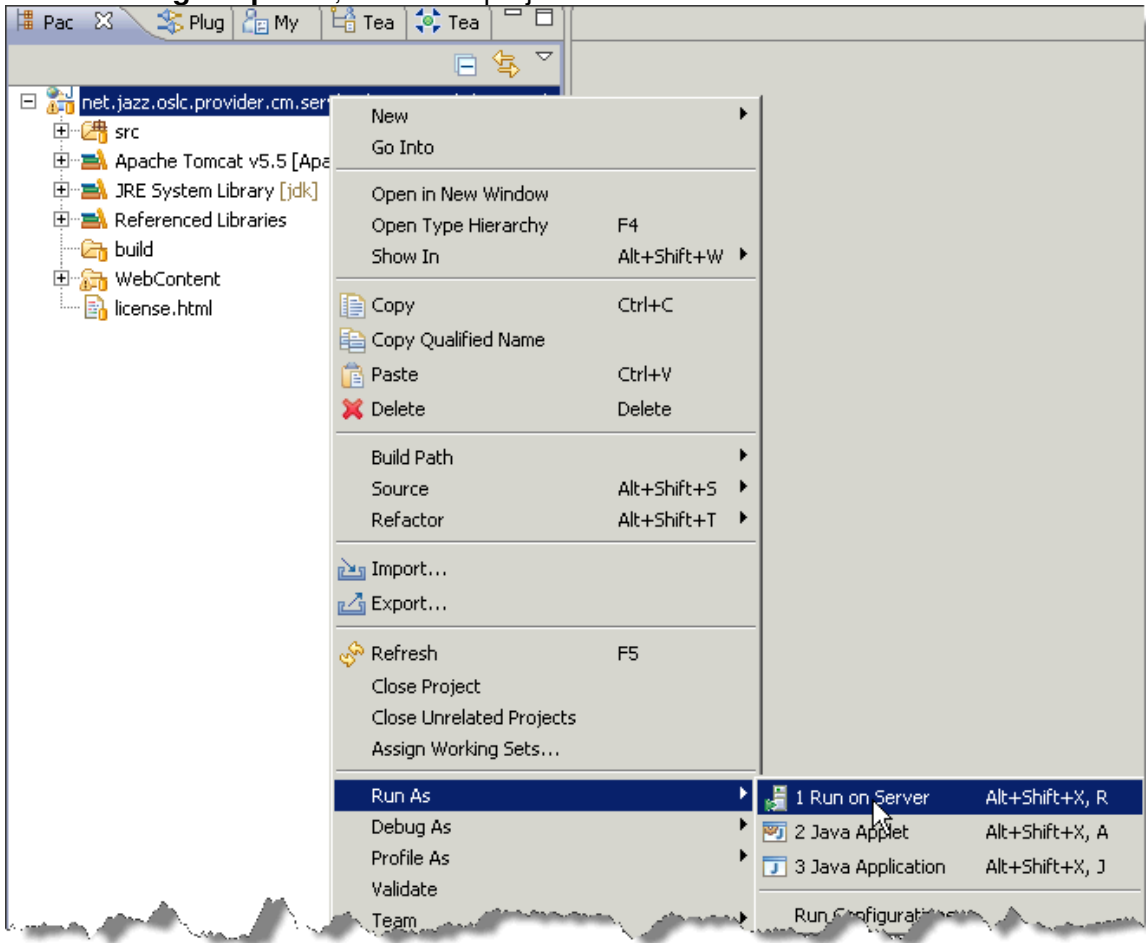


Apache Tomcat

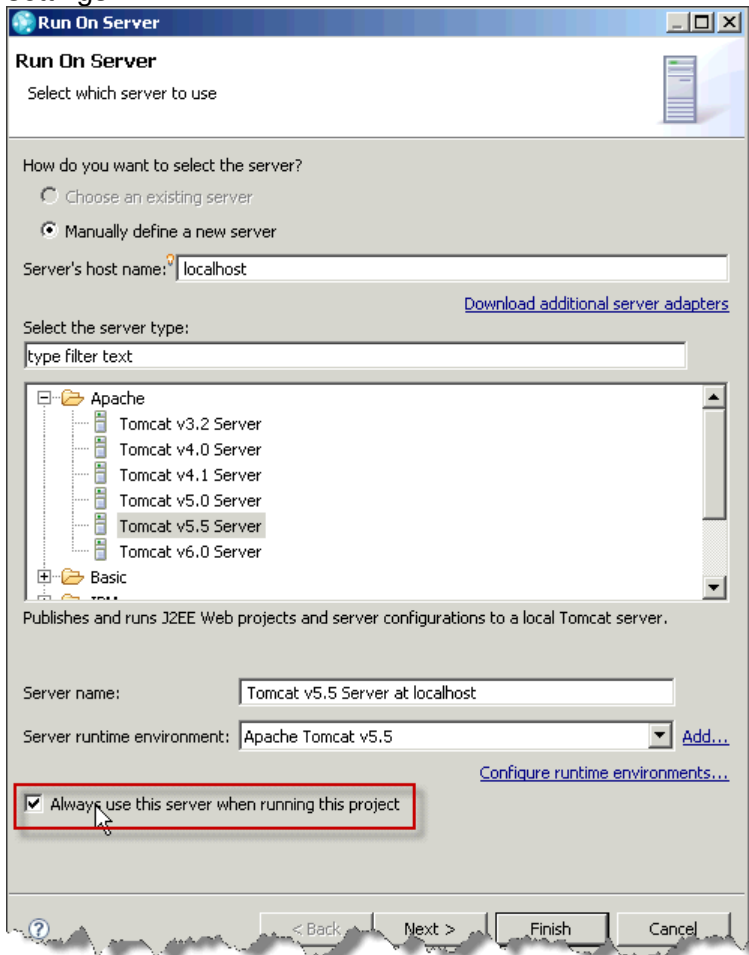
All our examples are based on Java Servlet technology and use Apache Tomcat Server as a test server. Other Java Servlet servers may be appropriate

4.2 Setting up the server runtime environment and running the sample server

3. In the **Package Explorer**, select the project name and then **Run->Run As->Run on Server**



- __4. Select **Always use this server when running this project** and leave all the other default settings.




On the **Run On Server** dialog select Finish.

The launcher will attempt to load a webpage in the embedded browser, when prompted select Cancel to download and close the embedded browser.

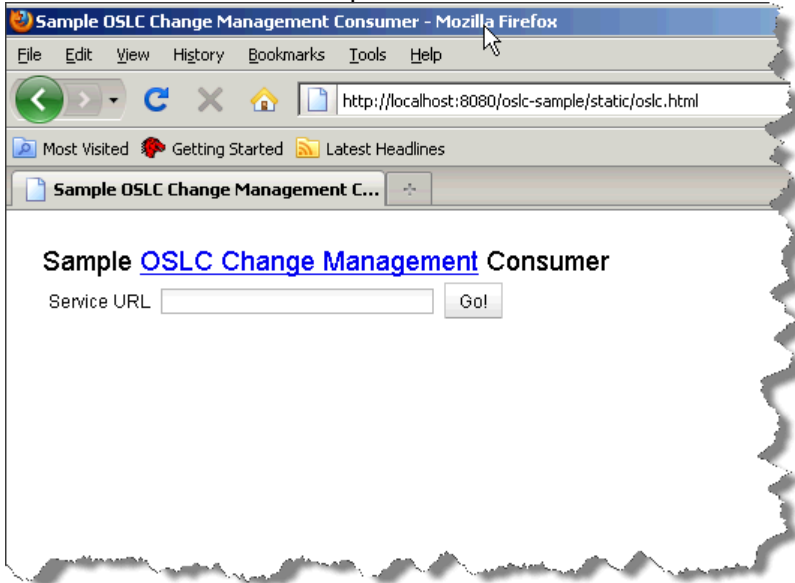


Restarting the sample server

This is a one time setup, for subsequent starts (or restarts) you will just need to select **Run on Server**

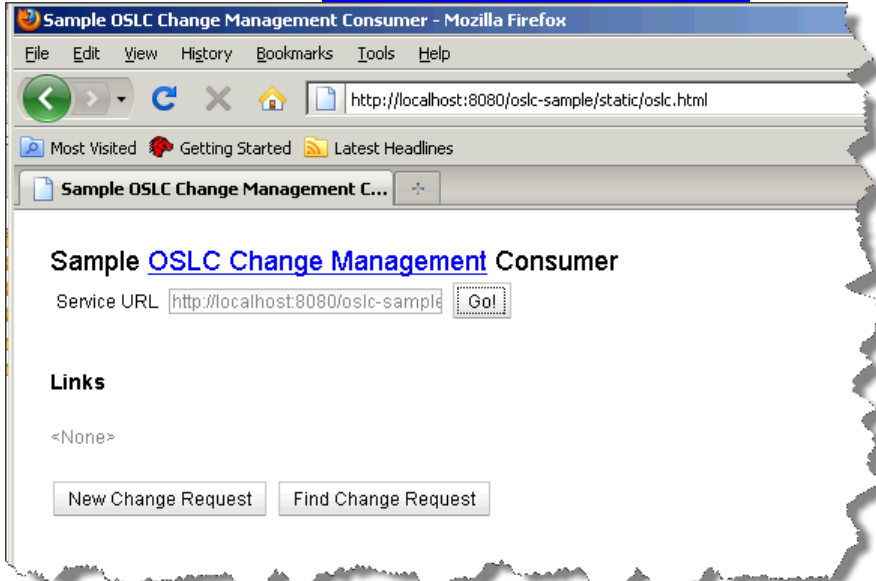
- __5. Open the **Firefox** internet browser by double-clicking the **Mozilla Firefox** shortcut  on the **Windows Desktop**.

- 6. Enter the URL: <http://localhost:8080/oslc-sample/static/oslc.html>
This URL will return a sample HTML OSLC-CM client consumer



4.3 Interacting with the sample provider

- 1. Enter the **Service URL**: <http://localhost:8080/oslc-sample> and select Go!



This has discovered the sample service provider document, used to enable the two buttons **New Change Request** and **Find Change Request**



HTML & Javascript sample

This simple sample highlights service discovery and access to delegated Web UI dialogs. It may be worth a few minutes to experiment with this. Nothing actually gets created on the server or deleted, reloading the page will reset to the beginning state.

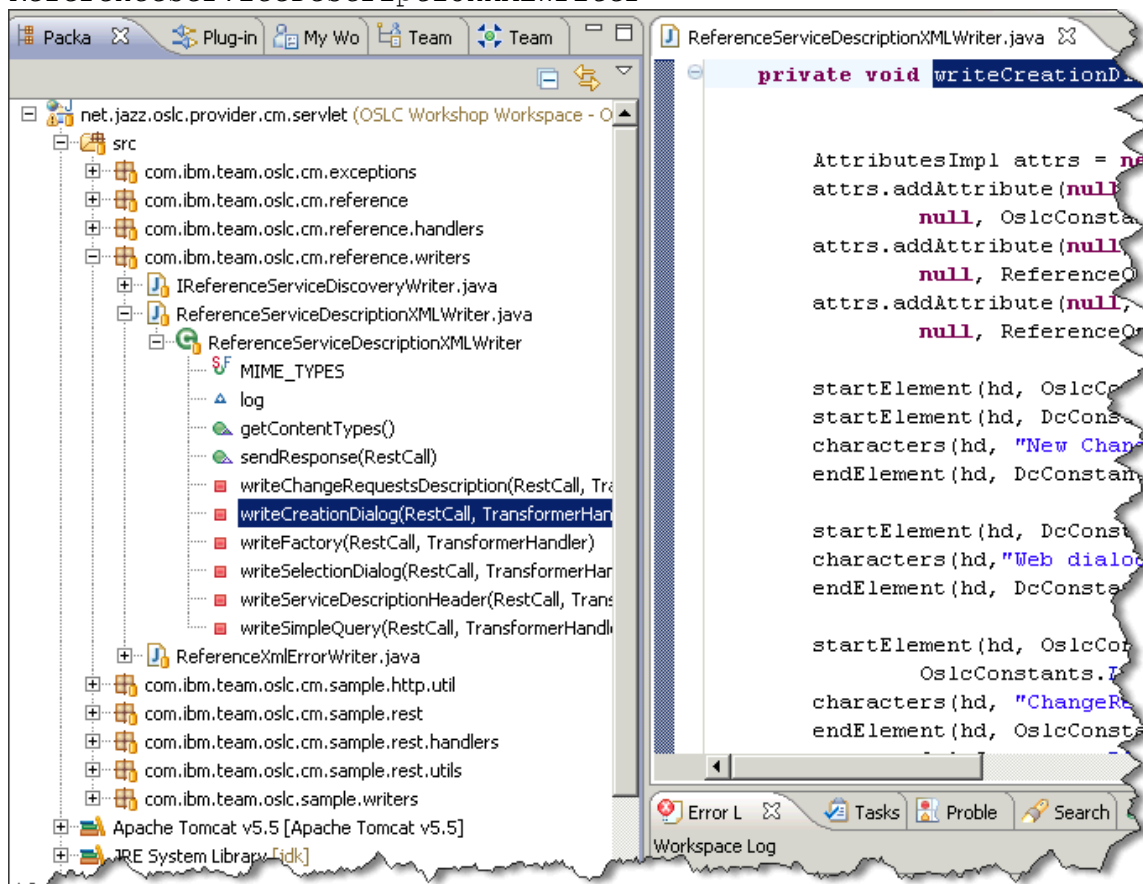
4.4 Modifying the provider, adding another dialog



Making code changes

You will become familiar with how to make code changes and test the changes. You will create a new entry in the service description document, a new creation dialog and make it the default

- ___1. Locate the source code to change from the **Package Explorer**, namely the `ReferenceServiceDescriptionXMLWriter`



- __2. Duplicate the function called **writeCreationDialog** and call the new function **writeCreationDialog2**
Use whatever method that works best for you.



Coding best practices

This sample doesn't focus on best practices to making some of these changes. It would be recommended to reuse the original method and refactor common aspects.

- __3. Update the function **writeChangeRequestDescription** to add call to this new function

```

ReferenceServiceDescriptionXMLWriter.java x
private void writeChangeRequestsDescription(RestCall restCall, TransformerHandler hd) {
    Map<String, String> map = new HashMap<String, String>();
    map.put(OslicConstants.VERSION_ATTR,
            OslicConstants.CHANGE_REQUESTS_VERSION);
    map.put(OslicConstants.CHANGE_REQUESTS_DOMAIN_ATTR,
            OslicConstants.CHANGE_MANAGEMENT_DOMAIN);
    startElement(hd, OslicConstants.CHANGE_REQUESTS_ELEMENT,
            createAttributes(hd, map));

    writeFactory(restCall, hd);

    writeSimpleQuery(restCall, hd);

    writeCreationDialog(restCall, hd);
    writeCreationDialog2(restCall, hd);

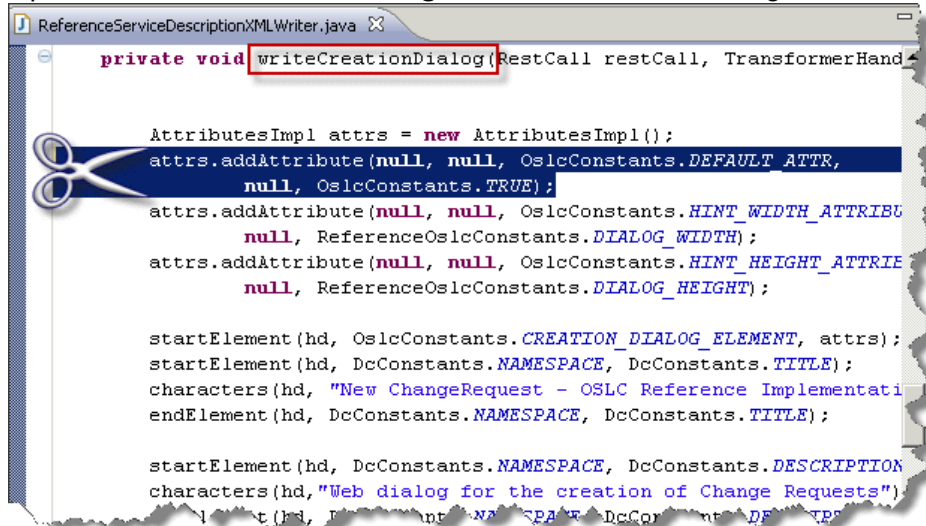
    writeSelectionDialog(restCall, hd);

    endElement(hd, OslicConstants.CHANGE_REQUESTS_ELEMENT);
}

private void writeSimpleQuery(RestCall restCall, TransformerHandler hd) {

```

4. Update the **writeCreationDialog** function to remove setting the default attribute to true.



```

private void writeCreationDialog(RestCall restCall, TransformerHandl

AttributesImpl attrs = new AttributesImpl();
attrs.addAttribute(null, null, OslcConstants.DEFAULT_ATTR,
null, OslcConstants.TRUE);
attrs.addAttribute(null, null, OslcConstants.HINT_WIDTH_ATTRIBUT
null, ReferenceOslcConstants.DIALOG_WIDTH);
attrs.addAttribute(null, null, OslcConstants.HINT_HEIGHT_ATTRIB
null, ReferenceOslcConstants.DIALOG_HEIGHT);

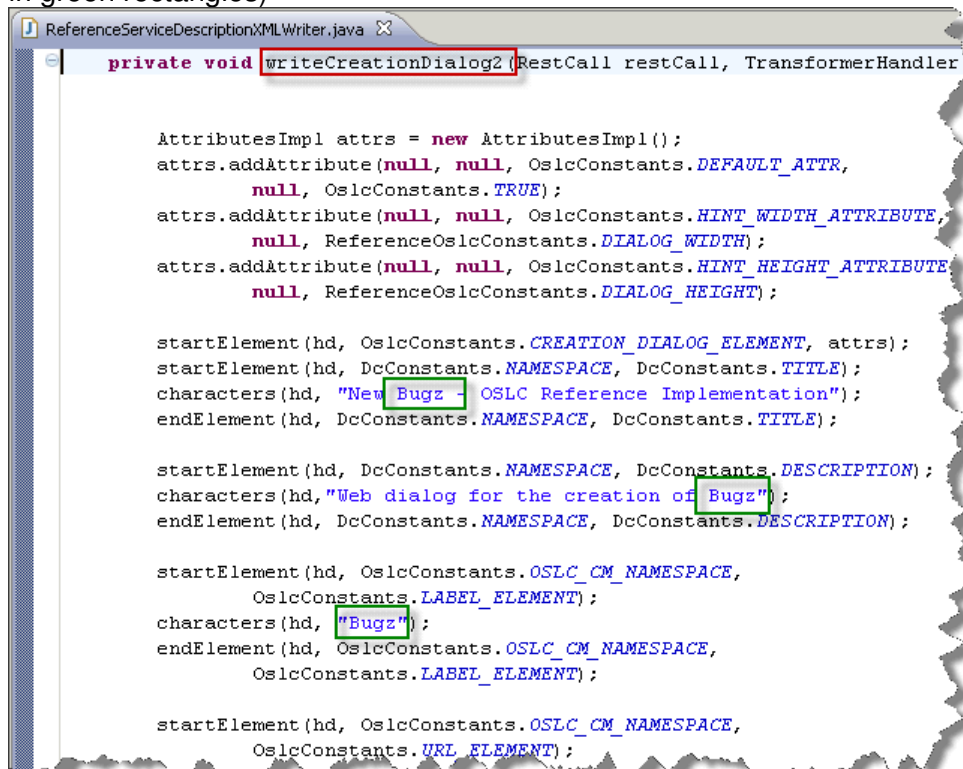
startElement(hd, OslcConstants.CREATION_DIALOG_ELEMENT, attrs);
startElement(hd, DcConstants.NAMESPACE, DcConstants.TITLE);
characters(hd, "New ChangeRequest - OSLC Reference Implementati
endElement(hd, DcConstants.NAMESPACE, DcConstants.TITLE);

startElement(hd, DcConstants.NAMESPACE, DcConstants.DESCRPTION
characters(hd, "Web dialog for the creation of Change Requests")
endElement(hd, DcConstants.NAMESPACE, DcConstants.DESCRPTION);

```

By doing this, the creation dialog entry in the service provider document will not longer be treated as the default dialog to use.

5. Update the **writeCreationDialog2** function to provide updated names on various labels (shown in green rectangles)



```

private void writeCreationDialog2(RestCall restCall, TransformerHandl

AttributesImpl attrs = new AttributesImpl();
attrs.addAttribute(null, null, OslcConstants.DEFAULT_ATTR,
null, OslcConstants.TRUE);
attrs.addAttribute(null, null, OslcConstants.HINT_WIDTH_ATTRIBUT
null, ReferenceOslcConstants.DIALOG_WIDTH);
attrs.addAttribute(null, null, OslcConstants.HINT_HEIGHT_ATTRIBUT
null, ReferenceOslcConstants.DIALOG_HEIGHT);

startElement(hd, OslcConstants.CREATION_DIALOG_ELEMENT, attrs);
startElement(hd, DcConstants.NAMESPACE, DcConstants.TITLE);
characters(hd, "New Bugz OSLC Reference Implementation");
endElement(hd, DcConstants.NAMESPACE, DcConstants.TITLE);

startElement(hd, DcConstants.NAMESPACE, DcConstants.DESCRPTION);
characters(hd, "Web dialog for the creation of Bugz");
endElement(hd, DcConstants.NAMESPACE, DcConstants.DESCRPTION);

startElement(hd, OslcConstants.OSLC_CM_NAMESPACE,
OslcConstants.LABEL_ELEMENT);
characters(hd, "Bugz");
endElement(hd, OslcConstants.OSLC_CM_NAMESPACE,
OslcConstants.LABEL_ELEMENT);

startElement(hd, OslcConstants.OSLC_CM_NAMESPACE,
OslcConstants.URL_ELEMENT);

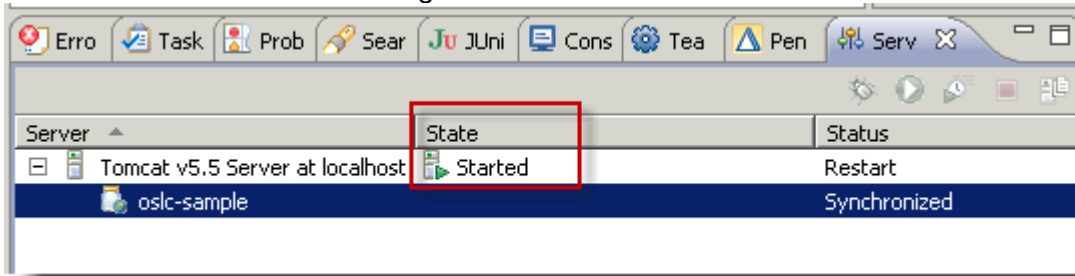
```

By leaving the setting of the attribute default to true, this will tell consumers to use this create dialog as the default.

Save the changes and validate there are no compilation errors. If there are errors, investigate the cause and repair

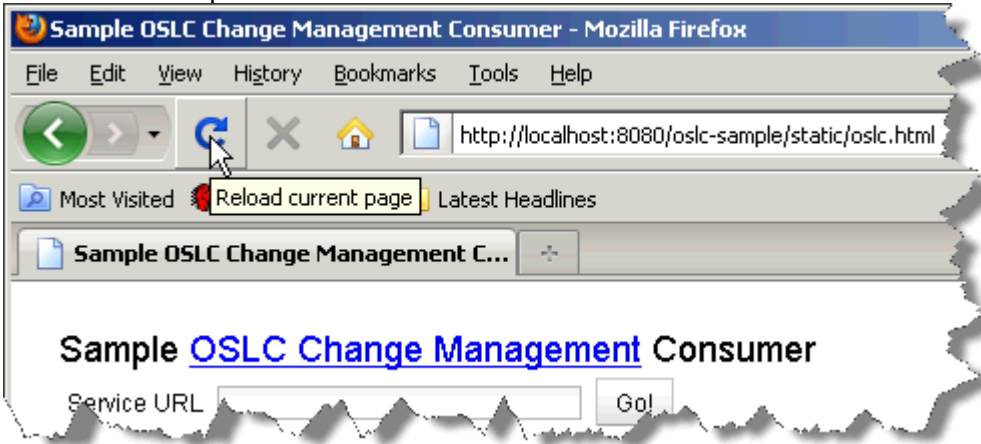
4.5 Test changes

- __1. Validate the server is still running



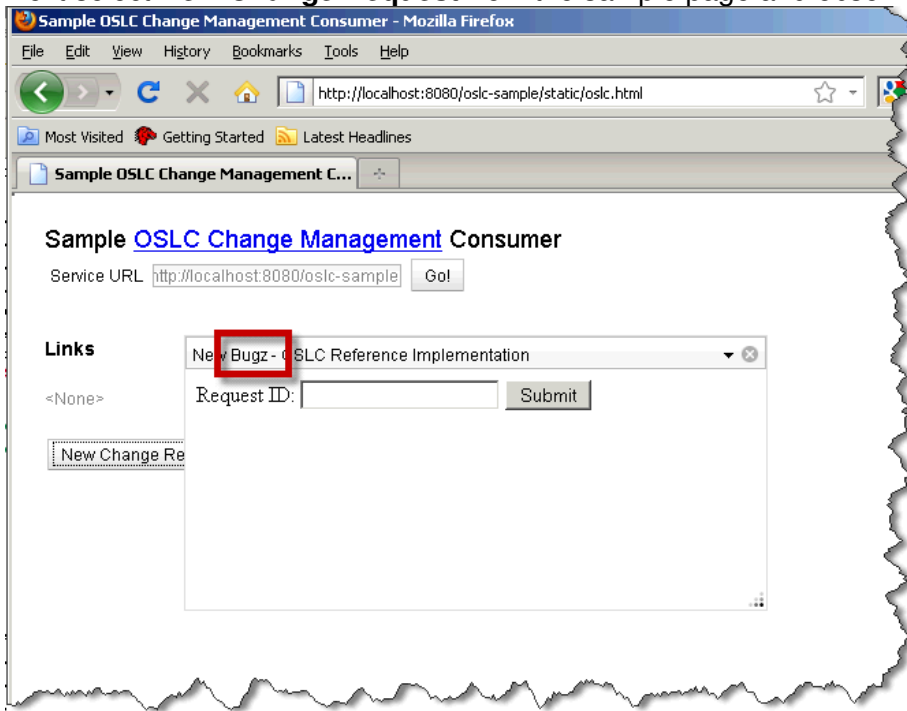
If the server is still running, no additional action is required. If the server is not running, see the section in 4.2.

- __2. Reload the sample OSLC-CM client



- __3. Enter Service URL: <http://localhost:8080/oslc-sample> as before, selecting Go!.

4. Next select **New Change Request** from the sample page and observe the new dialog



Conclusion

You have completed lab 8. You now have an understanding of the OSLC-CM sample provider, how to launch the sample provider server and modify some of its source and seeing the results

Appendix A. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have

been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental. All references to fictitious companies or individuals are used for illustration purposes only.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Appendix B. Trademarks and copyrights

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM

Rational

Jazz

Adobe, Acrobat, Portable Document Format (PDF), and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. See Java Guidelines

Apache Tomcat and Apache HttpComponents are trademarks of The Apache Software Foundation.

Mozilla and Firefox are registered trademarks of the Mozilla Foundation.

Other company, product and service names may be trademarks or service marks of others.



© Copyright IBM Corporation 2010. All rights reserved.

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

IBM, the IBM logo, and other IBM products and services are trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.

