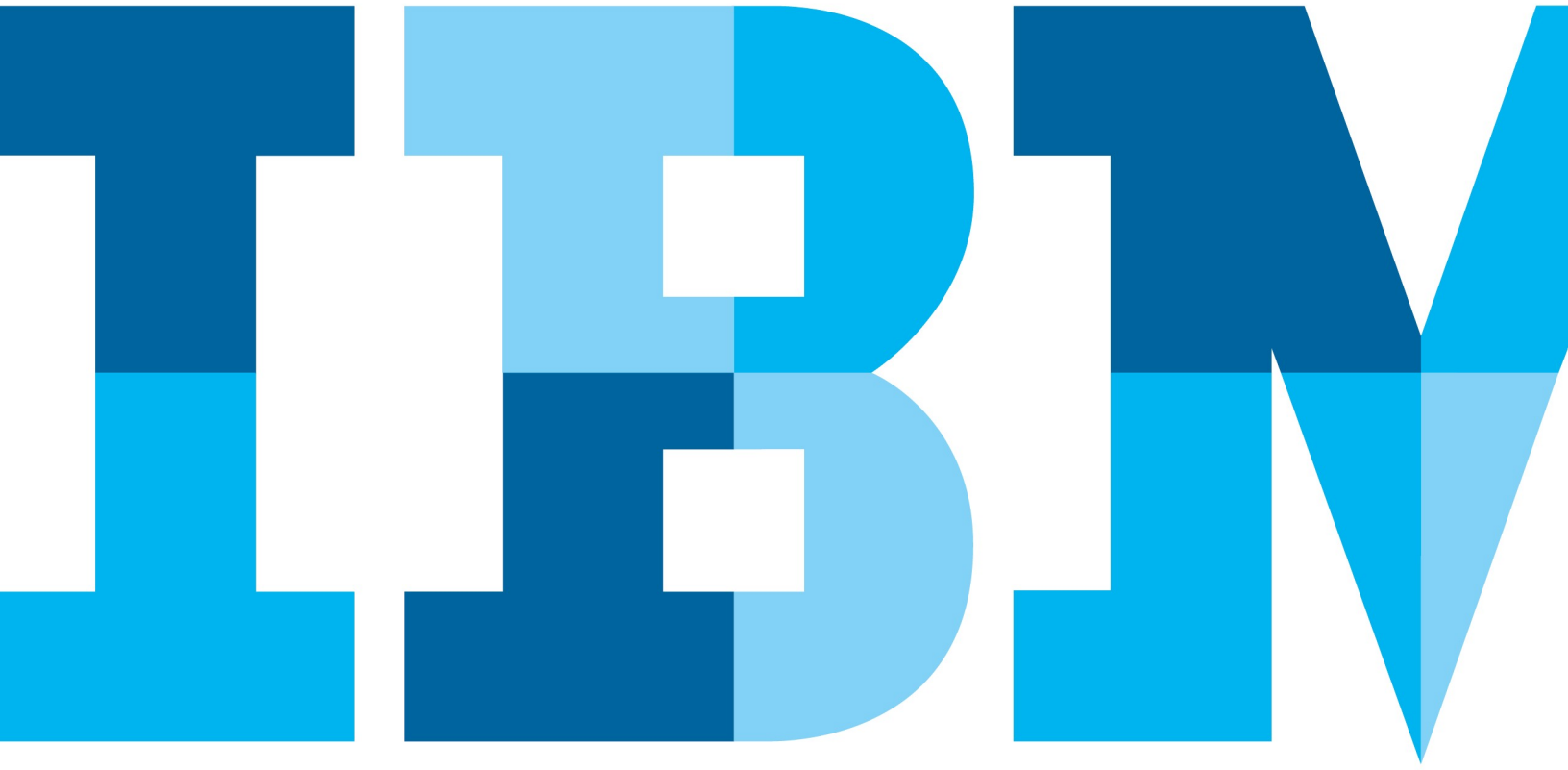
IBM Software
Rational

IBM Rational CLM 2012 OSLC Workshop

Lab Exercises



IBM

An IBM Proof of Technology

Catalog Number

Contents

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 4.0 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 YYYY-MM-DD-`oslc-workshop.zip`. It is an exported set of Eclipse projects containing the code we will use in the Labs. You will import it at the end of this lab.

1.1 Download and Install the Required Files from jazz.net

For supporting sections, use this format. Replace the heading above with your own title, and then add an introduction. If you need to add steps, copy and paste the step list below.

__1. Download Rational Team Concert 4.0 (https://jazz.net/downloads/rational-team-concert/releases/4.0) and follow the Getting Started (https://jazz.net/downloads/rational-team-concert/releases/4.0?p=gettingStarted) instructions to install it and setup properly:

- Jazz Team Server, CCM Application, RM Application
- Client for Eclipse IDE

__2. Add the Eclipse Web Tools Platform (WTP) to the RTC client.

__a. Start the RTC Eclipse client (<TeamConcert_Root_Folder> \eclipse.exe).

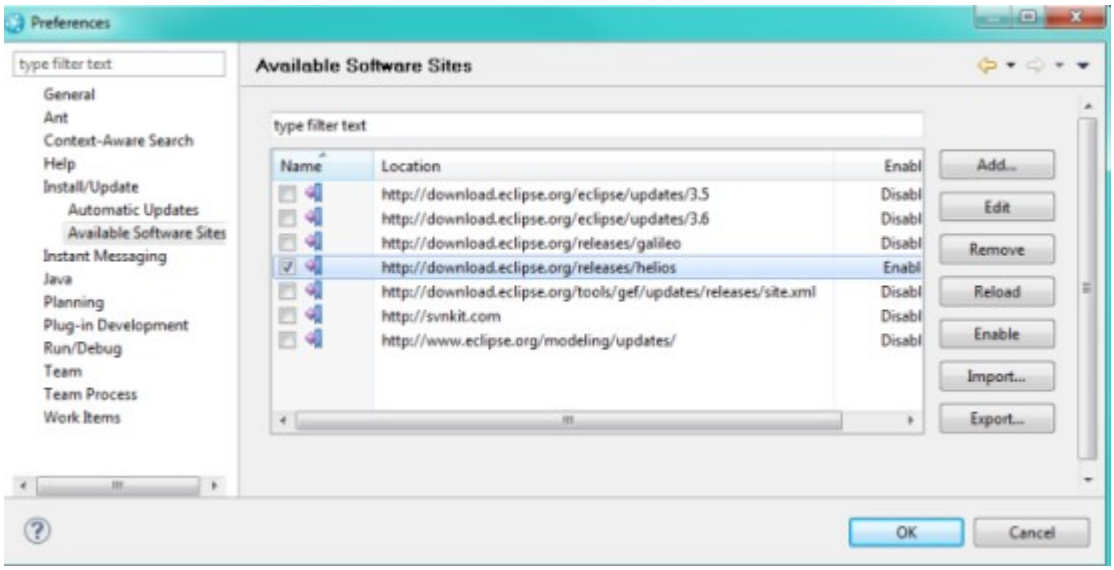
__b. When prompted, select an Eclipse workspace. These instructions will use C:\RTC40Dev\DevWS.

__c. Minimize the **Welcome** via this () button near the top of the window.

__d. From the menu bar, select **Window > Preferences**

__e. In the left column select the item **Install/Update > Available Software Sites**

In the **Available Software Sites** view, select http://download.eclipse.org/releases/helios , then click **OK**.



__f. Close the **Preferences** dialog.

__g. From the menu bar, select **Help > Install New Software…**

__h. In the **Available Software** dialog, select in the **Work with:** field, the item "**http://download.eclipse.org/releases/helios**"

Be patient, the first time, this request might take several minutes ….

__i. Group listed items by Category and select the features: **Web, XML, and Java EE Development**

__j. Press **Next**.

__k. Eclipse calculates the dependencies and shows you the features it will install. Click **Next** in the **Install** wizard.

__l. 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**.

__m. When prompted to restart Eclipse, click **Yes**.

1.2 Setup a Tomcat Test Server

__1. This server will be used to run the OSLC provider sample. Download a Tomcat 6.0.xx Tomcat server from http://tomcat.apache.org/download-60.cgi or one of the mirrors.

__2. Unzip the downloaded **apache-tomcat-6.0.xx.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-6.0.xx** 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-6.0.xx\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-6.0.xx\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.

There also exist the HttpRequester add-on which the RM team uses and released as an add-on. It can be downloaded from here: https://addons.mozilla.org/en-US/firefox/addon/httprequester/

__2. If you do not already have Mozilla Firefox installed, go to http://www.mozilla.com/firefox/, download version 3.6 or above and follow the installation instructions.

__3. Start Mozilla Firefox, go to this page (https://addons.mozilla.org/en-US/firefox/addon/restclient/), click the **Add to Firefox** button and follow the instructions, including the restart of Mozilla Firefox.

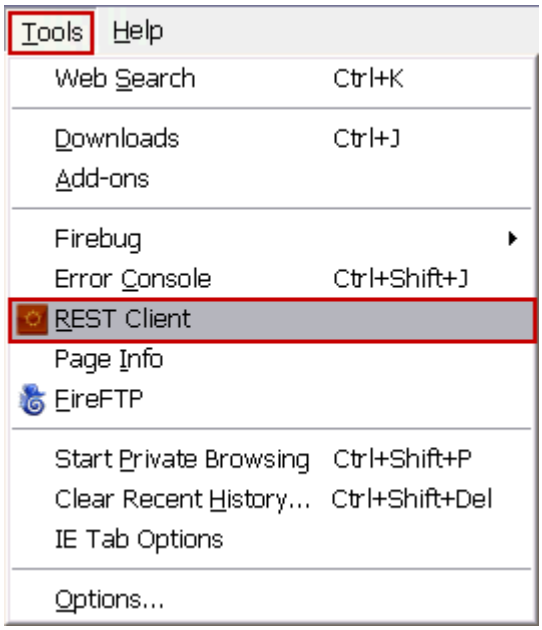
1.4 Test the REST Client Setup

__1. Start your Jazz Team Server and recall the public URI (root of all the URLs) you have set during the setup process. We will use in later steps the following public URI as the example: https://jazz.server.com:9443/jts

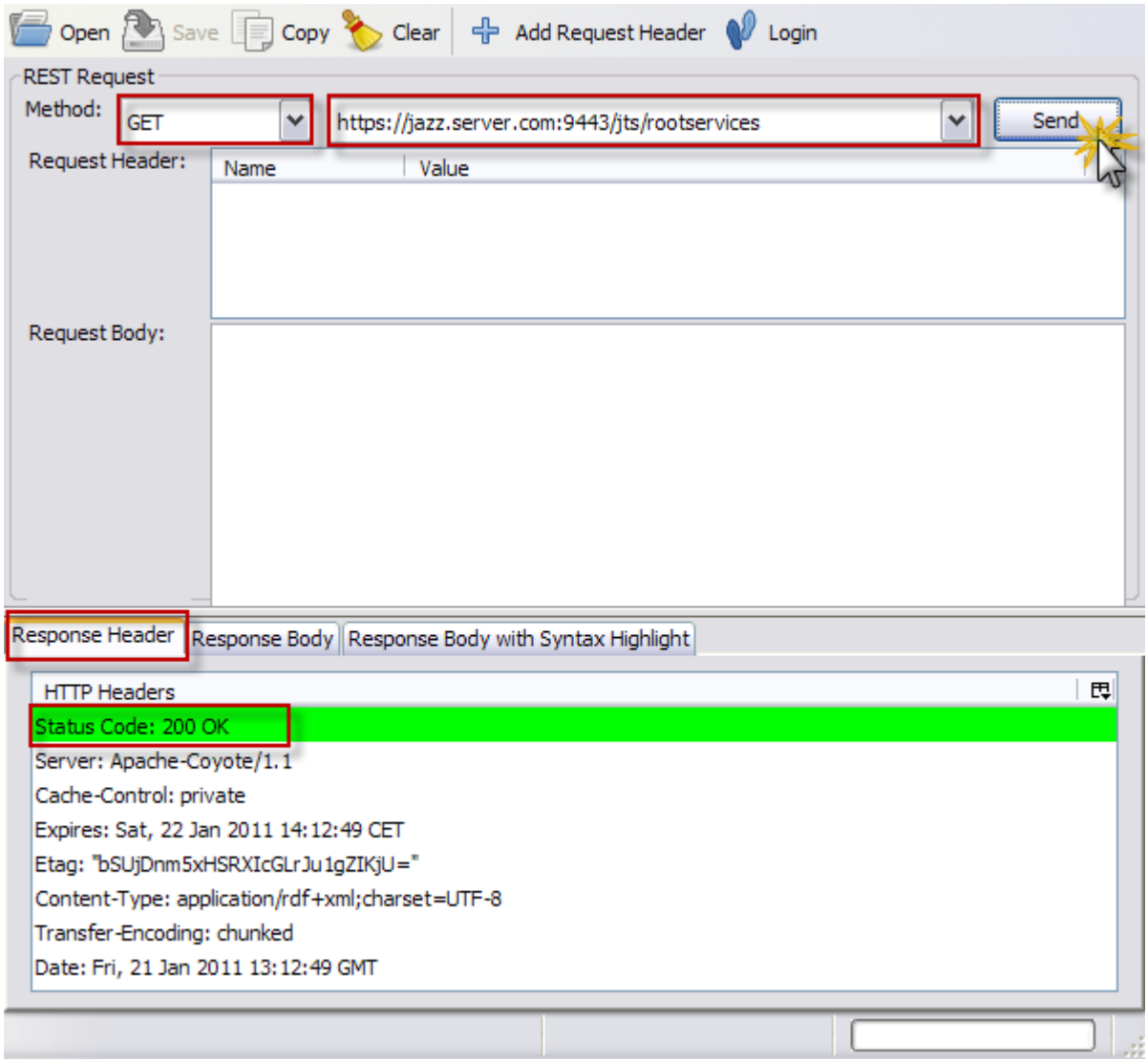
__a. In the Windows Explorer, navigate to the **<JazzTeamServer_Root_Folder>\server** folder.

__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://jazz.server.com:9443/jts/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 (Preview)** 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.

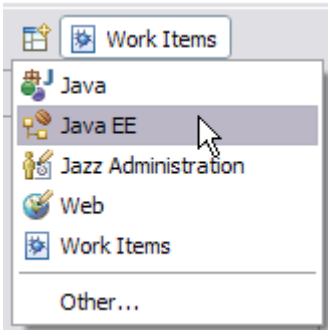
1.5 Test the WTP and Tomcat Setup

__1. Create a Tomcat Server definition.

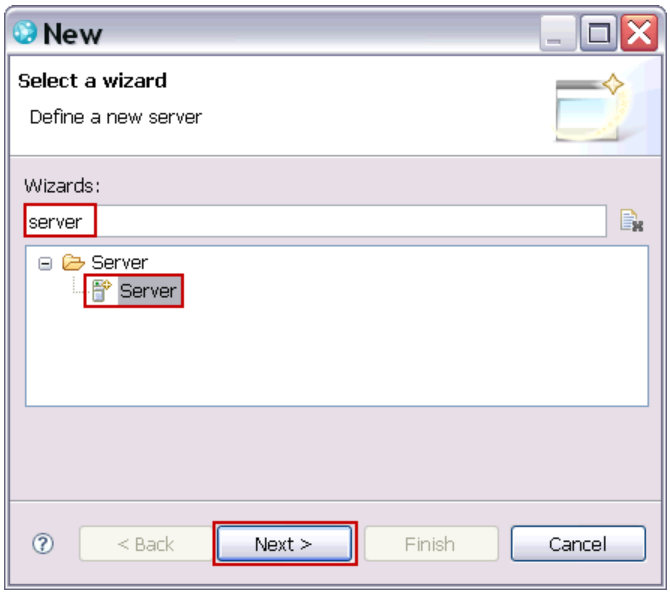
__a. For the OSLC Producer workshop, you will use the Web Tool Platform and deploy your code against the Apache Tomcat application server. You will test the configuration now.

__b. If the RTC client with WTP is not still running, start It now.

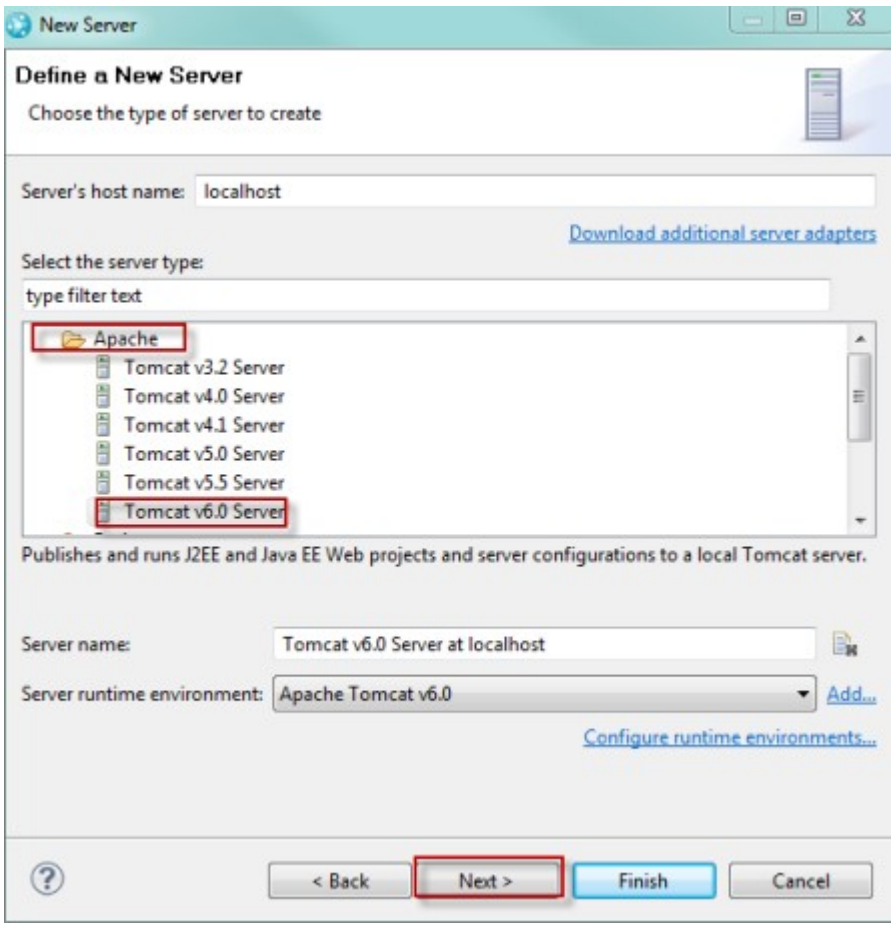
__c. Open the **Java EE** perspective.



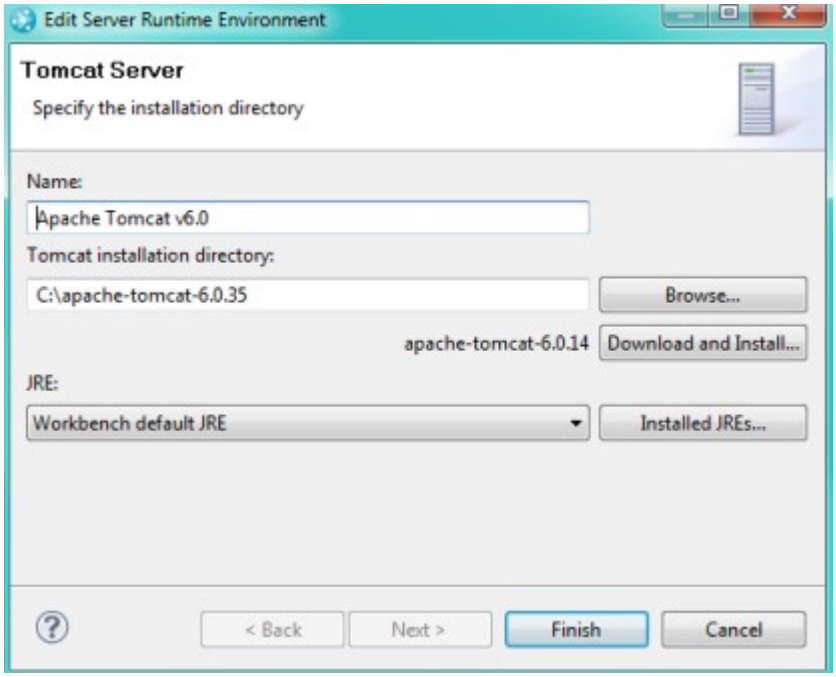
__d. From the menu bar select **File > New > Other…** then in the **New** wizard, type `server` in the filter, select **Server** from the list and then click **Next**.



__e. On the second page of the wizard, expand the **Apache** tree, select **Tomcat v6.0 Server** and then click **Next**.

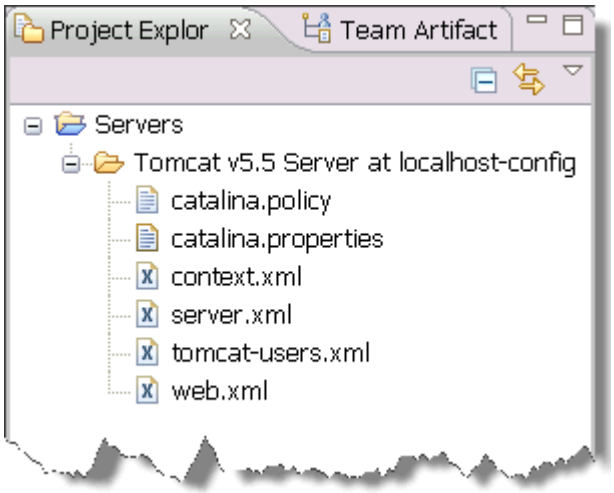


__f. On the final page of the wizard, specify where you have unzipped Tomcat (use the **Browse…** button) and press **Finish**.



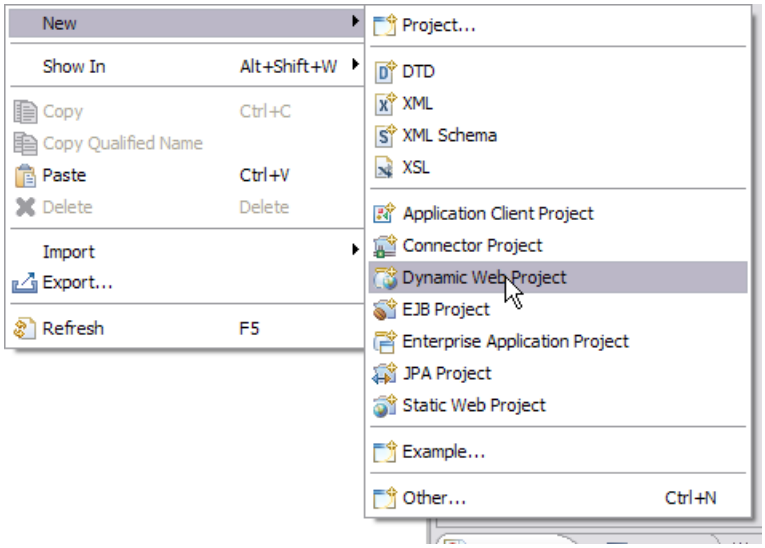
If you already have installed Tomcat in the Eclipse workspace then this wizard page will not show up.

__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.

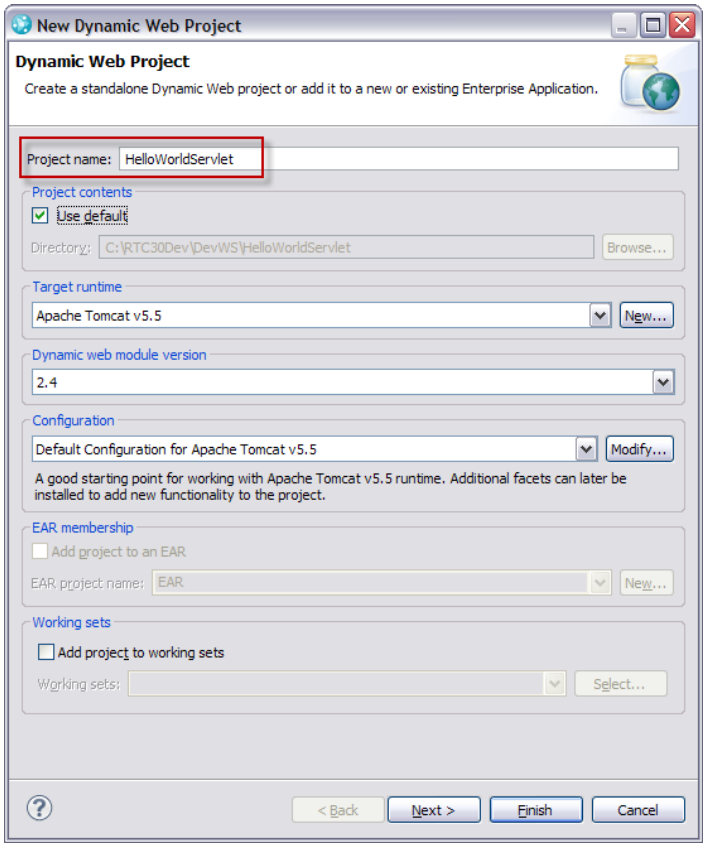


__2. Create a new web project to run on the Tomcat server.

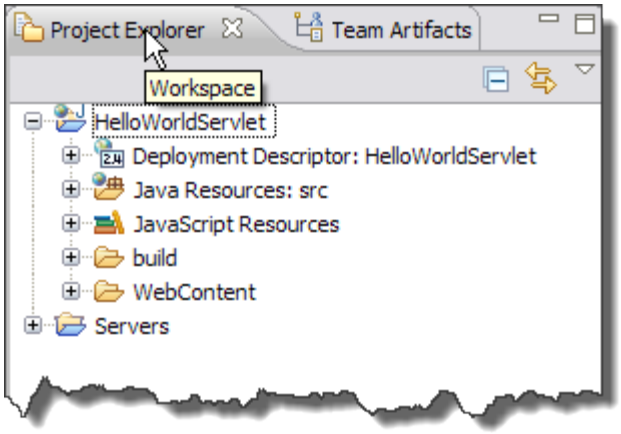
__a. Right click in empty space in the **Project Explorer** view, then select **New > Dynamic Web Project**.



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



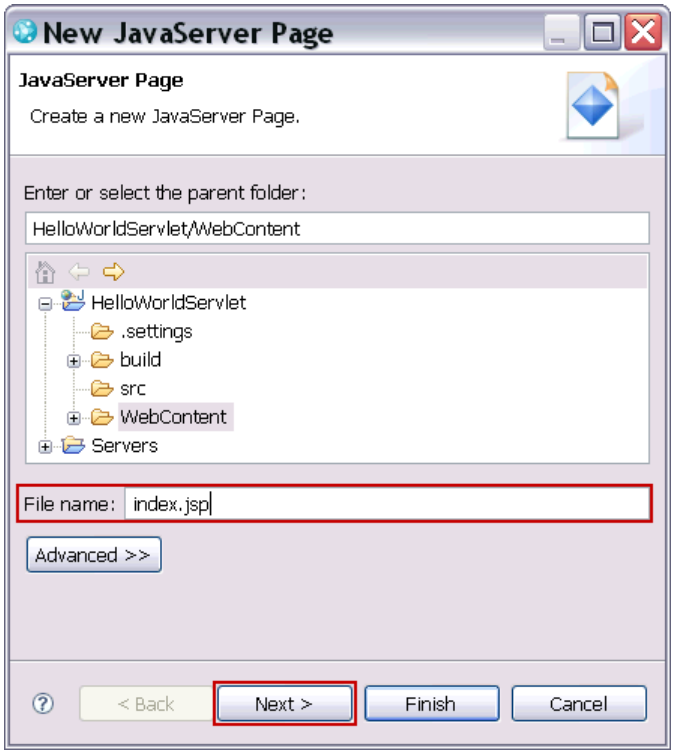
__c. A web project is created in the Project Explorer.



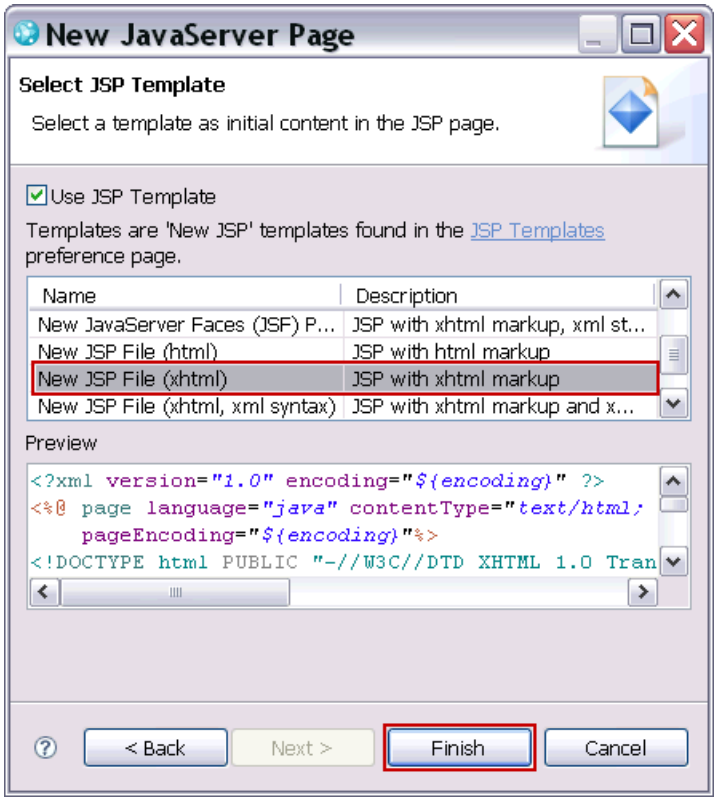
__3. Create the default page for the web project.

__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, select **New JSP File (xhtml)** and then click **Finish**.

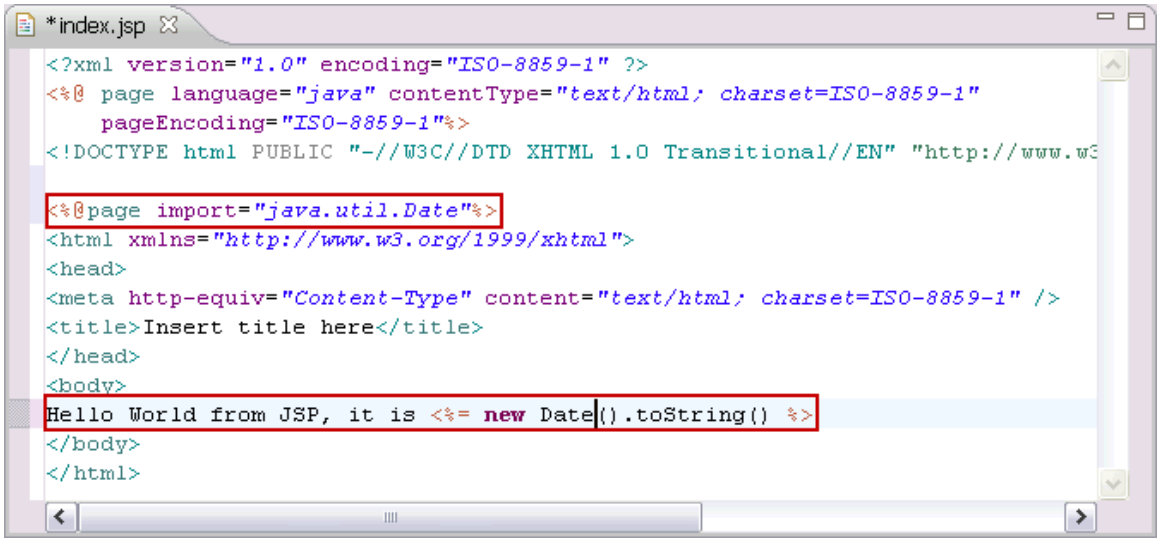


__d. A file named **index.jsp** will be created under the **WebContent** folder and an editor will open for this new file.

__e. Within the <body> element, type the following HTML.

```
Hello World from JSP, it is <%= new Date().toString() %>
```

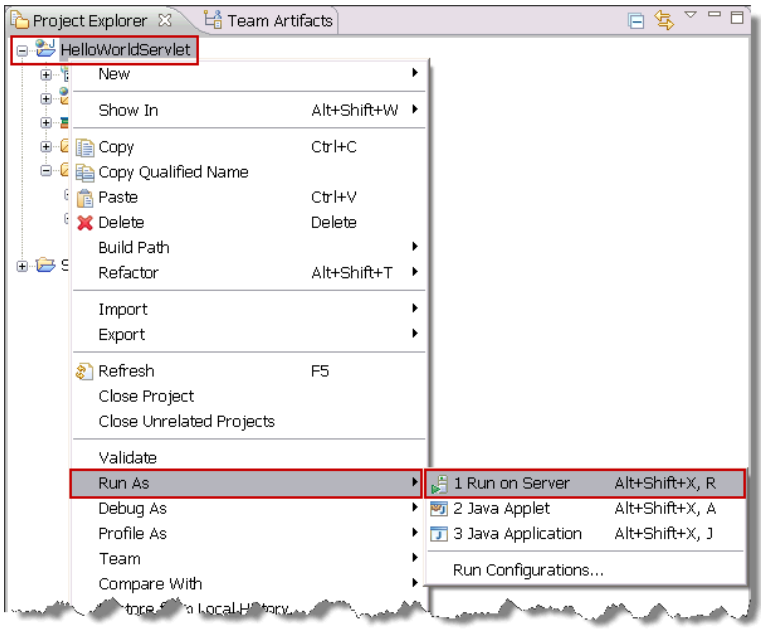
__f. If after typing Date, you use code assist (Ctrl+Space), a list of possible classes will be presented. Select java.util.date from the list and the import of that class will be automatically added to your file as shown here.



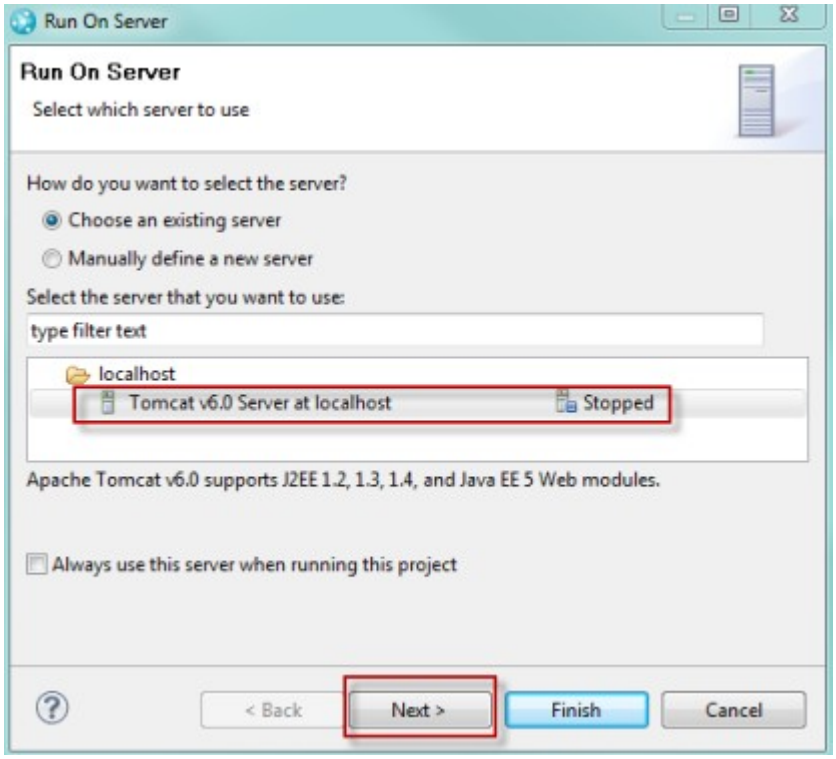
__g. Save your changes (Ctrl+S).

__4. Run the web project on the Tomcat server.

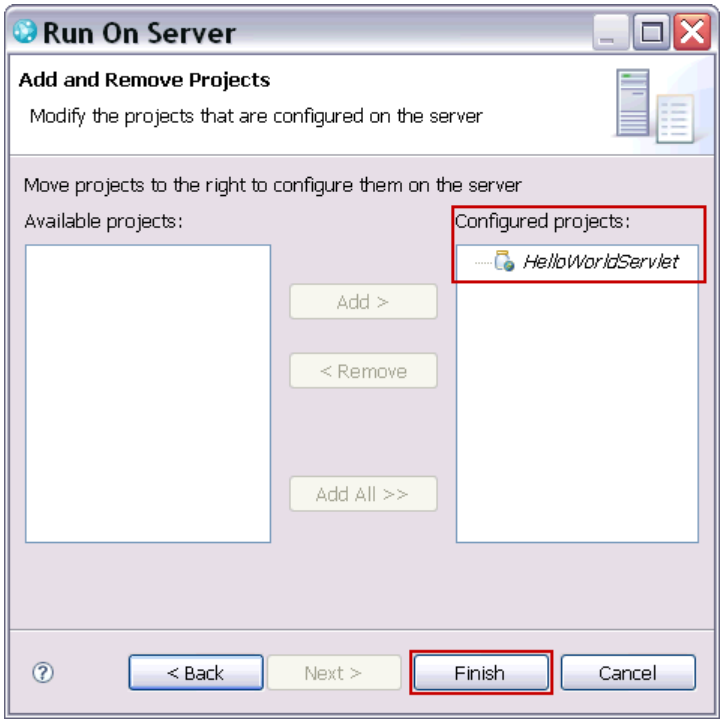
__a. In the Project Explorer view, right click the HelloWorldServlet project then select **Run As > Run on Server** from the menu.



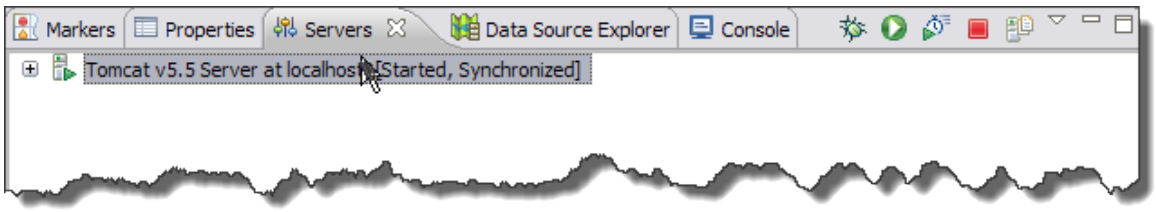
__b. In the **Run On Server** wizard select your Tomcat server and press **Next**.



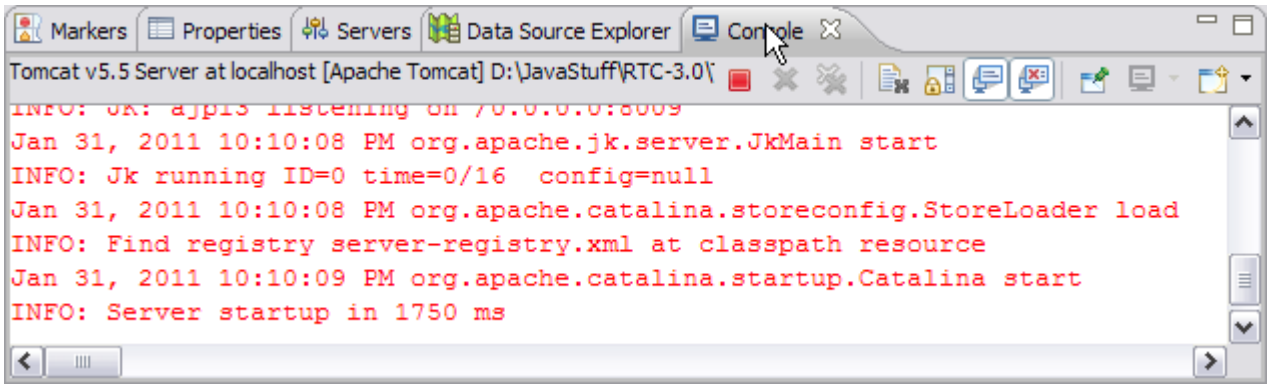
__c. On the second page verify that your servlet has been correctly added to the **Configured projects** list and then click **Finish**.



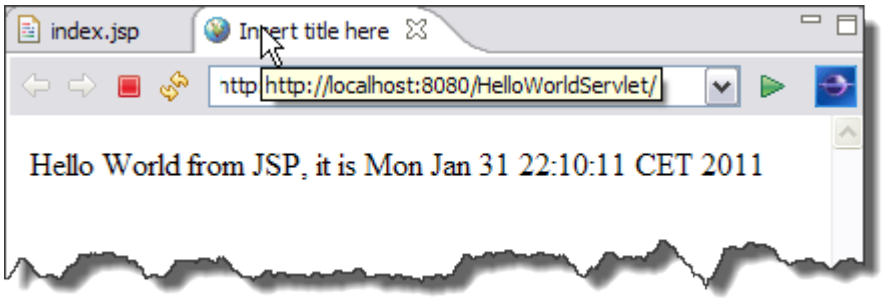
__d. Soon, the **Servers** view will show that the server has started…



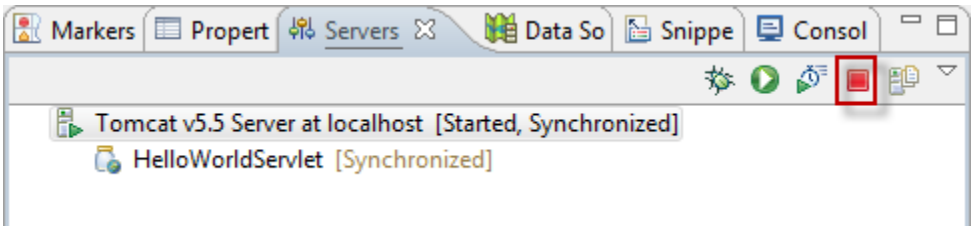
__e. The **Console** view will display some info about the running server...



__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.



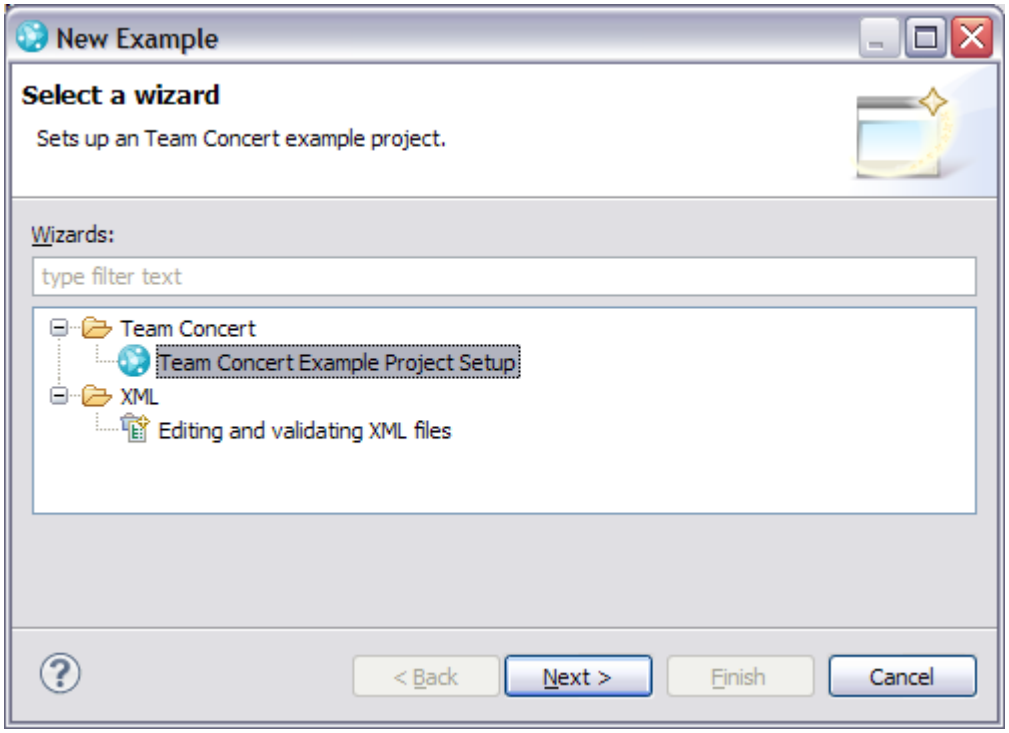
1.6 Create a default project (or reuse an existing one!)

If you already have an RTC project area to use for the labs in your repository then skip this section and go to section 1.7. If you don't already have a project area, you can create one using the following steps:

__1. From the Eclipse RTC Client, open the **Work Item** perspective

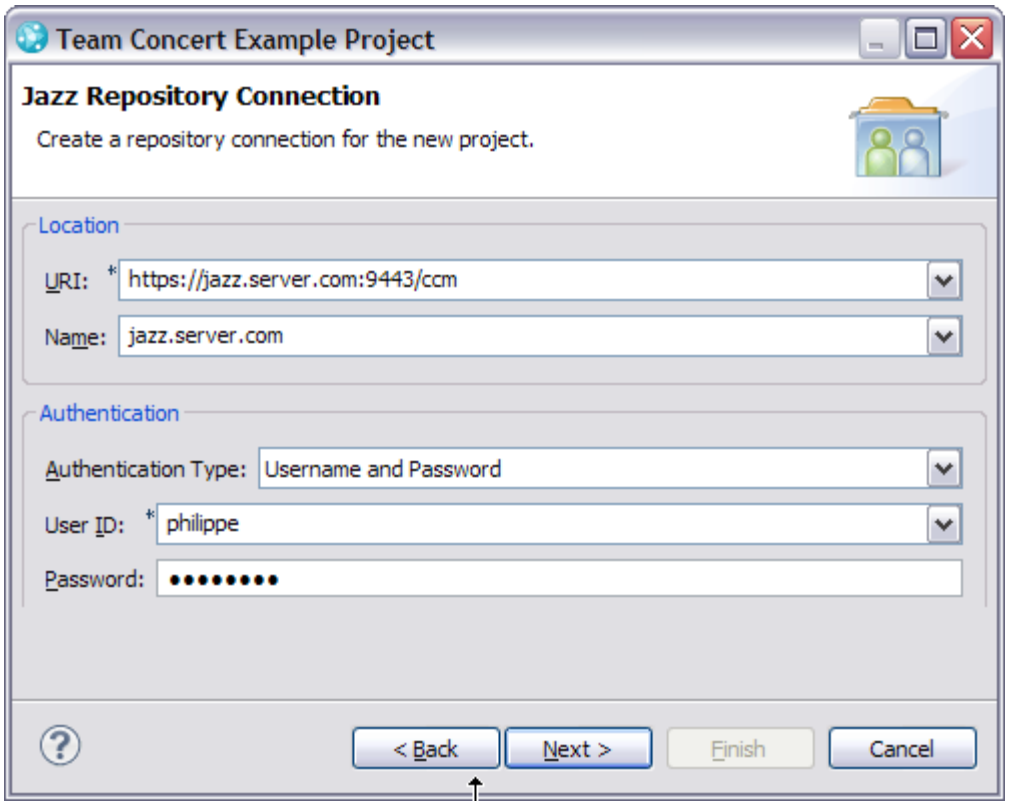
__2. From the menu bar select **File > New > Example…**

__3. Select in the list **Team Concert Example Project Setup** and press **Next**



__4. Press **Next** again

__5. On the next page, specify the Public URI of your CCM repository and login as a user having the RTC - developer Client Access License and press **Next**



__6. Press **Next** one last time.

__7. Finally press **Finish** to create this project example in your repository.

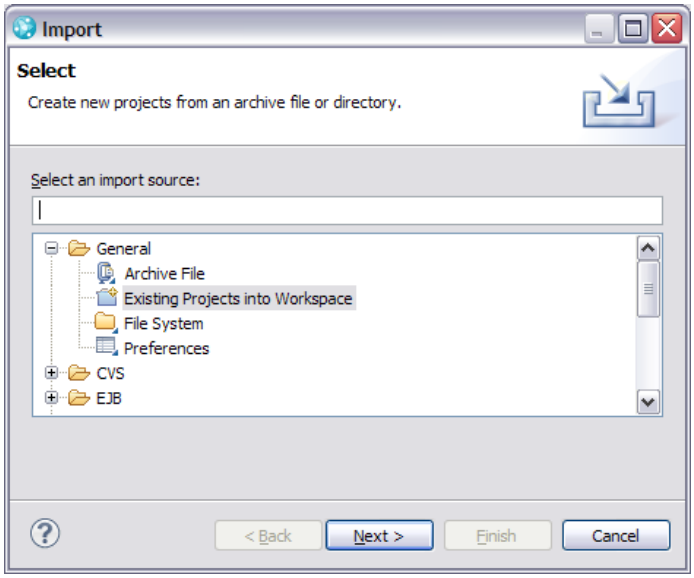
1.7 Loading examples

The zip file `YYYY-MM-DD-oslc-workshop.zip` contains all the Eclipse projects that you need to the OSLC workshop. We will import them in your current Eclipse workspace.

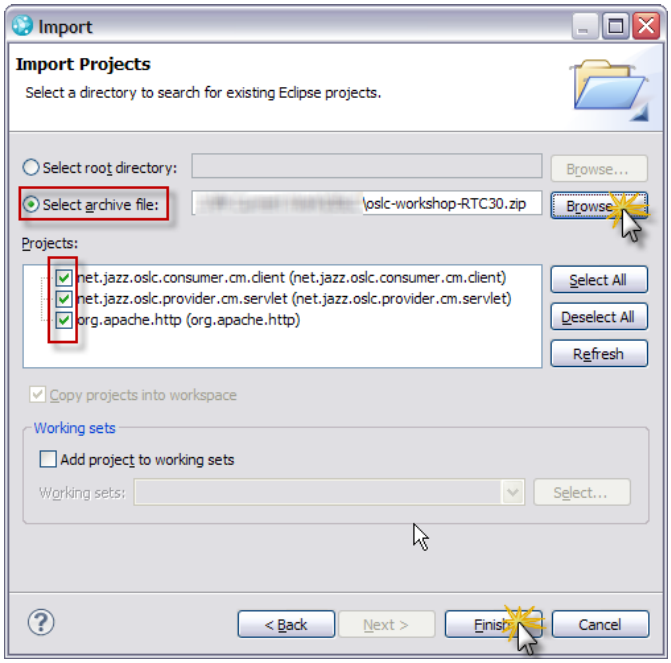
__1. Open the **Java EE** perspective

__2. From the menu bar access to **File > Import…**

__3. Select in the list **General > Existing Projects into Workspace** and press **Next**.

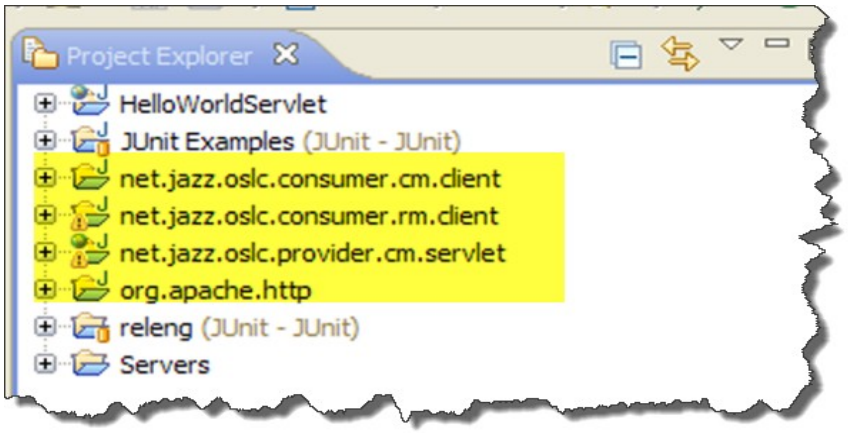


__4. Select the radio-button **Select archive file** then press Browse to retrieve the `YYYY-MM-DD-oslc-workshop.zip` file.



__5. Verify that all projects in the zip file are selected and press **Finish**.

__6. At this point, you should retrieve these 3 projects into your **Project Explorer**:



- The `net.jazz.oslc.consumer.cm.client` project contains all the samples for Lab 3
- The `net.jazz.oslc.provider.cm.servlet` project contains the code for Lab 4
- The net.jazz.oslc.consumer.rm.servlet project contains all the samples for Lab 6
- The `org.apache.http` project contains the Apache libraries used by the `net.jazz.oslc.consumer.cm.client` project.

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.

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 (<JazzTeamServer_Root_Folder>`\server\server.startup.bat`).

OSLC-CM Release 2

Since the first publication of this workshop, a new version of the OSLC-CM specs has been released (http://open-services.net/bin/view/Main/CmSpecificationV2).

For compatibility issues between 1.0 and 2.0 specifications, please read http://open-services.net/bin/view/Main/CmSpecificationV2?sortcol=table;table=up;up=#Version_Compatibility_with_1_0_S

RTC 3.0.1 supports both OSLC-CM specs.
By default, RTC 3.0.1 returns its results using the OSLC-CM 1.0 specification. So, if you wrote any code based on the OSLC-CM 1.0 rlease, it should be still working.

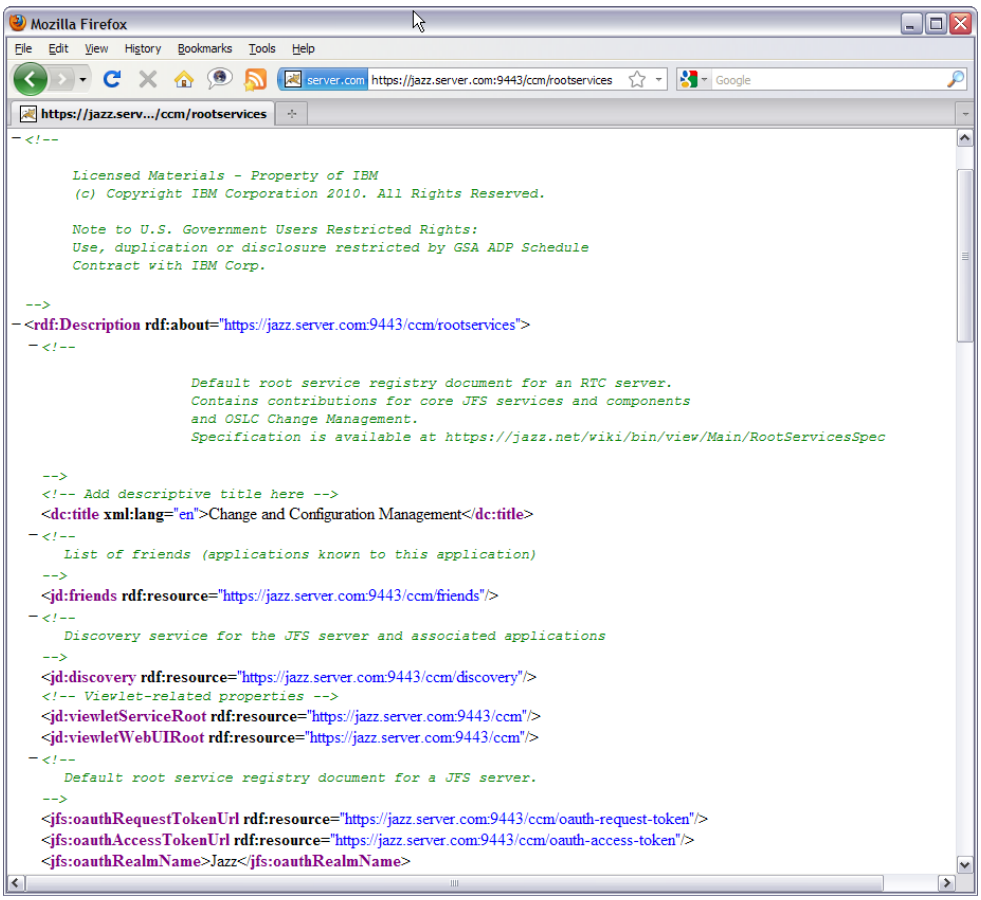
This workshop is based on the newest OSLC-CM specification to help you use the latest features of these APIs.

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://jazz.server.com:9443/ccm/rootservices
This URL will return the **Root Services** document which is an XML informational resource that lists a set of REST services and capabilities.



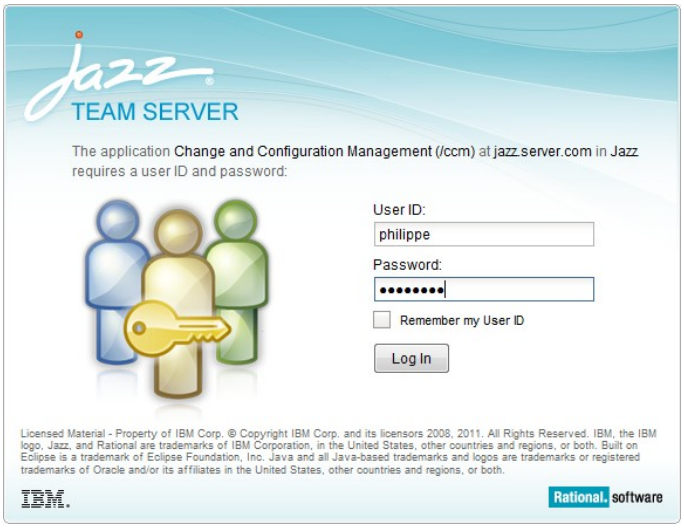
Root Services
The Root Services URL locates a REST API for discovering the Jazz Team Server's various services and specific capabilities, such as a way to discover a web UI presentation for particular kinds of resources.
This Service is an open-ended mechanism that can be used to keep track of and share 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 “**whoami**” service.
Look for a tag named **jfs:currentUser**. The URL associated to the unique attribute **rdf:resource** is the URL to access to the “**whoami**” service.

```
<!--
    Service to redirect to the resource that represents the authenticated user
-->
<jfs:currentUser rdf:resource="https://jazz.server.com:9443/ccm/whoami"/>
<!-- JFS storage service -->
<jfs:storage rdf:resource="https://jazz.server.com:9443/ccm/storage"/>
<!-- JFS SPARQL query service -->
<jfs:query rdf:resource="https://jazz.server.com:9443/ccm/query"/>
```

__4. Copy this URL (https://jazz.server.com:9443/ccm/whoami) and paste it into the navigation field of your web browser and press **Enter**.

__5. If it is the first time that you try to reach this page, you will have to login first.
Log in using the credentials you defined to access the CCM application.

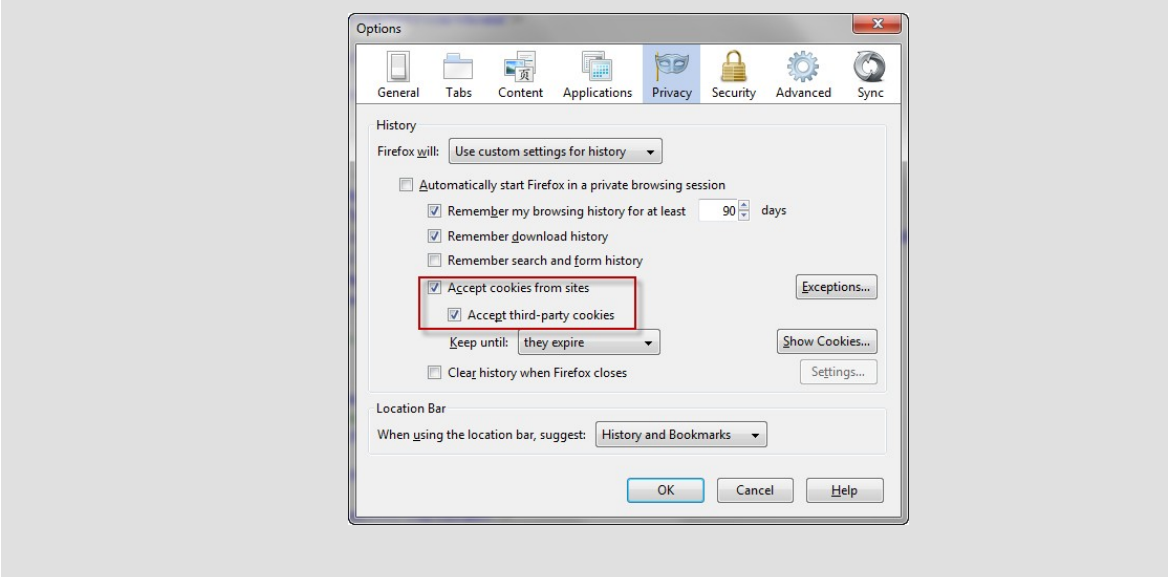


Authentication repeatedly fails

If you cannot pass the login page it might come from the fact that your Web Browser doesn't accept third-party cookies.

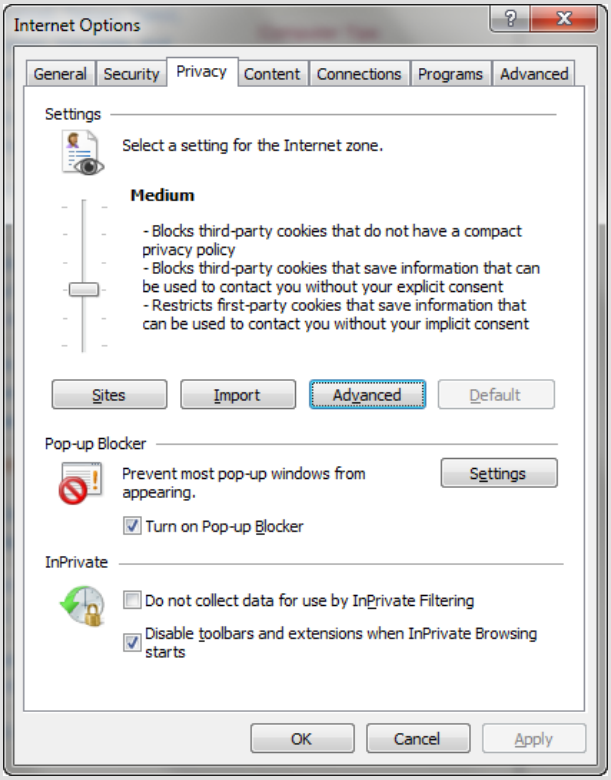
To fix this issue in **Firefox**, follow these steps:

1. Open the **Tools > Options..**.
2. Select the **Privacy** page
3. Verify that the following check-boxes are checked:

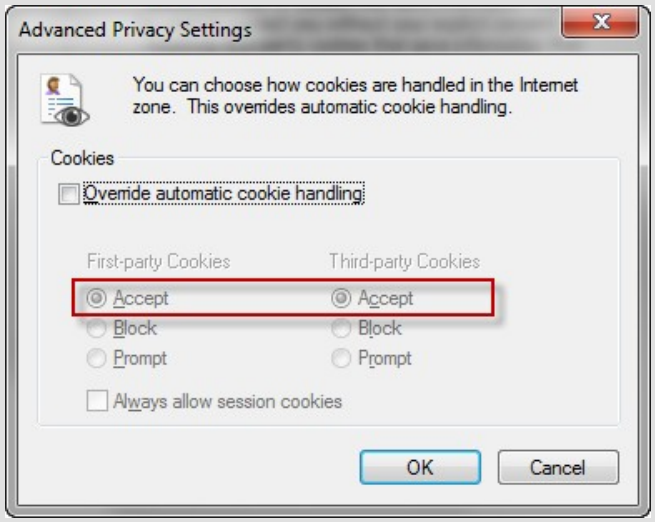

Options
General
Tabs
Content
Applications
Privacy
Security
Advanced
Sync
History
Firefox will:
Use custom settings for history
Automatically start Firefox in a private browsing session
Remember my browsing history for at least
90
days
Remember download history
Remember search and form history
Accept cookies from sites
Exceptions...
Accept third-party cookies
Keep until:
they expire
Show Cookies...
Clear history when Firefox closes
Settings...
Location Bar
When using the location bar, suggest:
History and Bookmarks
OK
Cancel
Help

To fix this issue in **Internet Explorer**, follow these steps:

1. Open the **Tools > Internet Options**
2. Select the **Privacy** page

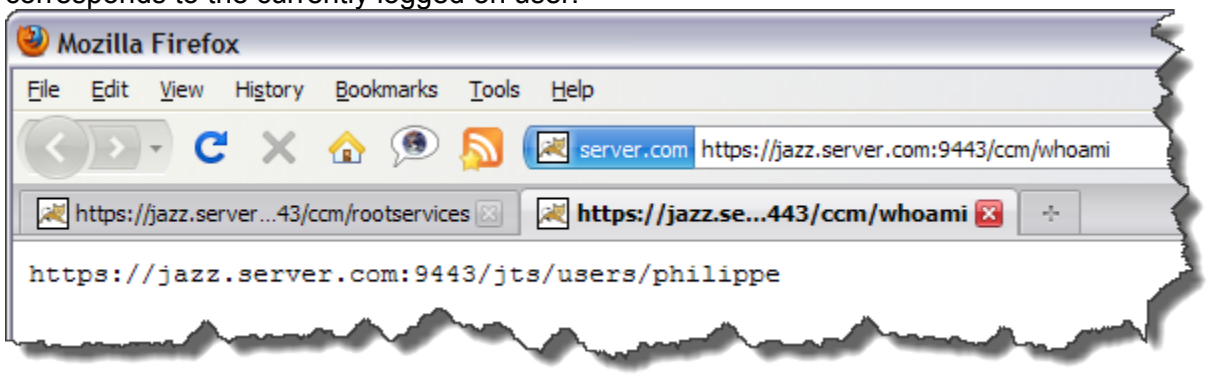


3. Press the **Advanced** button

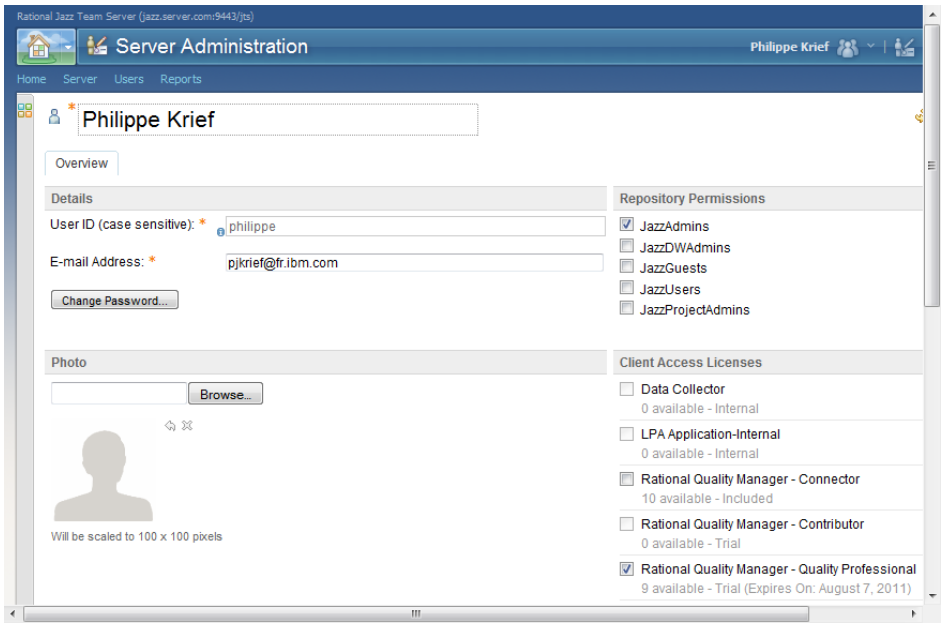


4. Verify the the **Third-party Cookies** are selected

__6. 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

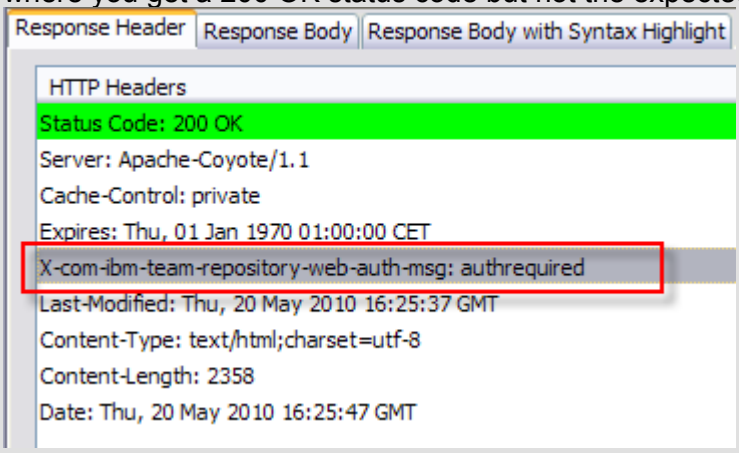
To be able to properly handle the response of a REST service, we need to use a REST client application which will let us specify all of the parameters required for an HTTP method.

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).

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:



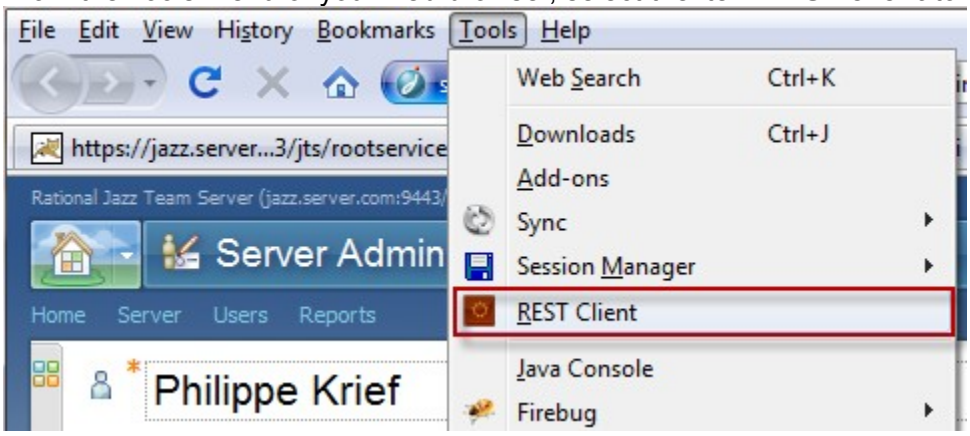
In this case, please check out if the HTTP headers you got doesn't have a key/value field set to:

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

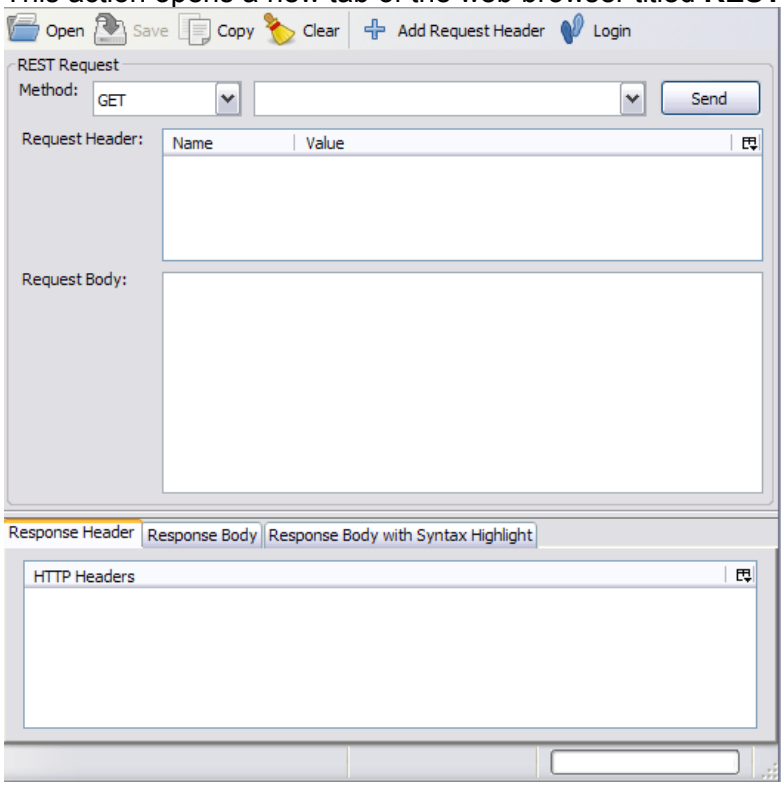
Then it means that you try to access to a protected resource and you need to login first. In such case, follow these steps to fix this issue:

1. Open a new tab on your current web browser
2. Login to the server entering the following URL in your navigation field: https://jazz.server.com:9443/ccm/web
   The login dialog Web UI will show up:
3. Login with using the credentials you have defined during the Jazz Team Server setup.
4. Once you are logged, switch back to the REST client and retry to run the described sample. It should work as expected.

__1. From the Tools menu of your web browser, select the item **REST client** to open the client UI.



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

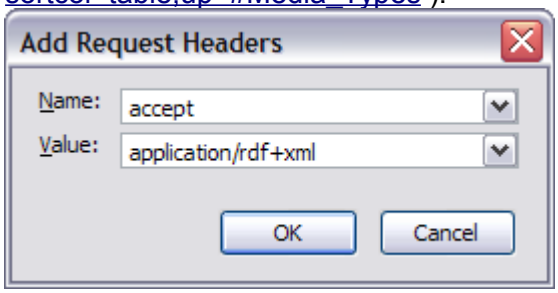


__2. Now we should be able to specify the correct parameters to access the XML representation of the user resource.

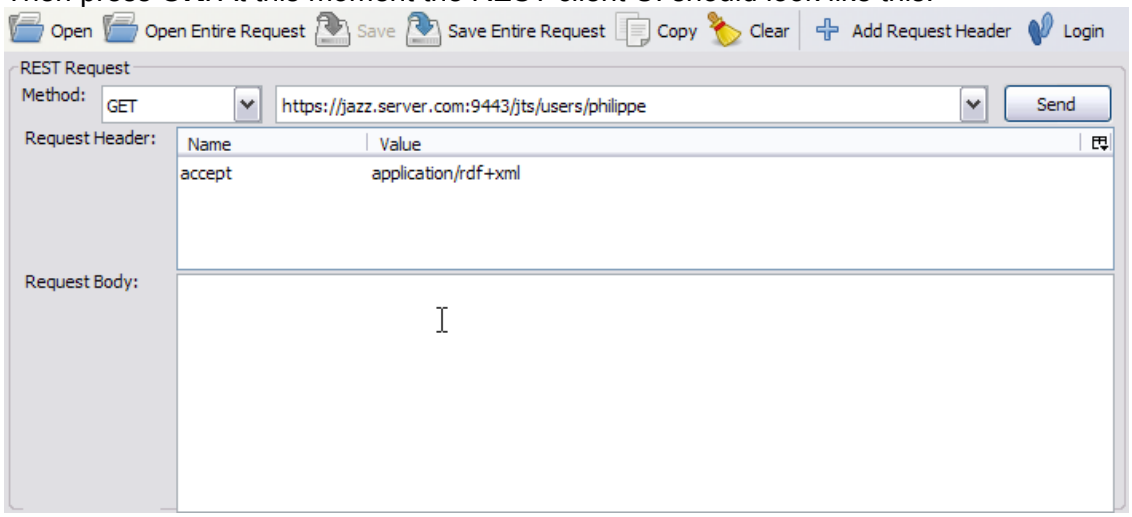
__a. Choose the `GET` method from the **Method** combo-box

__b. In the URL field, copy/paste the URL of the current user that the “whoami” service has revealed to you (e.g. https://jazz.server.com:9443/jts/users/philippe).

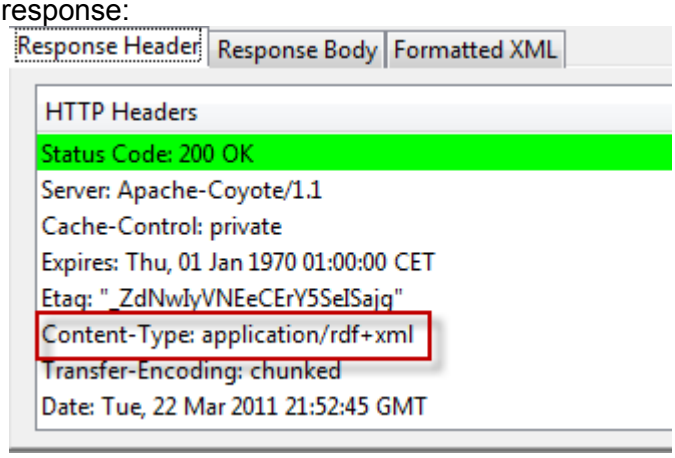
__c. Press the **Add Request Header** button and add the header key `accept` and the value `application/rdf+xml`. This header specifies the Media Type of the response you are expecting: RDF/XML (http://open-services.net/bin/view/Main/CmSpecificationV2?sortcol=table;up=#Media_Types).



__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:



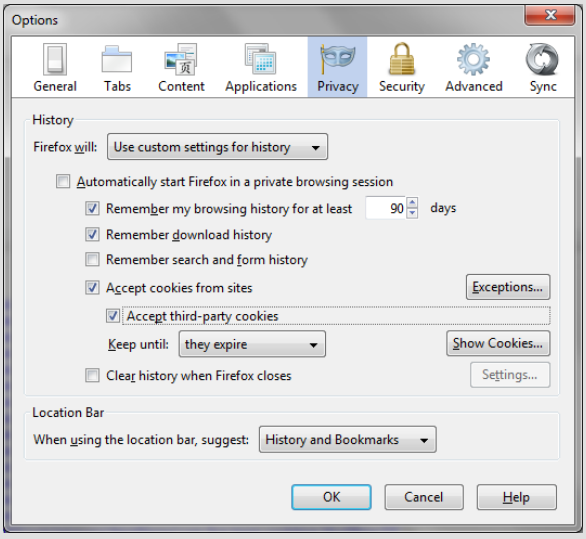
If the **Status Code** has the value **"200 OK"** it means that the HTTP request was handled properly. If not, please check out the URL and the header you have provided.
You can also check that the type of the response (Content-Type) corresponds to your request **application/rdf+xml**.

Cache issue

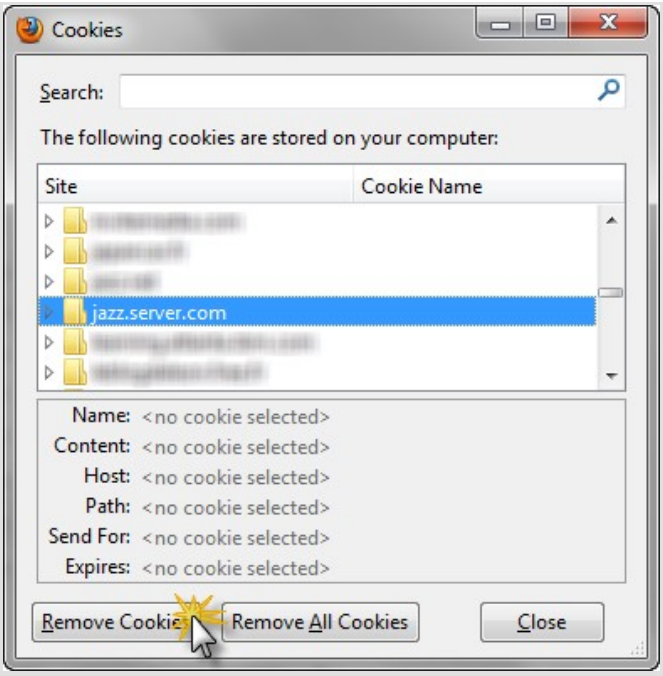
It happened during the writing of this workshop that the **REST client** plugin will not return the expected media type. For example instead of **application/xml** we will get an **application/rdf+xml** result. It seems to be a Web Browser cache issue.

The work around will consist in:

1. Opening the **Tools > Options...** of your Firefox web browser
2. Accessing to the **Privacy** page



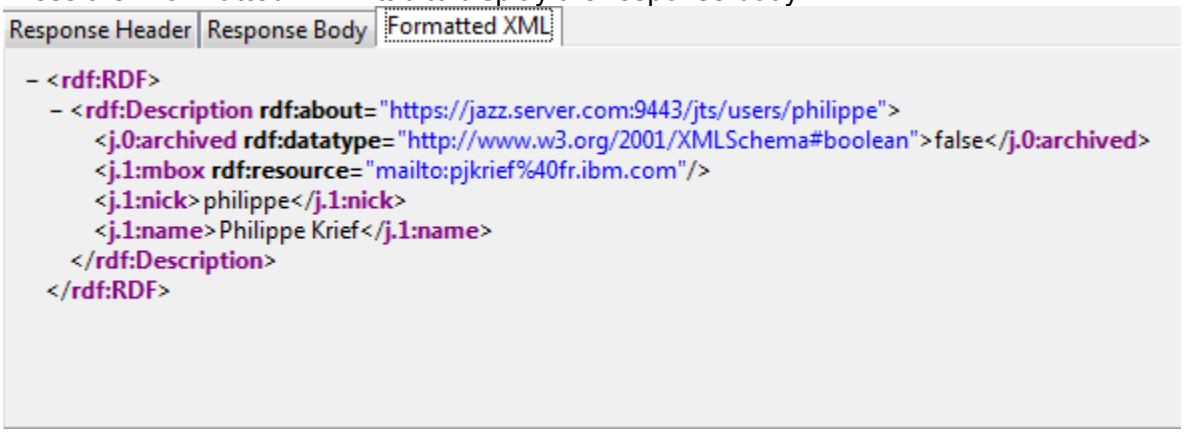
3. Pressing the **Show Cookies...** button
4. Retrieving the server you are talking too in your OSLC requests



5. And, finally, the pressing the button **Remove Cookies**.

After these steps, you can retry to run your OSLC request, it should work.

__4. Press the "**Formatted XML**" 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-CM 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-CM** 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/bin/view/Main/CmSpecificationV2).

__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://jazz.server.com:9443/ccm/oslc/workitems/catalog"/>
```

__2. Copy this URL (https://jazz.server.com:9443/ccm/oslc/workitems/catalog) and paste it into the URL field of REST client.

__3. If you already have a 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/xml`. This header specifies you are expecting an XML response (and not an RDF/XML).
You can also double-click on the header to edit it and adapt it to the example.

Resource Format

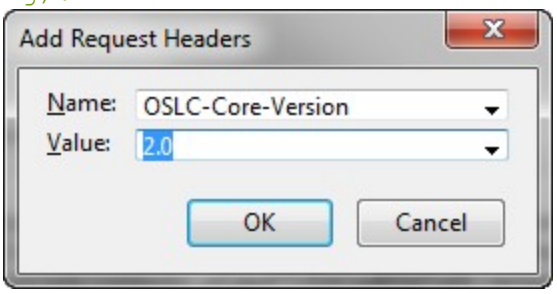
The Specification says (http://open-services.net/bin/view/Main/CmSpecificationV2?sortcol=table;table=up#Resource_Formats):

When CM Consumers request:

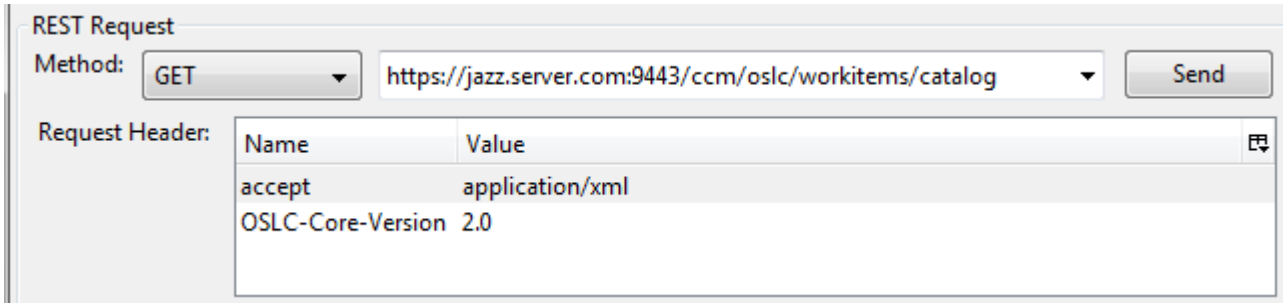
- **application/rdf+xml** CM Providers MUST respond with RDF/XML representation without restrictions.
- **application/json** CM Providers MUST respond with JSON representation as defined in the OSLC Core Representations Guidance.
- **application/xml** CM Provider MUST respond with OSLC-defined abbreviated XML representation as defined in the OSLC Core Representations Guidance
- **application/atom+xml** CM Provider MUST respond with Atom Syndication Format XML representation as defined in the OSLC Core Representations Guidance
- The Atom Syndication Format XML representation SHOULD use RDF/XML representation without restrictions for the atom:content entries representing the resource representations.

For the readability of this workshop we require as often as it is possible the **application/xml** media type. Feel free to use another resource format if it is more convenient for you.

__5. **Press the Add Request Header button a second time and declare a new header OSLC-Core-Version: 2.0.** The header indicates to the OSLC producer that you are expecting a response based on the latest release of the OSLC-CM specifications (http://open-services.net/bin/view/Main/OslcCoreSpecification#Specification_Versioning).

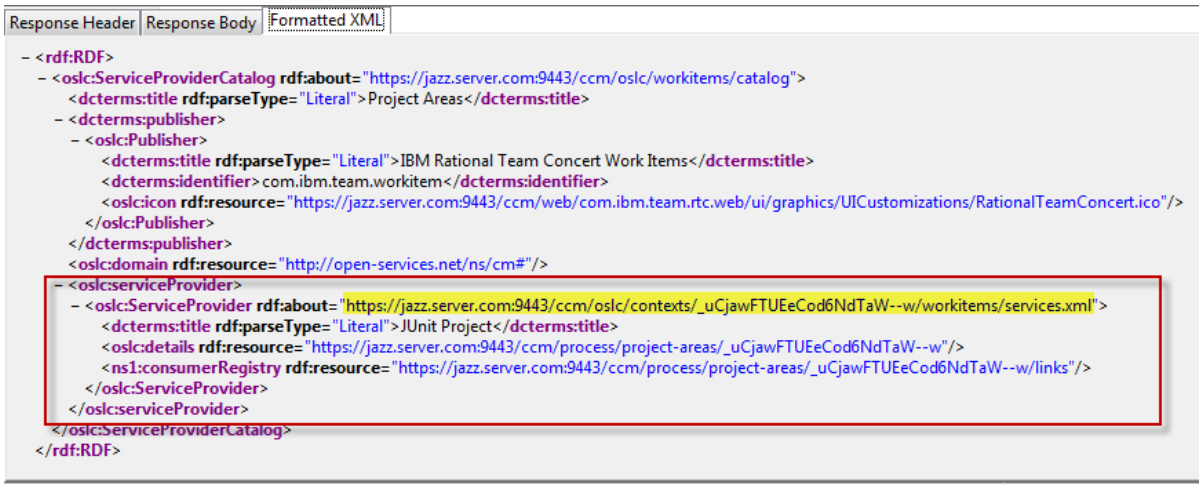


__6. Then press **OK**. At this moment, the Firefox REST client UI should look like this:



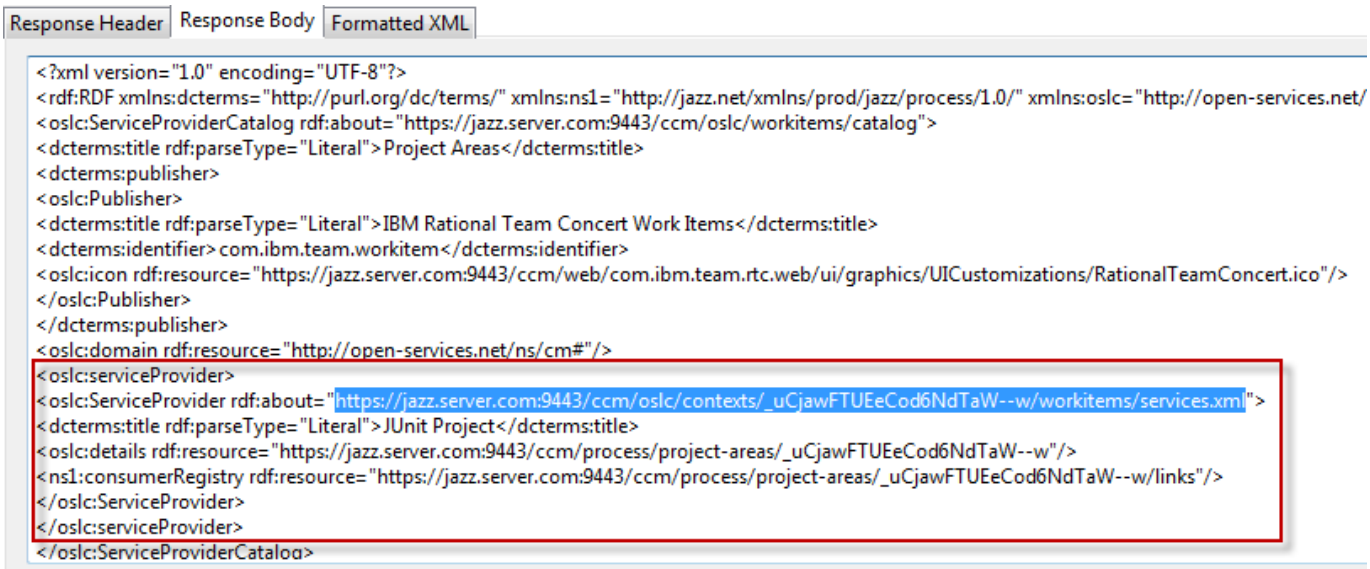
__7. Press the **Send** button. The REST service response header will be displayed at on bottom part of the UI. If the Status Code doesn't show the value "200 OK", please check out the URL and the headers you have provided.

__8. Press the “**Formatted XML**” tab to display the response body.



The resulting document contains a list of **oslc: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.

__9. Press the “**Response Body**” tab to display the response body source text.



__10. From this view, you should be able to retrieve the Service Provider element for the **JUnit Project** Project Area (or your chosen Project Area) and copy the URL associated to the attribute **rdf:about** defined in the element **oslc:ServiceProvider** .

__11. Paste the Service Provider URL in the URL field of the REST client and keep the headers as they are:

- Accept: application/xml
- OSLC-Core-version: 2.0

REST Request

Method: GET https://jazz.server.com:9443/ccm/oslc/contexts/_uCjawFTUEeCod6NdTaW--w/workitems/services.xml

Request Header:

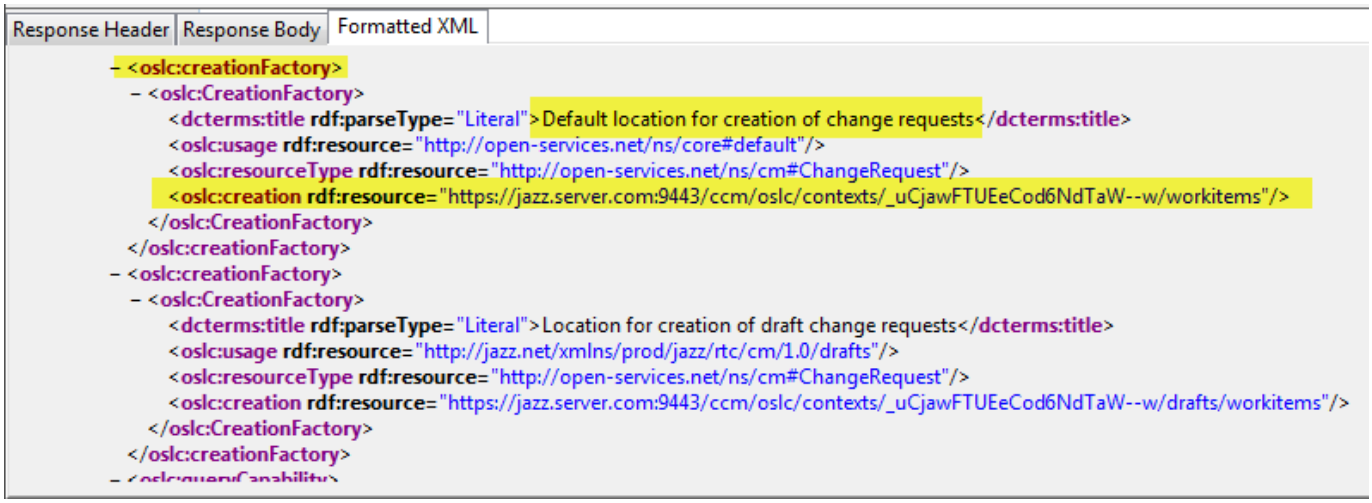
Name	Value
accept	application/xml
OSLC-Core-Version	2.0

```
<ns1:consumerRegistry rdf:resource="https://jazz.server.com:9443/ccm/process/project-areas/_uCjawFTUEeCod6NdTaW--w/links" />
- <dcterms:publisher>
  - <oslc:Publisher>
      <dcterms:title rdf:parseType="Literal">IBM Rational Team Concert Work Items</dcterms:title>
      <dcterms:identifier>com.ibm.team.workitem</dcterms:identifier>
      <oslc:icon rdf:resource="https://jazz.server.com:9443/ccm/web/com.ibm.team.rtc.web/ui/graphics/UICustomizations/RationalTeamConcert.ico"/>
    </oslc:Publisher>
  </dcterms:publisher>
  <oslc_cmx:whoami rdf:resource="https://jazz.server.com:9443/ccm/oslc/whoami"/>
- <oslc:service>
  - <oslc:Service>
      <oslc:domain rdf:resource="http://open-services.net/ns/cm-x#"/>
    - <calm:home>
      - <calm:Home>
          <dcterms:title rdf:parseType="Literal">Plans Home</dcterms:title>
          <calm:webHome rdf:resource="https://jazz.server.com:9443/ccm/web/projects/JUnit%20Project#action=com.ibm.team.apt.welcome"/>
        </calm:Home>
      </calm:home>
    - <oslc:selectionDialog>
      - <oslc:Dialog>
          <dcterms:title rdf:parseType="Literal">Plan</dcterms:title>
          <oslc:label>Plan</oslc:label>
          <oslc:usage rdf:resource="http://open-services.net/ns/core#default"/>
          <oslc:resourceType rdf:resource="http://open-services.net/ns/cm-x#Plan"/>
          <oslc:dialog rdf:resource="https://jazz.server.com:9443/ccm/_ajax-modules/com.ibm.team.apt.PlanSelectionDialog?projectAreaId=_uCjawFTUE
          <oslc:hintWidth>45em</oslc:hintWidth>
          <oslc:hintHeight>30em</oslc:hintHeight>
        </oslc:Dialog>
```

__12. Press the **Send** button. The response body will display Service Provider document (http://open-services.net/bin/view/Main/OslcCoreSpecification#Service_Provider_Resources) listing all the REST services available for this Service Provider (alias Project Area):

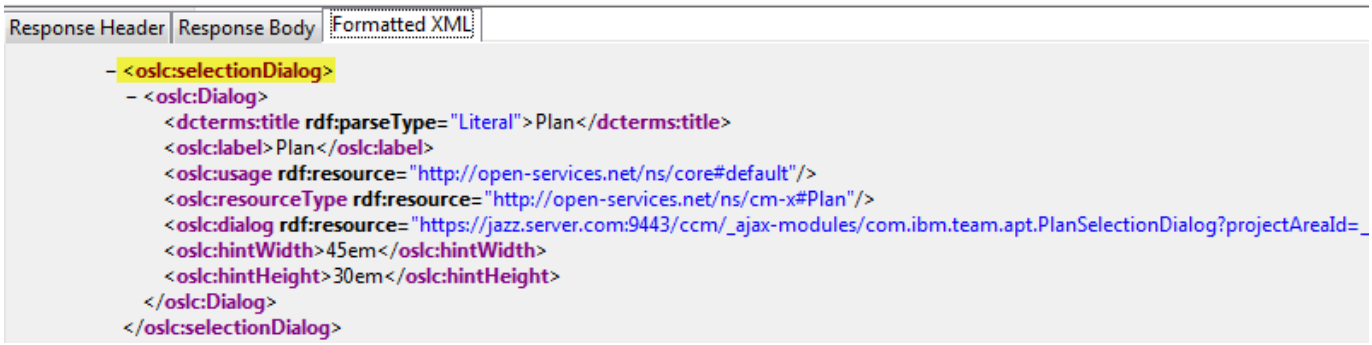
__13. You will find 4 kinds of services:

__a. **Creation Factory**: Enables clients to create new resources:

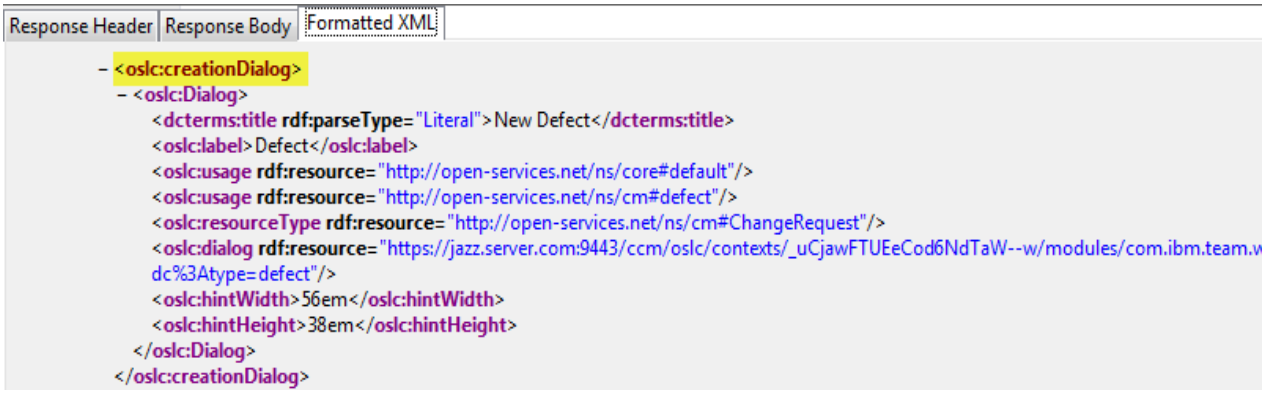


__b. **Query Capability:** Enables clients query across a collection of resources

__c. **Selection Dialog**: Enables clients to select a resource via UI



__d. **Creation Dialog**: Enables clients to create a resource via UI

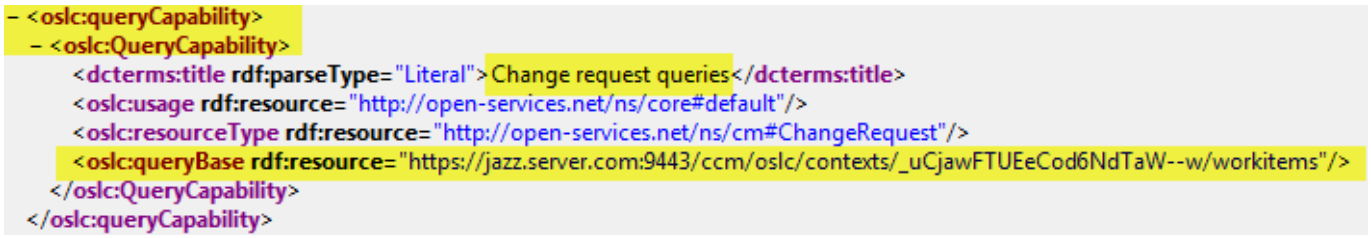


2.4 Search for Work Items

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

__6. Scroll back to the **oslc:QueryCapability** element in the Service Provider services document.

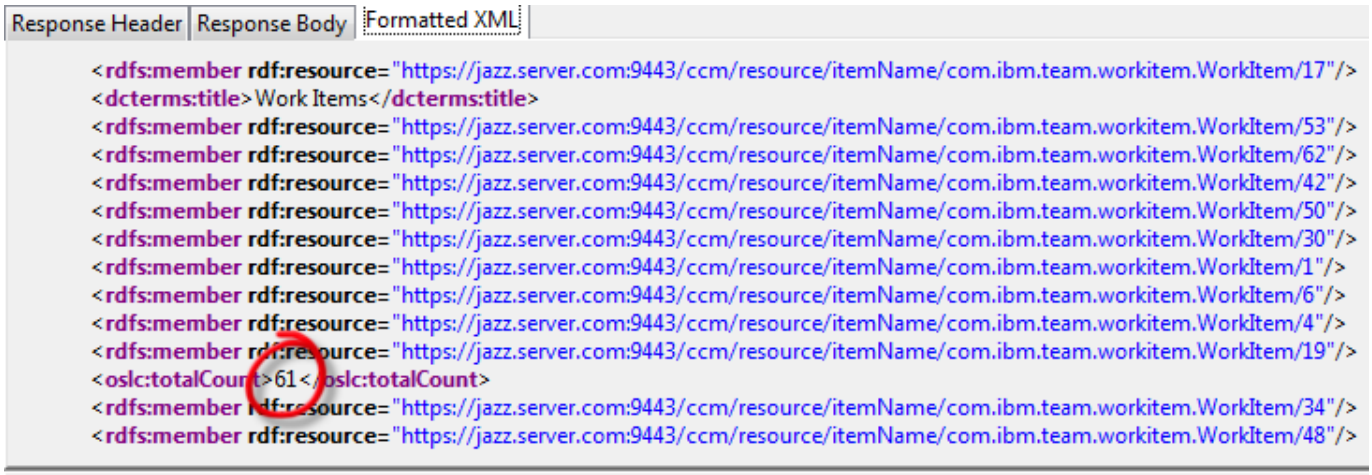
__7. From this element, look for the **oslc:queryBase** element and copy the URL associated to the attribute **rdf:resource** defined.



__8. Paste this URL into in the URL field of the REST client and keep the headers as they are.

__9. Press the **Send** button, the response body will display **ALL** the work items (**rdfs:member** elements) listed in the corresponding Project Area.

__10. Scroll down a little bit in the result, you will find an **oslc:totalCount** element indicating the number of **rdfs:member** elements.



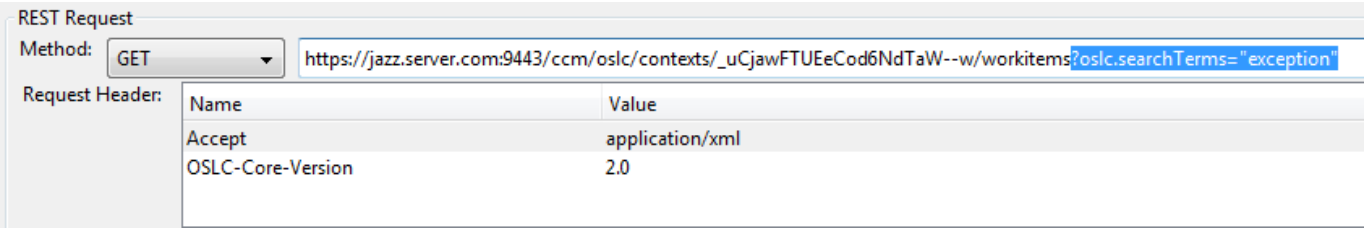
Actually, this REST service supports the following parameters (http://open-services.net/bin/view/Main/OSLCCoreSpecQuery): **oslc.searchTerms**, **oslc.where, oslc.select, oslc.properties and oslc.prefix.**

These parameters offer the possibility to filter the work items you are looking for and which data to fetch.

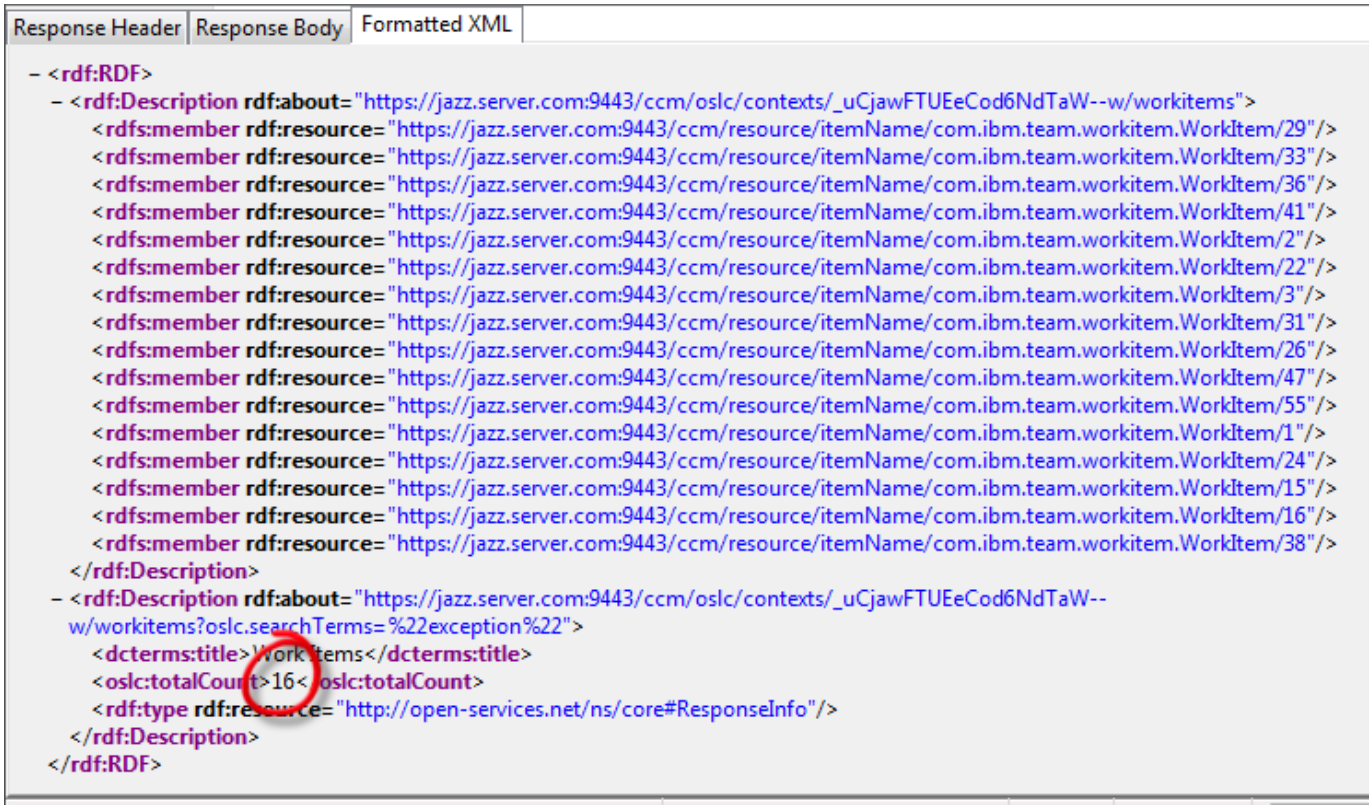
__11. For example, let try to retrieve all the work items which contain the word “exception” in it. To do so, complete the Query URL with the following parameter:

```
?oslc.searchTerms="exception"
```

So the URL should look like this:
https://jazz.server.com:9443/ccm/oslc/contexts/_uCjawFTUEeCod6NdTaW--w/workitems?oslc.searchTerms="exception" :



__12. Press the **Send** button. the response body will display a subset of these work items:



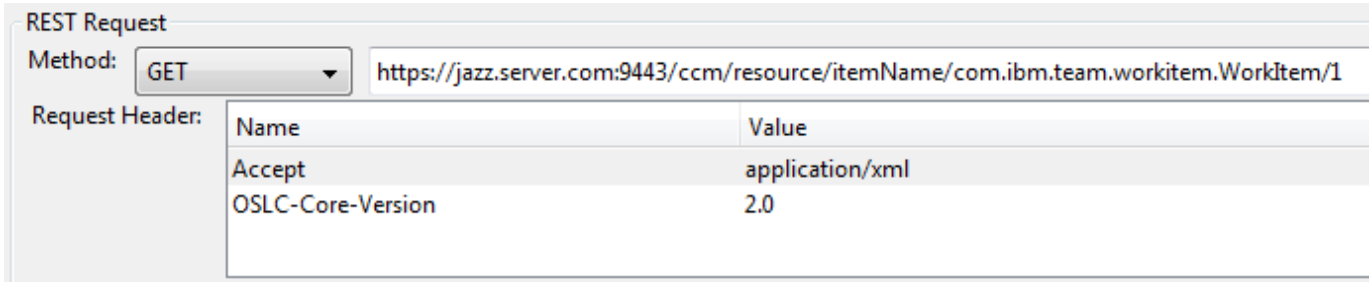
If we complete the previous request with the oslc.select parameter, we will be able to specify the subset of attributes/elements we want to fetch from the server. For example, let say you are only interested in the work item ID (dcterms:identifier) and the work item title (dcterms:title). In this case, complete the previous URL with the following expression:

```
&oslc.select=dcterms:identifier,dcterms:title
```

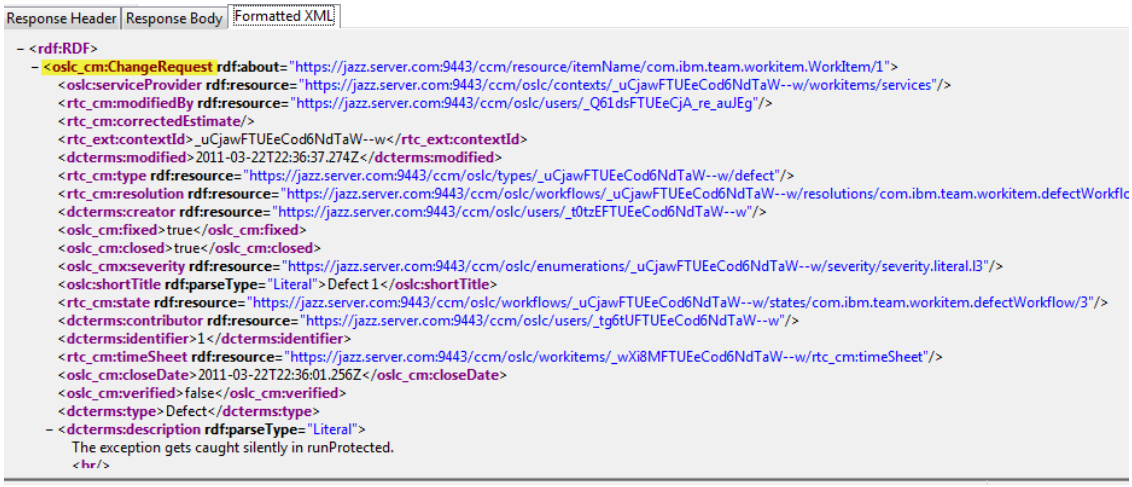
So the URL should look like this:
https://jazz.server.com:9443/ccm/oslc/contexts/_uCjawFTUEeCod6NdTaW--w/workitems?oslc.searchTerms="exception"&oslc.select=dcterms:identifier,dcterms:title

__13. Press the **Send** button. The REST client will only display the requested attributes:

__14. If you want to retrieve all the attributes of a work item, then grab the URL of a **rdf:member** previously listed and paste it in the URL field:



__15. Press **Send.** The **Formatted XML** tab should display all the attributes of the ChangeRequest (alias Work Item)



Conclusion

You have completed lab 2. 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.

This lab uses the OSLC-CM API. Later, in Lab 5, OSLC-RM API examples will be provided.

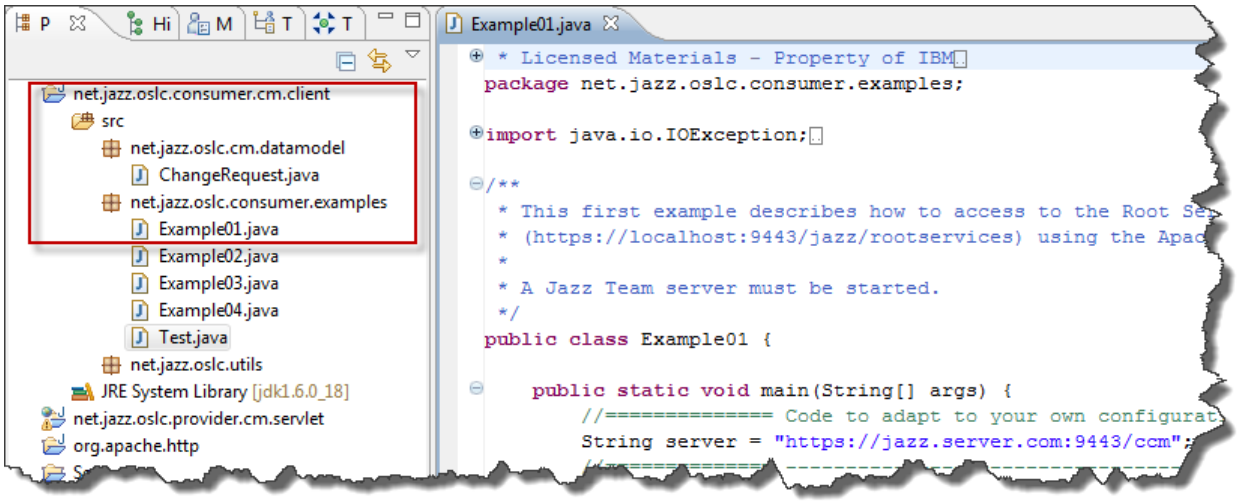
If your Jazz Team Server is not running, start it now (<JazzTeamServer_Root_Folder>`\server\server.startup.bat`).

3.1 Accessing 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. If not already running, start the RTC Eclipse client (<TeamConcert_Root_Folder> \eclipse.exe). When prompted to select an Eclipse workspace, use **C:\RTC30Dev\DevWS**.

__2. 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.



__3. Change the RTC Server URL, if necessary, by changing the value of the **String server** variable. For example, if your CCM server is accessed as **https://jazz.server.com:9443/ccm** then the initialization of the **server** variable would be

```
String server = "https://jazz.server.com:9443/ccm";
```

__4. Make sure you save the file (**Ctrl-S)**.

__5. The following snippet of code is extracted from the `main` method.

```
// Setup the HttClient
HttpClient httpclient = new DefaultHttpClient();
// Disabling SSL Certificate Validation
HttpUtils.setupLazySSLSupport(httpclient);
// Setup the HTTP GET method
HttpGet rootServiceDoc = new HttpGet(rootServices);
rootServiceDoc.addHeader("Accept", "application/rdf+xml");
rootServiceDoc.addHeader("OSLC-Core-Version", "2.0");

HttpResponse response;
try {
    // Execute the request
    response = httpclient.execute(rootServiceDoc);
    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();
}
```

__6. 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 HttClient
HttpClient httpclient = new DefaultHttpClient();
```

__7. 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.

__8. 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 CCM Root Services document. The request is completed with the header specifying the expected media type result and expected format base of the OSLC-CM Release 2.0 specifications.

```
// Setup the HTTP GET method
HttpGet rootServiceDoc = new HttpGet("https://jazz.server.com:9443/ccm");
rootServiceDoc.addHeader("Accept", "application/rdf+xml");
rootServiceDoc.addHeader("OSLC-Core-Version", "2.0");
```

__9. 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(rootServiceDoc);
```

__10. The next line prints out the **status code** of the HTTP response

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

__11. 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);
```

__12. If the response is an error, then the code releases any created resources:

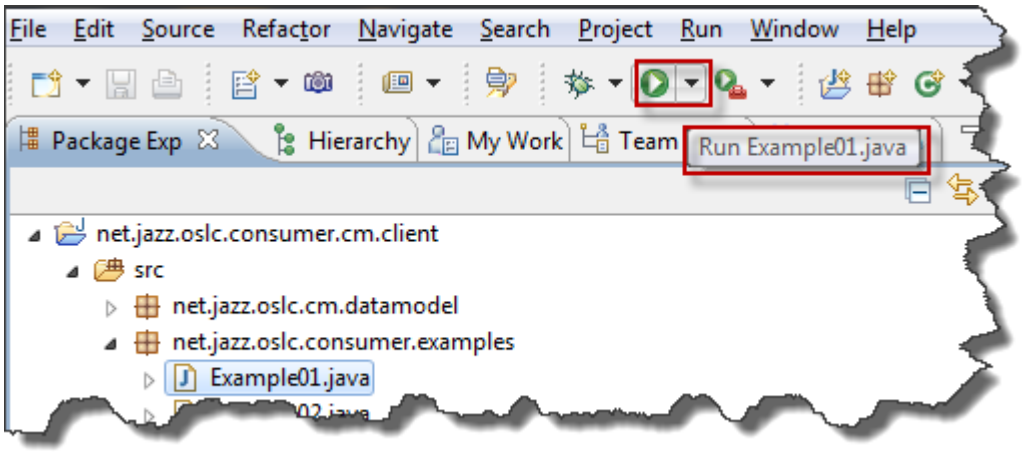
```
// Release allocated resources
response.getEntity().consumeContent();
```

__13. Finally, the last line shuts down the HTTP client by releasing the connections.

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

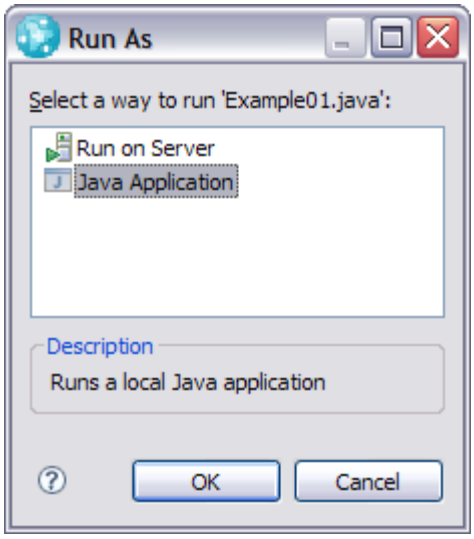
__14. Now that we have a good understanding of the code, lets run it.

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

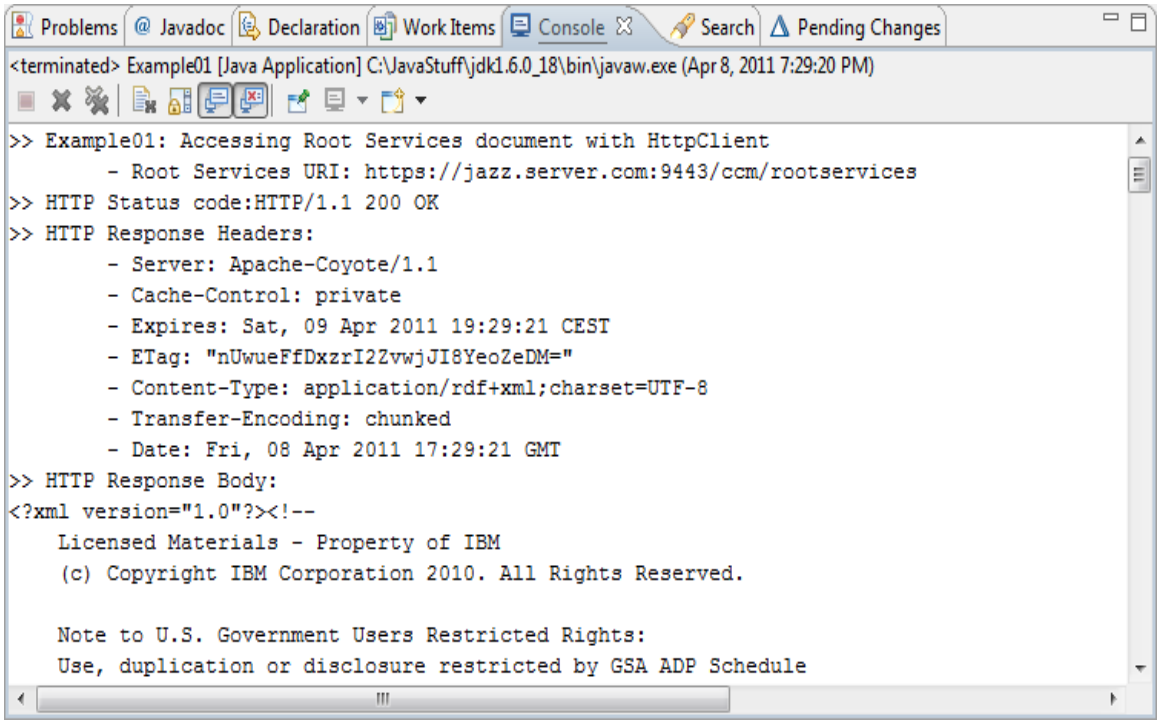


__b. Press the **Run as…** button located on toolbar

__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.



The log should look like this:

```
>> Example01: Accessing Root Services document with HttpClient
    - Root Services URI: https://jazz.server.com:9443/ccm/rootservices
>> HTTP Status code:HTTP/1.1 200 OK
>> HTTP Response Headers:
    - Server: Apache-Coyote/1.1
    - Cache-Control: private
    - Expires: Sat, 09 Apr 2011 19:29:21 CEST
    - ETag: "nUwueFfDxzrI2ZvwjJI8YeoZeDM="
    - Content-Type: application/rdf+xml;charset=UTF-8
    - Transfer-Encoding: chunked
    - Date: Fri, 08 Apr 2011 17:29:21 GMT
>> HTTP Response Body:
<?xml version="1.0"?><!--
   Licensed Materials - Property of IBM
   (c) Copyright IBM Corporation 2010. All Rights Reserved.

   Note to U.S. Government Users Restricted Rights:
   Use, duplication or disclosure restricted by GSA ADP Schedule
   Contract with IBM Corp.
 -->

<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:jdb="http://jazz.net/xmlns/prod/jazz/dashboard/1.0/"
    xmlns:jp06="http://jazz.net/xmlns/prod/jazz/process/0.6/"
     xmlns:jp="http://jazz.net/xmlns/prod/jazz/process/1.0/"
     xmlns:jtp="http://jazz.net/xmlns/prod/jazz/jtp/0.6/"
    xmlns:ju="http://jazz.net/ns/ui#"
    xmlns:oslc="http://open-services.net/ns/core#"
    rdf:about="https://jazz.server.com:9443/ccm/rootservices">

…/...

</rdf:Description>
```

3.2 Retrieve the Service Provider catalog using Xpath

This example shows how an OSLC consumer can retrieve an element or an attribute of an element in an XML representation, such as 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 (__) 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
foo	**Selects all the child nodes named foo.**
/foo	**Selects from the root node the nodes named foo.**
//foo	**Selects nodes named foo no matter where they are in the document**
@att	**Selects the attribute node named att.**
foo/bar	**Selects all the nodes named bar having a parent node named foo.**
//foo[@att]	**Select all the nodes named foo no matter where they are having an attribute named att.**
//foo [@att="val”]	**Select all the nodes named foo no matter where they are having an attribute named att with the value val.**

XPath tester

if you need to test an XPath expression against some XML code, there are several interesting testers on the web.

- We found a first one which requires to upload the XML file to parse: http://www.whitebeam.org/library/guide/TechNotes/xpathtestbed.rhtm
- We found another one which accepts a copy/paste of the XML content to parse: http://www.futurelab.ch/xmlkurs/xpath.en.html

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:// jazz.server.com:9443/ccm/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 root 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
XPathFactory factory = XPathFactory.newInstance();
XPath xpath = factory.newXPath();
xpath.setNamespaceContext(
            new NamespaceContextMap(new String[]
                  {    "rdf", "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
                       "oslc_cm","http://open-services.net/xmlns/cm/1.0/"}));

// Parse the response body to retrieve the catalog URI
InputSource source = new InputSource(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 lines create an instance of an XPath evaluation environment. This environment is set up to be able to parse and understand nodes using the rdf and oslc-cm namespaces.

```
XPathFactory factory = XPathFactory.newInstance();
XPath xpath = factory.newXPath();
```

__3. The next instruction sets the namespace context. This interface is used to retrieve the namespaces corresponding to the prefixes used by the XPath. In this example, the XPath is `/rdf:Description/oslc_cm:cmServiceProviders/@rdf:resource`. It references two prefixes “rdf” and “oslc_cm”. So the NamespaceContext defines the mapping between these prefixes and the namespaces (xlmns declarations) used in the document that the XPath will parse.

```
xpath.setNamespaceContext(
        new NamespaceContextMap(new String[]
                { "rdf",      "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
                  "oslc_cm",  "http://open-services.net/xmlns/cm/1.0/"}));
```

__4. The next lines parse the response body (response.getEntity().getContent()) using the `XPath.evaluate` method. This method takes 3 arguments:

```
// Parse the response body
InputSource source = new InputSource(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.InputSource` 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. 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.

__5. 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());
```

__6. Now that we have a good understanding of the code, lets run it.

__a. Double-click the **Example02.java** file in the **Package Explorer**.

__b. First change the RTC Server URL if necessary by changing the value of the String **server** variable. For example if your CCM server is accessed as https://jazz.server.com:9443/ccm then the initialization of the **server** variable would be

```
String server = "https://jazz.server.com:9443/ccm";
```

__c. Make sure you save the file (**Ctrl-S)**.

__d. Run the sample as a Java application

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

```
>> Example02: Retrieving the Service Provider catalog URI with XPath
        - Root Services URI: https://jazz.server.com:9443/ccm/rootservices
        - Catalog Path: /rdf:Description/oslc_cm:cmServiceProviders/@rdf:resource
>> HTTP Status code:HTTP/1.1 200 OK
>> Catalog URI: https://jazz.server.com:9443/ccm/oslc/workitems/catalog
```

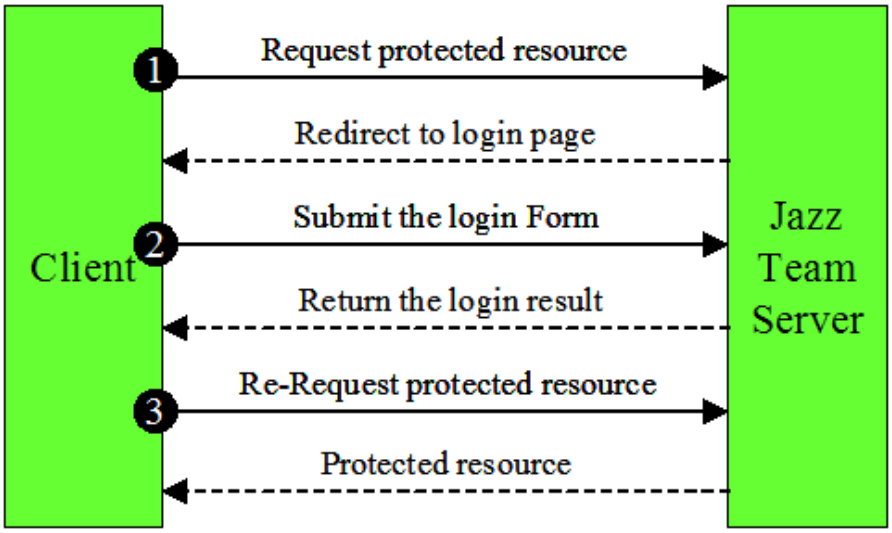
3.3 Jazz Form-based Authentication

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 (11).

__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.

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

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 = (HttpGet)(request.clone());
            return  httpClient.execute(documentGet2);
        }
    }
}
return  documentResponse;
```

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

```
// 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 = (HttpGet)(request.clone());
        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:

```
// Setup the catalog request
HttpGet catalogDoc = new HttpGet(serviceProvidersCatalog);
catalogDoc.addHeader("Accept", "application/xml");
catalogDoc.addHeader("OSLC-Core-Version", "2.0");

// Access to the Service Providers catalog
HttpResponse catalogResponse
        = HttpUtils.sendGetForSecureDocument(server, catalogDoc, login, password, httpclient);
if (catalogResponse.getStatusLine().getStatusCode() == 200) {
    // Define the XPath evaluation environment
    XPath xpath2 = factory.newXPath();
    xpath2.setNamespaceContext(
                new NamespaceContextMap(new String[]
                        {       "oslc",         "http://open-services.net/ns/core#",
                                "dcterms",      "http://purl.org/dc/terms/"}));

    // Parse the response body to retrieve the Service Provider
    source = new InputSource(catalogResponse.getEntity().getContent());
    NodeList titleNodes = (NodeList) (xpath2.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/xml`.

```
// Access to the Service Providers catalog
HttpResponse catalogResponse
        = HttpUtils.sendGetForSecureDocument(server, catalogDoc, 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:

```
// Define the XPath evaluation environment
XPath xpath2 = factory.newXPath();
xpath2.setNamespaceContext(
        new NamespaceContextMap(new String[]
            {   "oslc",         "http://open-services.net/ns/core#",
                "dcterms",      "http://purl.org/dc/terms/"}));

// Parse the response body to retrieve the Service Provider
source = new InputSource(catalogResponse.getEntity().getContent());
NodeList titleNodes
    = (NodeList) (xpath2.evaluate(
                "//oslc:ServiceProvider/dcterms: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"?>
<rdf:RDF xmlns:dcterms="http://purl.org/dc/terms/" xmlns:ns1="http://jazz.net/xmlns/prod/jazz/process/1.0/"
    xmlns:oslc="http://open-services.net/ns/core#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <oslc:ServiceProviderCatalog rdf:about="https://jazz.server.com:9443/ccm/oslc/workitems/catalog">
        <dcterms:title rdf:parseType="Literal">Project Areas</dcterms:title>
        <dcterms:publisher>
            .../...
        </dcterms:publisher>
        <oslc:domain rdf:resource="http://open-services.net/ns/cm#"/>
        <oslc:serviceProvider>
            <oslc:ServiceProvider rdf:about= "https://jazz.server.com:9443/ccm/oslc/.../services.xml">
                <dcterms:title rdf:parseType="Literal">JUnit Project</dcterms:title>
                <oslc:details
                    rdf:resource="https://jazz.server.com:9443/ccm/process/..."/>
                <ns1:consumerRegistry rdf:resource="https://jazz.server.com:9443/ccm/process/.../links"/>
            </oslc:ServiceProvider>
        </oslc:serviceProvider>
    </oslc:ServiceProviderCatalog>
</rdf:RDF>
```

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

```
"/rdf:RDF/oslc:ServiceProviderCatalog/oslc:serviceProvider/oslc:ServiceProvider/dcterms:title"
```

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

```
"//oslc:ServiceProvider/dcterms:title"
```

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

```
//dcterms:title
```

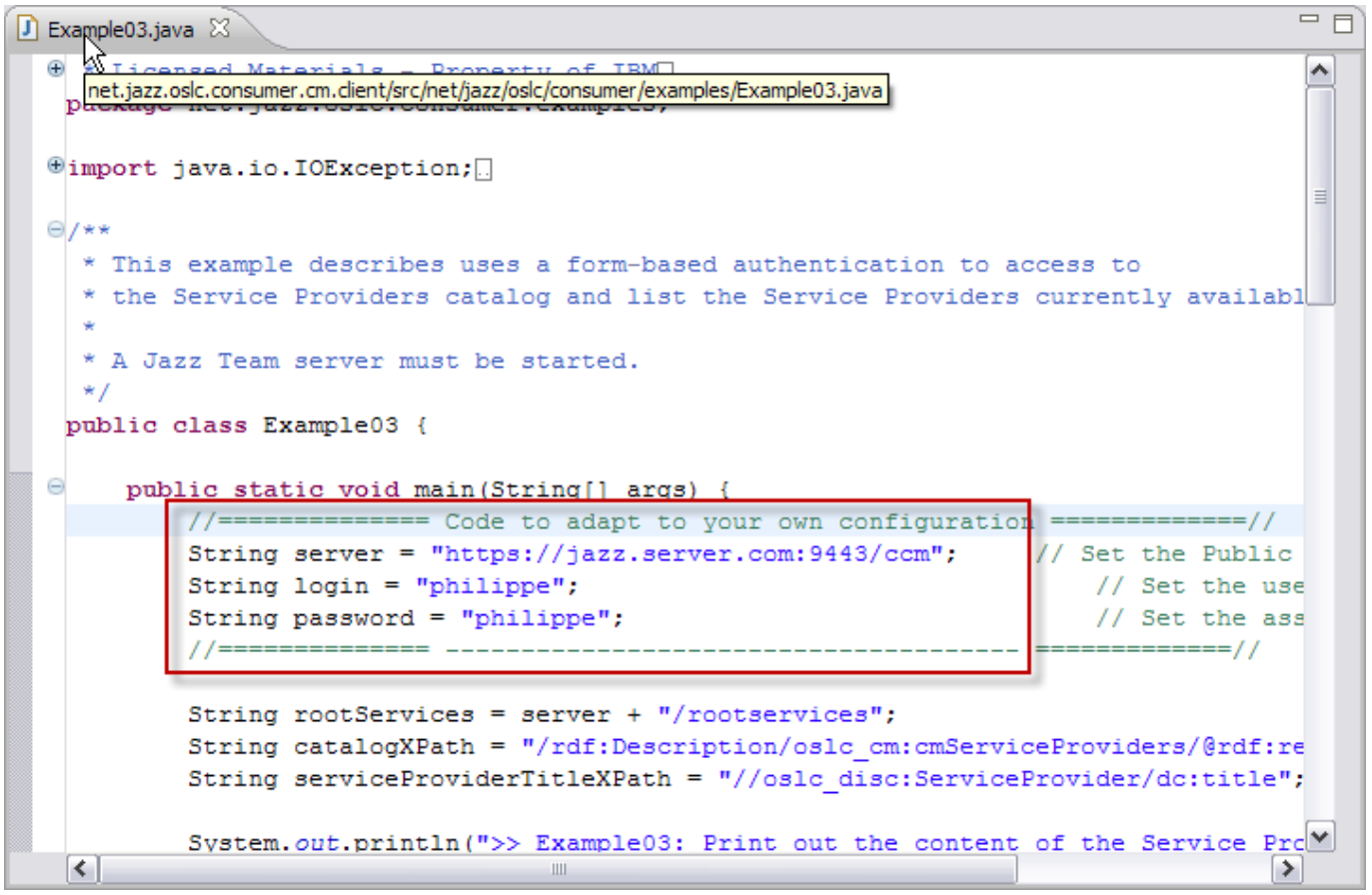
Actually, the above expression means: select all the `dcterms:title` nodes no matter where they are.

__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. *If necessary* first change the values of the following variables to match your setup:

```
String server = https://jazz.server.com:9443/ccm;
String login = "ADMIN";
String password = "ADMIN";
```



__c. Make sure you save the file (**Ctrl-S)**.

__d. Run it as Java application.

__e. The Console view will print out the titles of the Project Areas currently stored in the Jazz Team Server:.

```
>> Example03: Print out the content of the Service Providers catalog
       - Root Services URI: https://jazz.server.com:9443/ccm/rootservices
       - Service Providers catalog XPath expression: /rdf:Description/oslc_cm:cmServiceP
       - Service Provider title XPath expression: //oslc:ServiceProvider/dcterms:title
       - Login: philippe
       - Password: philippe
>> Project Areas:
>>       - JUnit Project
```

3.4 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. We will not spend time on the first 3 steps which have been explained during the previous examples. 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 (`dcterms:title, dcterms:identifier, dcterms:type, dcterms:description, dcterms:subject, dcterms:creator, dcterms:modified, rdf:about`):

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

HttpGet query = new HttpGet(queryWIs);
query.addHeader("Accept", "application/xml");
query.addHeader("OSLC-Core-Version", "2.0");

HttpResponse response = HttpUtils.sendGetForSecureDocument(server, query, login, password, httpclient);
```

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

```
// Define the XPath evaluation environment
XPathFactory factory = XPathFactory.newInstance();
XPath xpath = factory.newXPath();
String wiXPath = "//oslc_cm:ChangeRequest";
xpath.setNamespaceContext(
        new NamespaceContextMap(new String[]
                {"oslc_cm", "http://open-services.net/ns/cm#"}));
// Parse the response body to retrieve the Change Request DOM node
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:about");
return new ChangeRequest(wiURI, wiNode);
```

__2. Step (5): During this step, the client modifies the Change Request using the provided API. Actually, it concatenates the current time stamp at the end of the description.

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

__3. Step (6): In this step, the client uses the Update Change Request OSLC-CM API to update the modified Change Request. For that, the client needs to send an **HTTP PUT** message with the Change Request URI, the `content-type` header must be set to `application/xml` and the request's body must contain the XML representation of the modified Change Request.

This behavior is implemented by the `updateChangeRequest method`:

```
private HttpResponse updateChangeRequest(final ChangeRequest cr) throws InvalidCredentialsException,
IOException
{
        // 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();
           }
        };

        HttpEntity entity = new EntityTemplate(cp);
        HttpPut put = new HttpPut(cr.getUri());
        put.addHeader("Accept", "application/rdf+xml");
        put.addHeader("Content-type", "application/xml");
        put.setEntity(entity);

        // Call the PUT method against the Change Request URI
        return HttpUtils.sendPutForSecureDocument(server, cp, put, 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.

__4. 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. *If necessary* first change the values of the following variables to match your setup:

```
String server = https://jazz.server.com:9443/ccm;
String login = "ADMIN";
String password = "ADMIN";
String projectAreaName = "JUnit Project";
String changeRequestID = "1";
```

__c. Make sure you save the file (**Ctrl-S)**.

__d. Run the example as a Java application.

__e. The Console view should provide the following output:

```
>> Example04: Update a Change Request
       - Server: https://jazz.server.com:9443/ccm
       - Login: philippe
       - Password: philippe
>> Service Providers Catalog: https://jazz.server.com:9443/ccm/oslc/workitems/catalog
>> Project Area [JUnit Project]: https://jazz.server.com:9443/ccm/oslc/contexts/_uCjawFTUEeCod6NdTaW--w/workitems/servi
>> Simple Query URL: https://jazz.server.com:9443/ccm/oslc/contexts/_uCjawFTUEeCod6NdTaW--w/workitems
>> Change Request URL for [1]: https://jazz.server.com:9443/ccm/resource/itemName/com.ibm.team.workitem.WorkItem/1
>> Update Response Status code:HTTP/1.1 200 OK
>> Update Response Headers:
       - Server: Apache-Coyote/1.1
       - Cache-Control: max-age=0, must-revalidate
       - Expires: Fri, 08 Apr 2011 22:56:45 GMT
       - ETag: "_emgTEGIzEeCsw90ZRFFmEQ"
       - Last-Modified: Fri, 08 Apr 2011 22:56:45 GMT
       - Vary: Accept, Accept-Language
       - Content-Type: application/rdf+xml;charset=UTF-8
```

__5. Copy the URL of the Work Item from the Console view. It should be located after the label:
`Change Request URL for [1].`

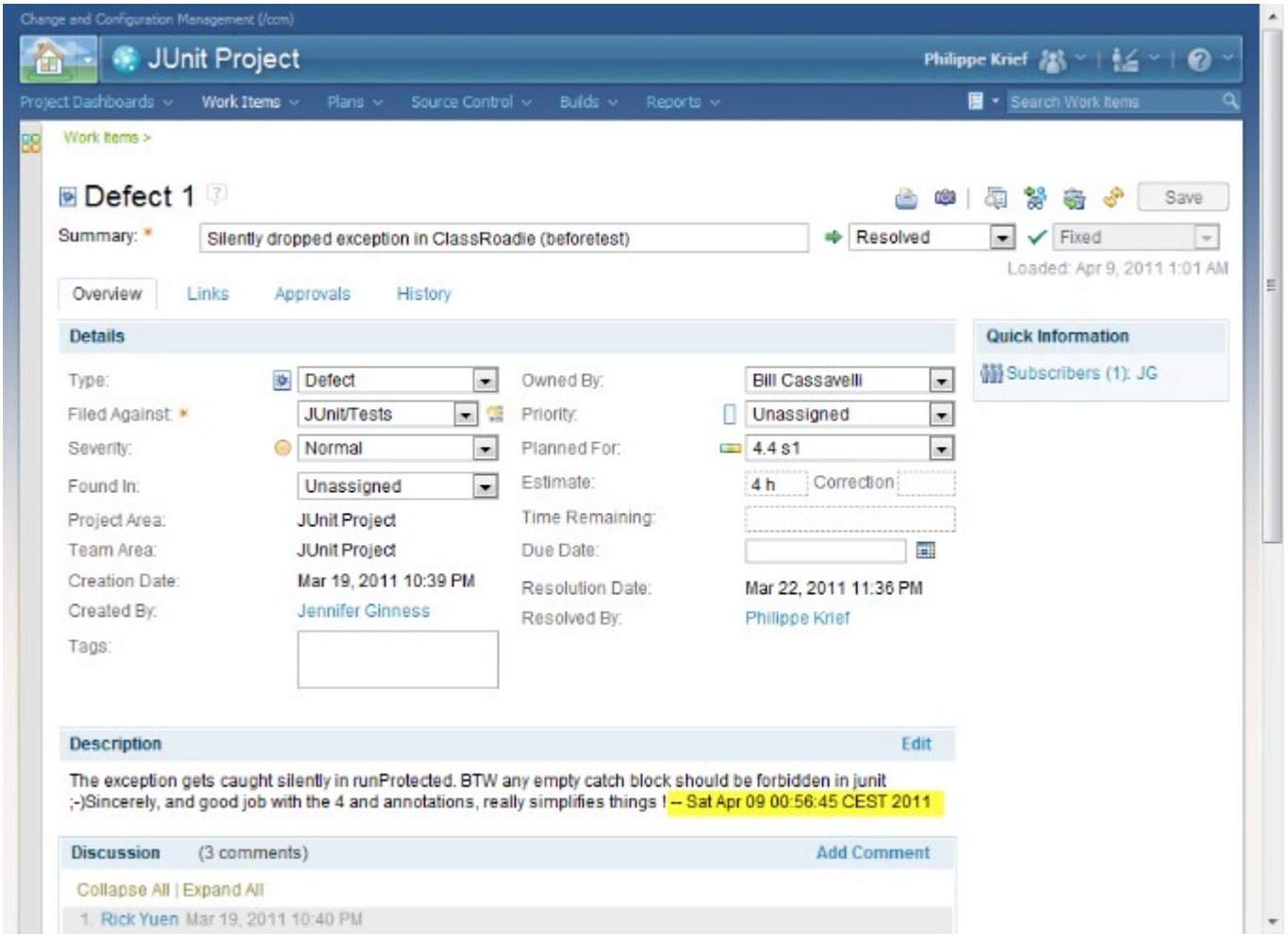
__6. Open the **Firefox** internet browser by double-clicking the **Mozilla Firefox** shortcut 

on the **Windows Desktop.**

__7. Paste the copied URL into the navigation field of your browser and Press Enter.

__8. If you were not already logged in, the web UI will display the login dialog.
Login with your current admin login.

__9. 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:



3.5 Build your first OSLC servlet

At this point of the workshop, you have all the knowledge to build a web application which will take advantage of the OSLC-CM consumer API.

In this example, we will implement a simple Web portal listing all the Project Areas of a designated OSLC-CM server.

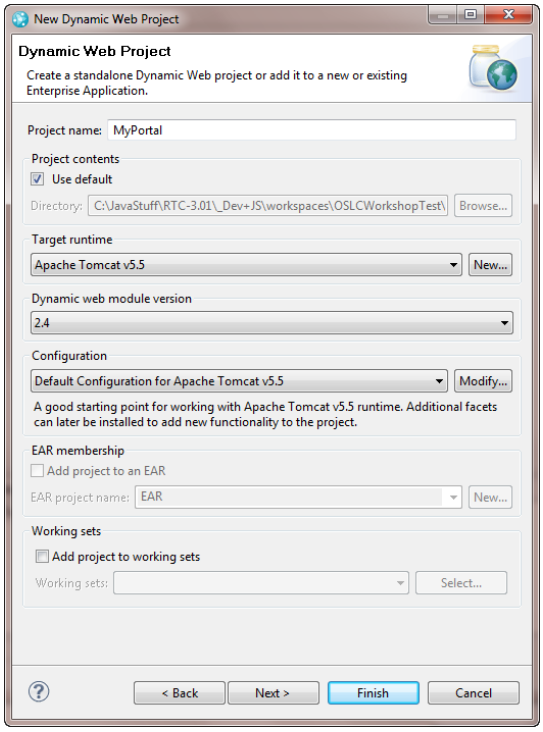
__3. Launch the Eclipse client

__4. Create an Eclipse Dynamic Web project. Unlike with static Web projects, dynamic Web projects enable you to create resources such as JavaServer Pages and servlets which is what we need.

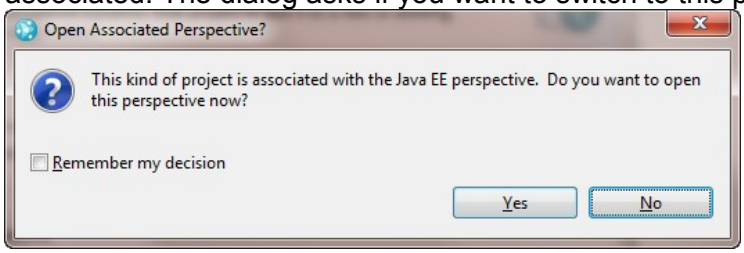
__a. From the **File** menu, select **New > Other...**

__b. From the **New** dialog wizard select the **Dynamic Web Project** item and press **Next**.

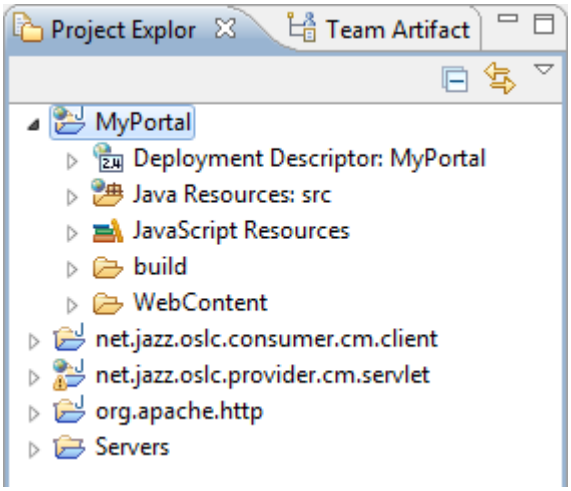
__c. For the project name, type **MyPortal** and press **Finish**.



__d. A dialog will pop up telling you that for this kind of project, the Java EE perspective is associated. The dialog asks if you want to switch to this perspective. Answer **Yes**.



__e. At this point, you should have a new eclipse project named **MyPortal** listed in the **Project Explorer** view of the **Java EE** perspective.

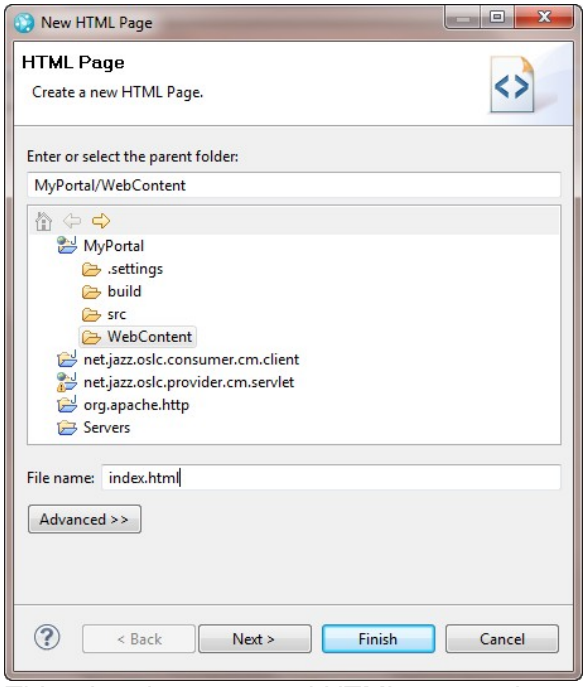


__6. Create an HTML page to connect to an OSLC server and display its Project Areads. For that we will need to know the Public URI of the server, and the login / password for authentication.

__a. Click on the **WebContent** folder of your new Eclipse project.

__b. From the context menu, select **New > HTML Page** to create a HTML page

__c. In the file name field, type **index.html** and press **Finish**



This wizard creates and HTML page using a default template and open the HTML editor on it.

__d. Between the **<title>** tags, replace **Insert title here** with **My Project Area portal**

__e. Between the **<body>** tags, insert the following HTML snipped code:

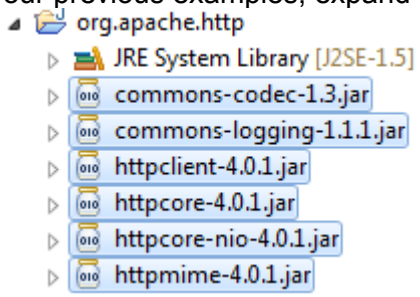
```
<h2>Connect to:</h2>
<form action="ListOfProjectAreas" method="get">
    <table>
        <tr><td>Public URI:</td> <td><input type="text" name="publicURI" size="40"/></td></tr>
        <tr><td>Login:</td> <td><input type="text" name="login" size="20"/></td></tr>
        <tr><td>Password:</td> <td><input type="password" name="password" size="20"/></td></tr>
        <tr></tr>
        <tr><td><input type="submit" value="Submit"/></td></tr>
    </table>
</form>
```

Please notice the **ListOfProjectAreas** value of the **action** attribute of the form. It designates the name of the servlet we will implement in the next step.

Please notice also the method we will use to connect to the server (**get**), it designates the method called in the servlet.

__6. Add the libraries that your servlet will require.

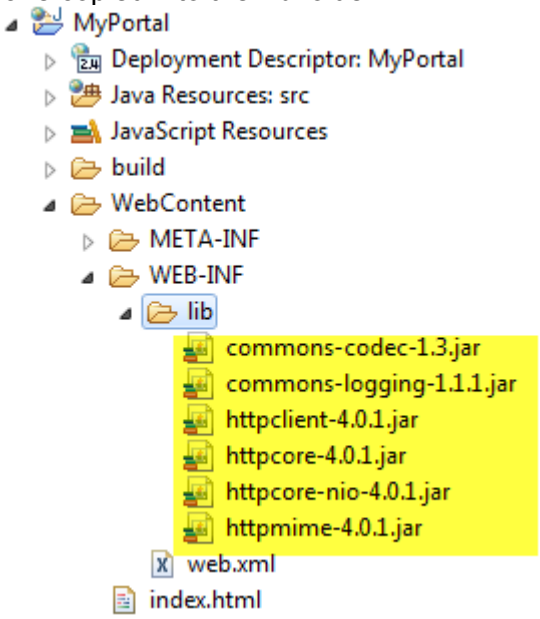
__a. Because we will use in the servlet the same Apache HTTP client APIs that we used in our previous examples, expand the org.appache.http project and select the listed jars:



__b. From the context menu, select **Copy**.

__c. In the Web project (**MyPortal**), select the **WebContent/WEB-INF/lib** folder.

__d. From the context menu, select **Paste**. At this point you should retrieve all the jars you have copied into the **lib** folder:



__5. It is time to create the servlet which will connect to the OSLC server, retrieve the list of Project Areas and return an HTML page showing this list.

__a. From the context menu of the **MyPortal** Web project, select **New > Servlet**.

__b. For the **Java package** of the servlet, type: **net.jazz.oslc.consumer.cm.servlets**

__c. For the Class name of the servlet, type: **ListOfProjectAreas**. As mentioned earlier, this name should be the same as the one that you specify as the value of the action attribute in the form.

__d. Press **Finish**. This wizard creates a new class, subclass of `javax.servlet.http.HttpServlet` and binds the URL **/ListOfProjectAreas** to the designated class: **net.jazz.oslc.consumer.cm.servlets.ListOfProjectAreas**.

FYI, this binding is described in the file **WebContent/WEB-INF/web.xml**.

__5. Write the code of your servlet

A Java editor is now open against this new servlet class. This class implements 2 default methods: **doGet**, to support the HTTP GET method and **doPost** to support the HTTP POST method. As we mentioned earlier, our form will submit its parameters using the HTTP GET method, which means that the doGet method will be called.

You have certainly noticed that the servlet we are implementing is supposed to implement most of the behavior described in the **Example03** (6.3 Jazz Form-based Authentication). So we will use this opportunity to copy/paste the code of the Example03.java into our servlet and adapt it to our needs.

__a. Edit the **Example03** example located in net.jazz.oslc.consumer.cm.client/src/net.jazz.oslc.consumer.examples/Example03.java

__b. Copy all the code contained in the main() method and paste it in the **doGet** method of your servlet.

__c. In the pasted code, instead of providing URL of the server and the credentials directly in the code, we will get them from the `HttpServletRequest`. So, replace the 3 lines:

```
String server = "https://jazz.server.com:9443/ccm";
String login = "philippe";
String password = "philippe";
```

with

```
String server = request.getParameter("publicURI").toString();
String login = request.getParameter("login").toString();
String password = request.getParameter("password").toString();
```

__d. Now, instead of printing the project area names on the System.out, we will build an HTML page which will display this list as the response to the request. So replace the loop located at the bottom on the code:

```
// Print out the title of each Service Provider
int length = titleNodes.getLength();
System.out.println(">> Project Areas:");
```

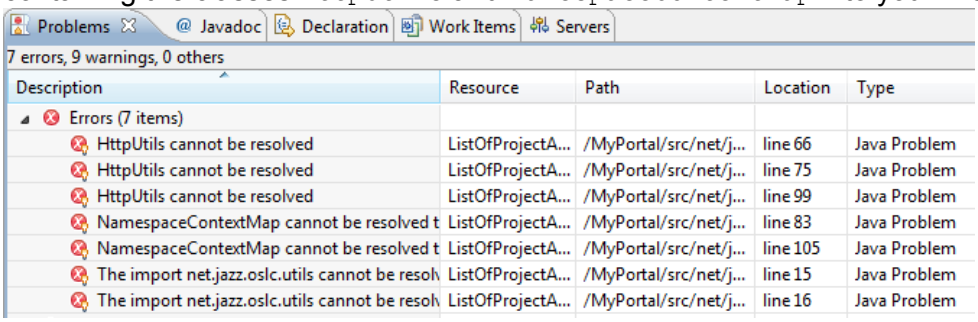
```
                for (int i = 0; i < length; i++) {
                    System.out.println(">> \t - "+ titleNodes.item(i).getTextContent());
                }
```

with

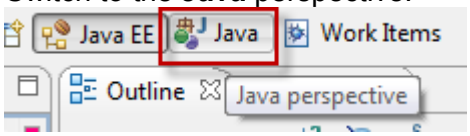
```
        // Build the HTML response
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();

    out.println("<html>");
    out.println("<head>");
    out.println("<h1>Project Areas in: "+server+"</h1>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>Project Areas</h1>");
    out.println("<ul>");

        // Print out the title of each Service Provider
        int length = titleNodes.getLength();
        Node node;
        Element element;
        String name;
        String url;
        for (int i = 0; i < length; i++) {
            node = titleNodes.item(i);
            name = node.getTextContent();
            element = (Element)(node.getParentNode());
            url = element.getAttribute("rdf:about");
            out.println("<li><a href=\""+url+"\">"+name+"</a></li>");
        }

    out.println("</ul>");
    out.println("</body>");
    out.println("</html>");

    out.close();
```

__e. Save the file.

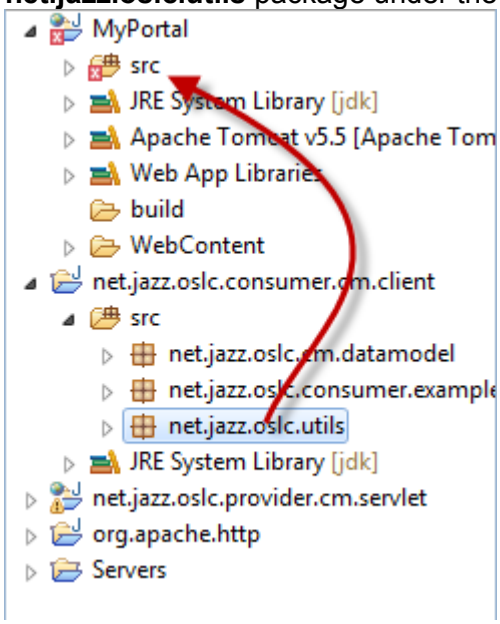
__6. At this point, you should still have a few errors. To fix them, you need to copy the package containing the classes `HttpUtils` and `NamespaceContextMap` into your Web project.



__a. Switch to the **Java** perspective.



__b. In the **net.jazz.oslc.consumer.cm.client** Eclipse project, select the package **net.jazz.oslc.utils** package under the **src** source folder.



__c. Access to the context menu and select **Copy**.

__d. In the **MyPortal** Web project, select the **src** source folder.

__e. In the context menu select **Paste**.
This action should fix all the errors that you had in your servlet.

__f. Switch back to the **Java EE** perspective.

__7. It is time to test the servlet.

__a. Your Jazz team Server should be running. If it is not the case, start it.

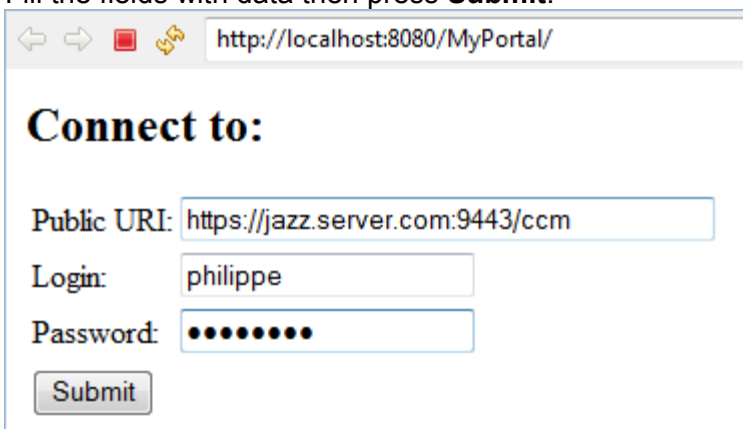
__b. Select the **MyPortal** web project.

__c. From the context menu select **Run As > Run on Server**

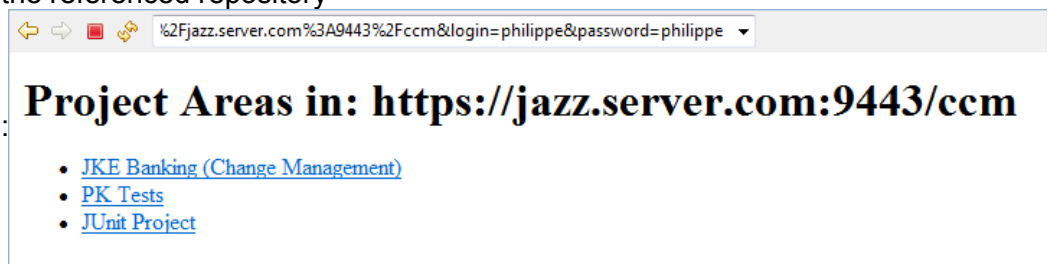
__d. The workbench will ask to select the Application Server to run against. Select your Tomcat server and press **Finish**.

The server should start and the **index.html** page should appear into the Eclipse embedded web browser.

__e. Fill the fields with data then press **Submit**.



__f. After few seconds, a new page will appear with the list of the Project Areas contained in the referenced repository



__g. At this point and using this servlet based pattern your can build type of web application consuming OSLC apis...

__8. Shut down the RTC server by running (<JazzTeamServer_Root_Folder>`\server\server.shutdown.bat`).

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

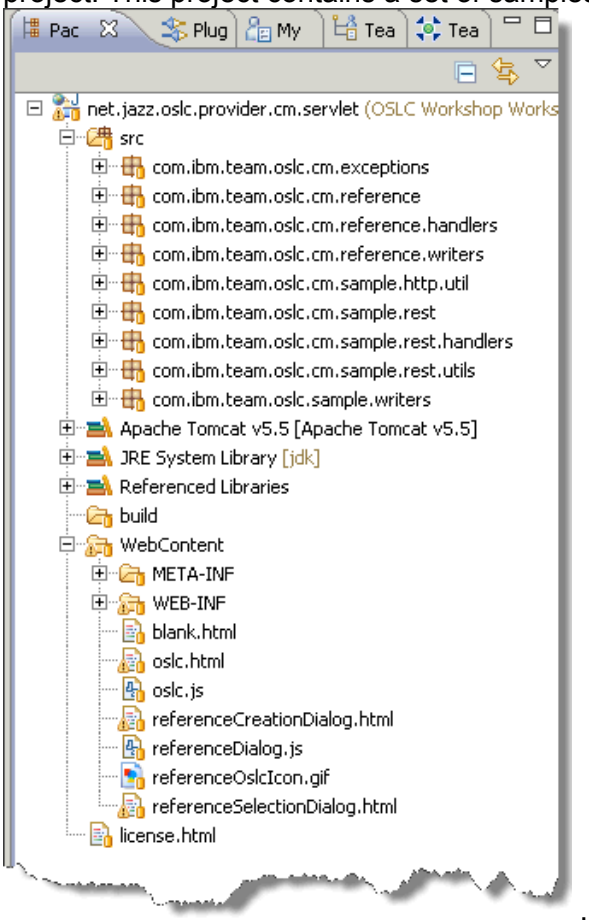
If you have not done the setup, see OSLC Lab 1, “Setting up for OSLC Development”.

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 Setting up the server runtime environment and running the sample server

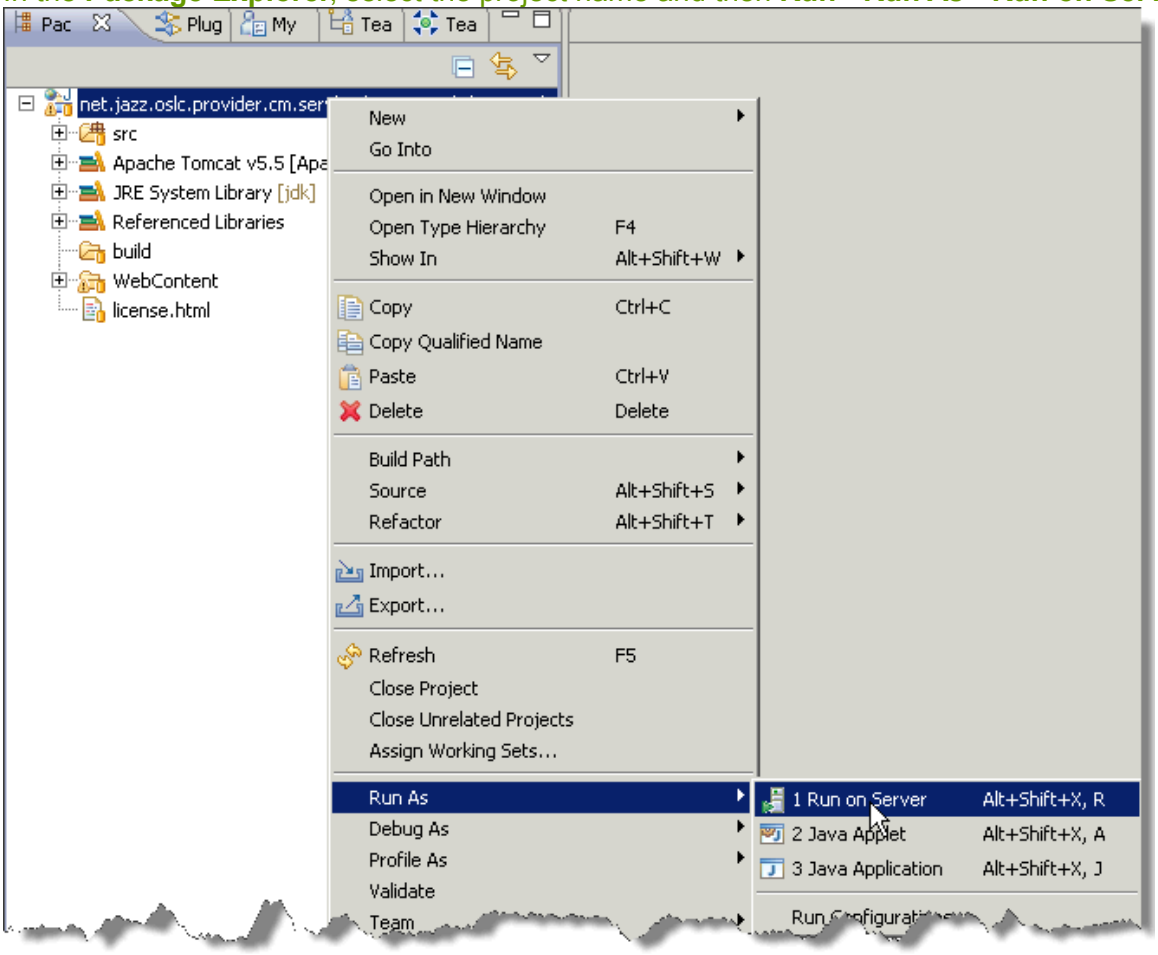
__1. In this lab we will mainly work with the `net.jazz.oslc.provider.cm.servlet` Eclipse project. This project contains a set of samples we will explain and run during this lab.



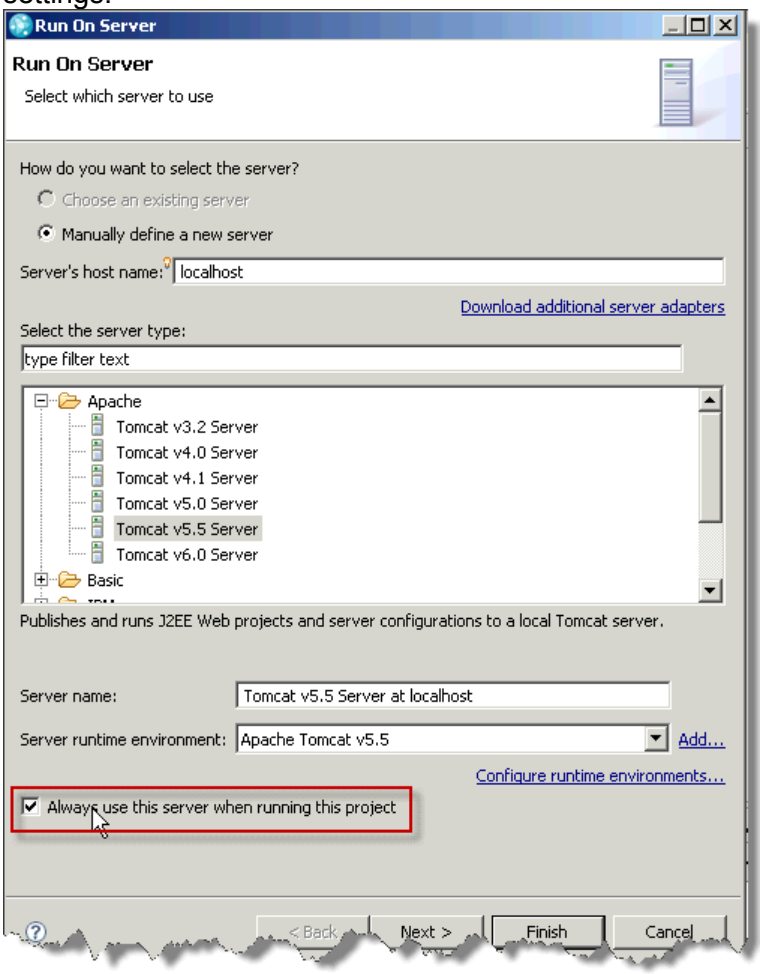
Apache Tomcat

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

__2. In the **Package Explorer**, select the project name and then **Run->Run As->Run on Server**



__3. 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 the download and then close the embedded browser.

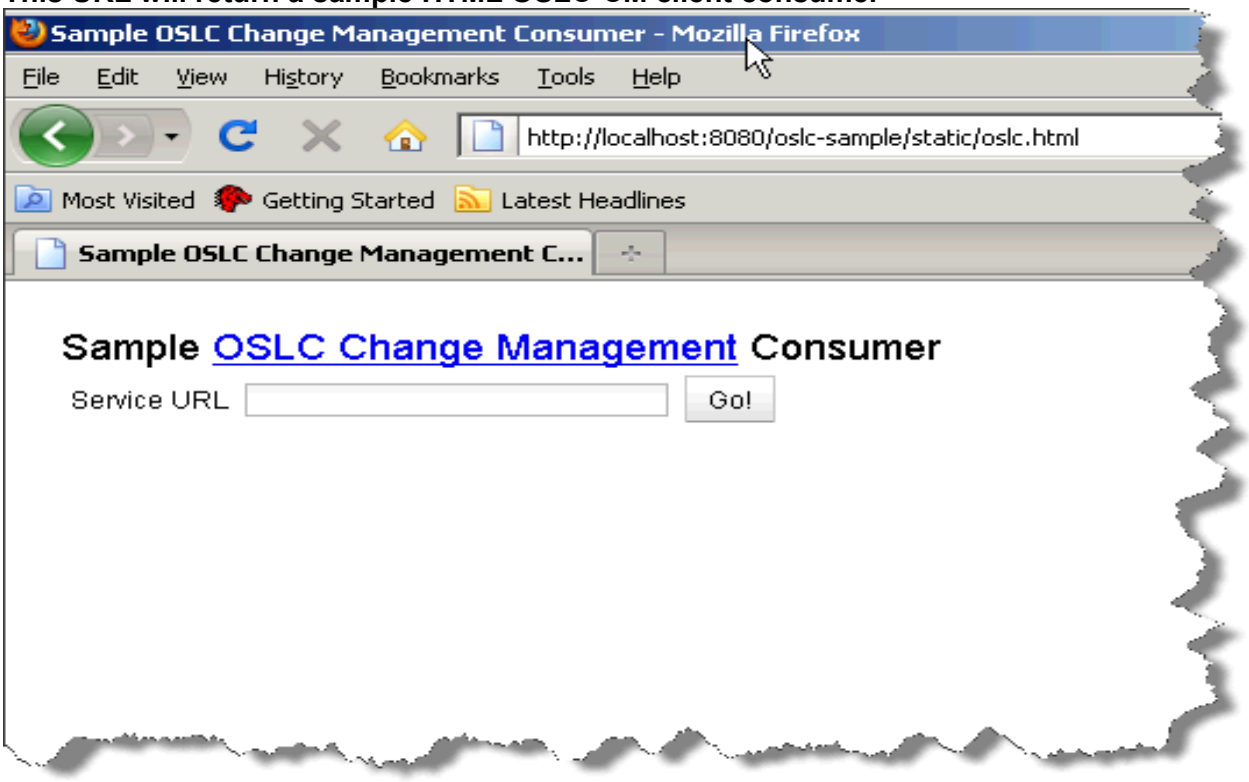
Restarting the sample server

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

__4. Open the **Firefox** internet browser by double-clicking the **Mozilla Firefox** shortcut 

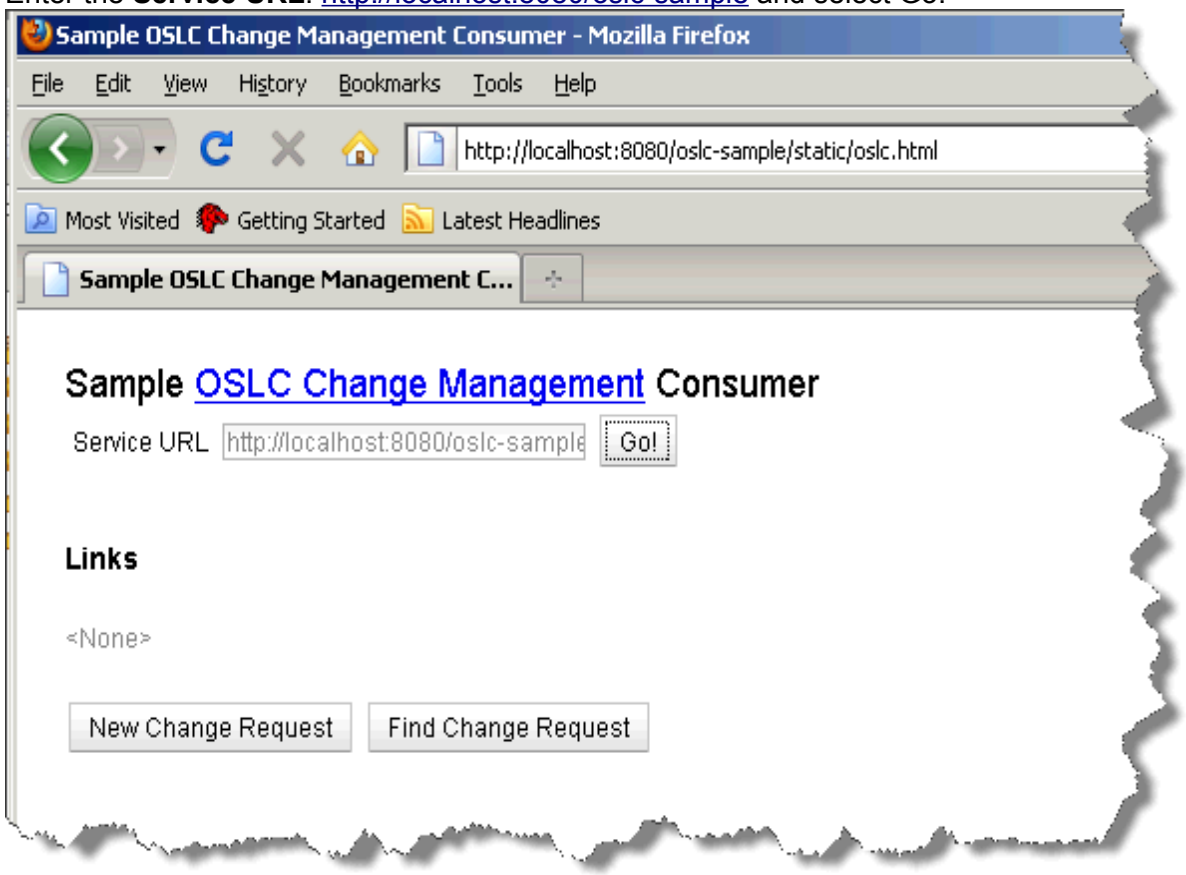
on the **Windows Desktop.**

__5. **Enter the URL: http://localhost:8080/oslc-sample/static/oslc.html**
This URL will return a sample HTML OSLC-CM client consumer



4.2 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

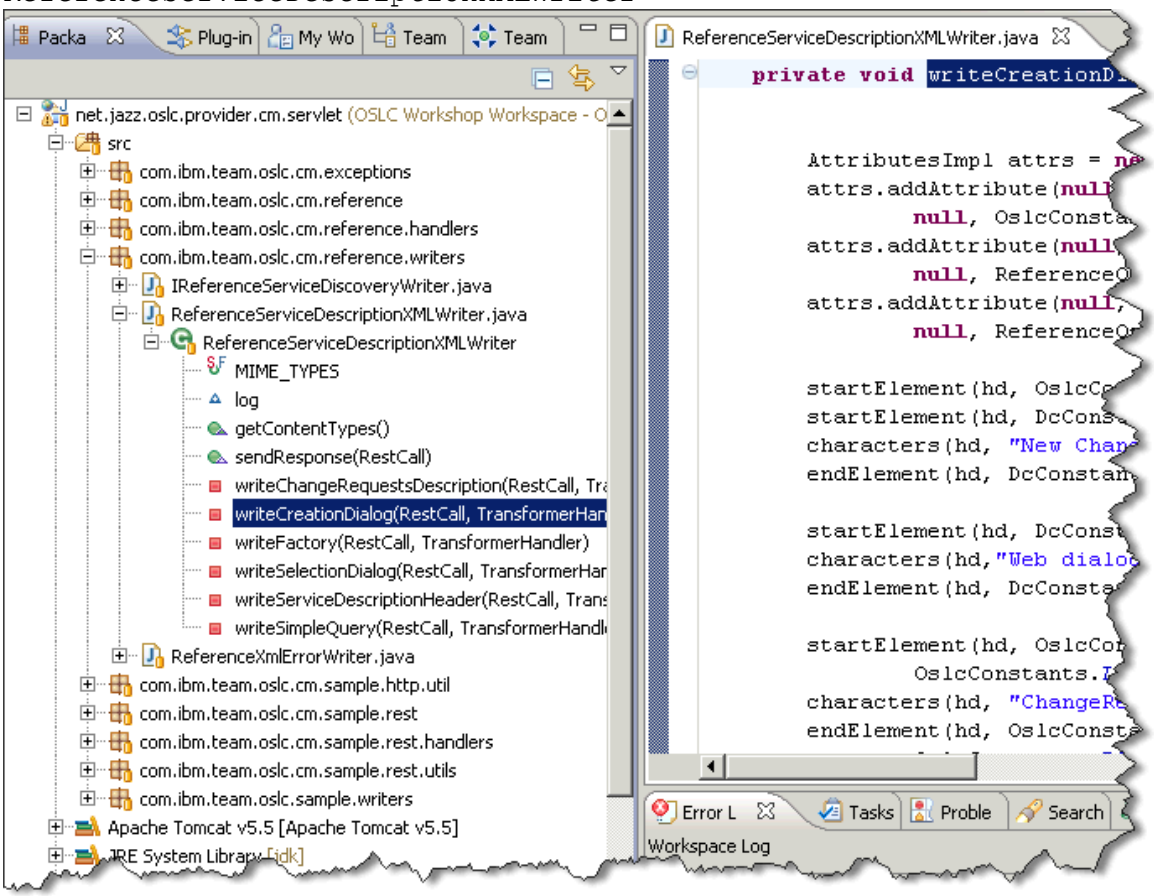
This simple sample highlights service discovery and access to delegated Web UI dialogs. It may be worth a few minutes to experiment with this. Nothing actually gets created on the server or deleted, reloading the page will reset to the beginning state.

4.3 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`

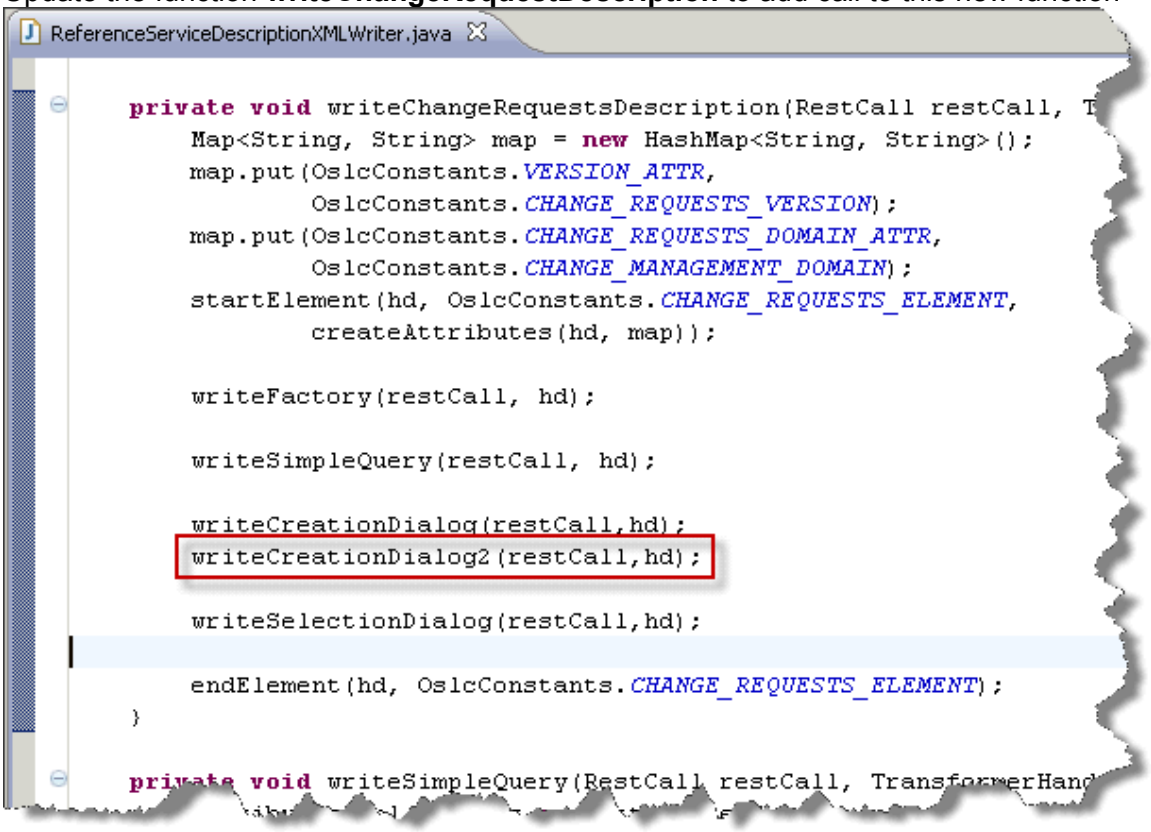


__2. Duplicate the function called **writeCreationDialog** and call the new function **writeCreationDialog2.** Use whatever method that works best for you.

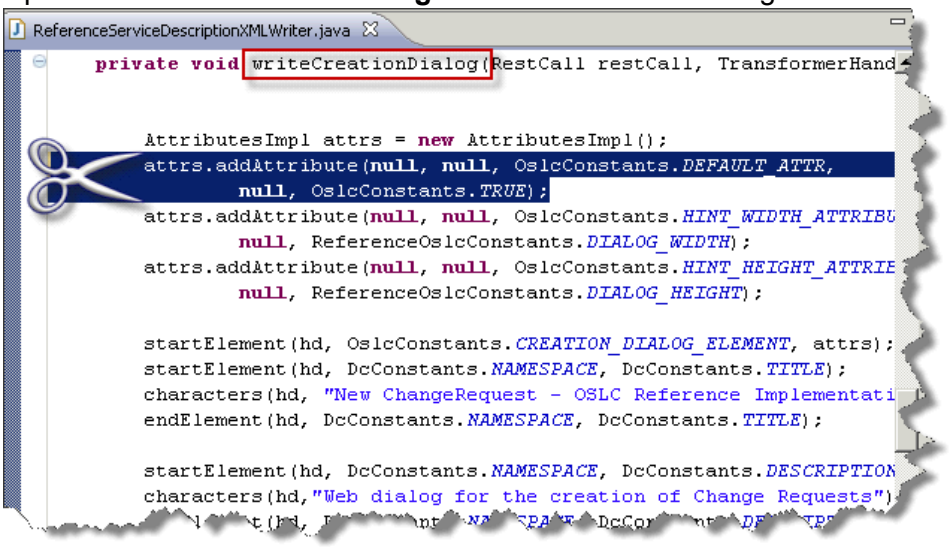
Coding best practices

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



__4. Update the **writeCreationDialog** function to remove setting the `default` attribute to `true`.



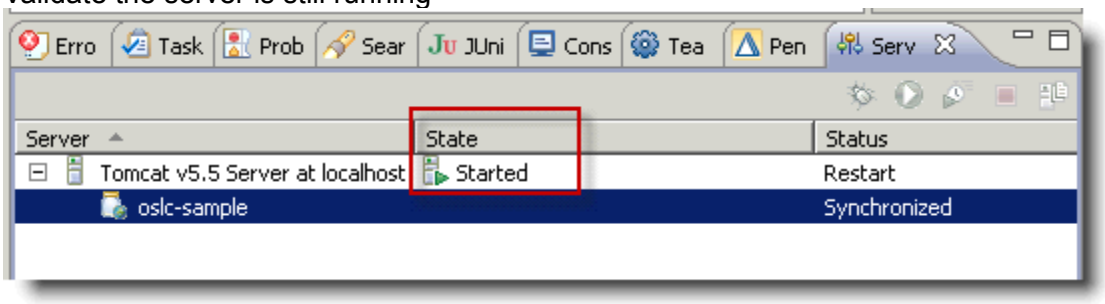
By doing this, the creation dialog entry in the service provider document will no longer be treated as the default dialog to use.

__5. Update the **writeCreationDialog2** function to provide updated names on various labels (shown in green rectangles)

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.

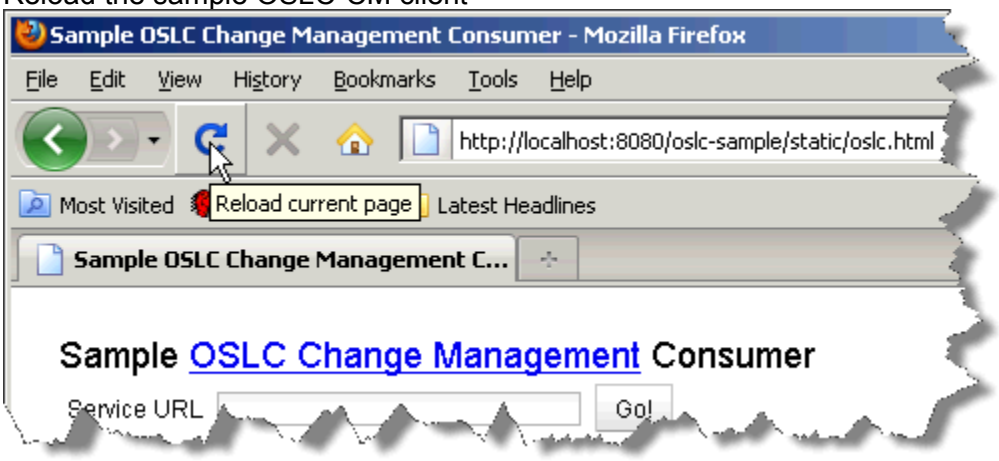
4.4 Test changes

__7. Validate the server is still running



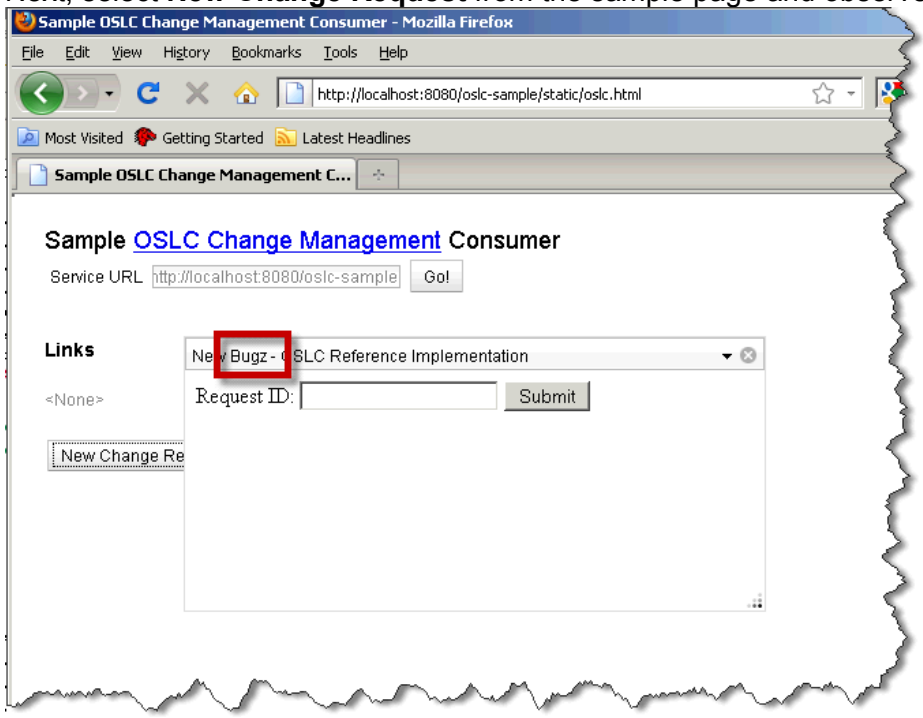
If the server is still running, no additional action is required. If the server is not running, see the section in Error: Reference source not found.

__8. Reload the sample OSLC-CM client



__9. Enter Service URL: http://localhost:8080/oslc-sample as before, selecting **Go!**.

__10. Next, select **New Change Request** from the sample page and observe the new dialog



Conclusion

You have completed lab 4. 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

Lab 5 Introduction to OSLC RM API

Lab Scenario

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

If your Jazz Team Server is not running, start it now (<JazzTeamServer_Root_Folder>`\server\server.startup.bat`).

OSLC-RM Release 2

OSLC RM v2.0 specification is available (http://open-services.net/bin/view/Main/RmSpecificationV2).

For compatibility issues between 1.0 and 2.0 specifications, please read http://open-services.net/bin/view/Main/RmSpecificationV2?sortcol=table;table=up;up=#Version_Compatibility_with_1_0_S

RRC 4.0 is backwards compatible with OSLC-RM 1.0.

If you wrote any code based on the OSLC-RM 1.0 release, it should still be working.

This workshop is based on the latest OSLC-RM specification, V2, to help you use the latest features of these APIs.

5.1 The Root Services document

There are many similarities between the OSLC CM and the RM specification.

Previous labs have already introduced some of the concepts that are common such as Service Provider element, Catalog URL, and REST Services available.

In addition, for OSLC RM, the same Jazz Foundation REST services apply as well such as the rootservices document, the whoami service, etc.

NOTE: Some screenshots below from the REST Client Add-on do not match the latest version. For example, the **FormattedXML** tab has been replaced with **Response Body (Preview)** tab in version 2.0.3 of the add-on.

The rootservices document for RM is https://jazz.server.com:9443/rm/rootservices

5.2 OSLC-RM services

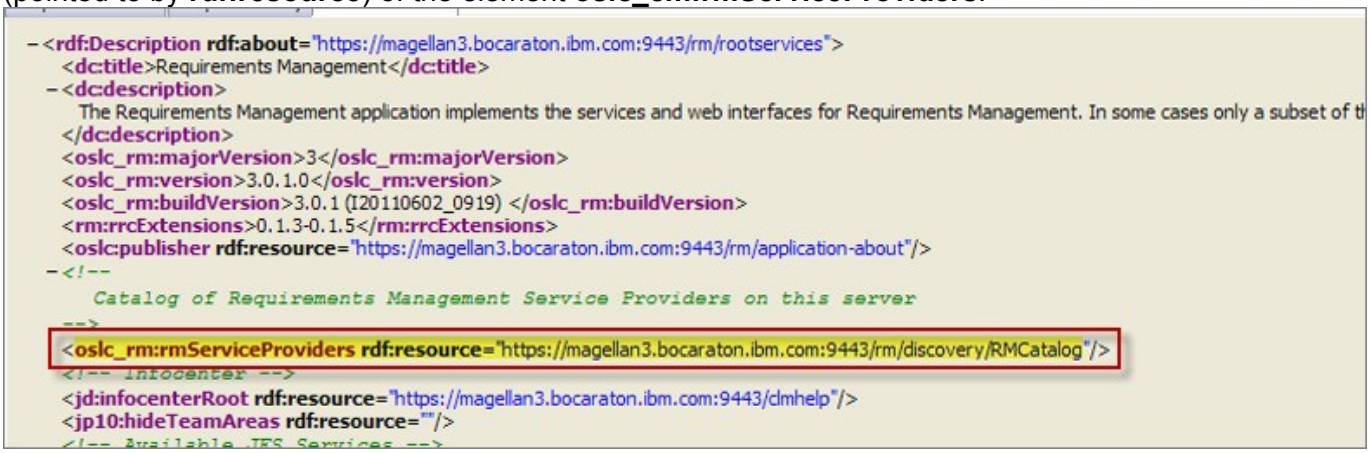
Let's introduce the **OSLC-RM** REST services implemented in the Requirements Management Server. The specification for these APIs is defined on the web site of the Open Services for Lifecycle Collaboration community (http://open-services.net/bin/view/Main/RmSpecificationV2).

NOTE: If you need more information on some of the concepts, refer to Lab 2.

The rootservices document for RM is https://jazz.server.com:9443/rm/rootservices

As we learned in the previous labs, the **Root Services** document is an XML informational resource that lists a set of REST services and capabilities.

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



__2. Copy this URL (https://jazz.server.com:9443/rm/discovery/RMCatalog) and paste it into the URL field of REST client.

Press the **Add Request Header** button and add the header key `accept` with the value `application/xml`. This header specifies you are expecting an XML response (and not an RDF/XML).
You can also double-click on the header to edit it and adapt it to the example.

Resource Format

The Specification says (http://open-services.net/bin/view/Main/RmSpecificationV2?sortcol=table;table=up#Resource_Formats):

When RM Consumers request:

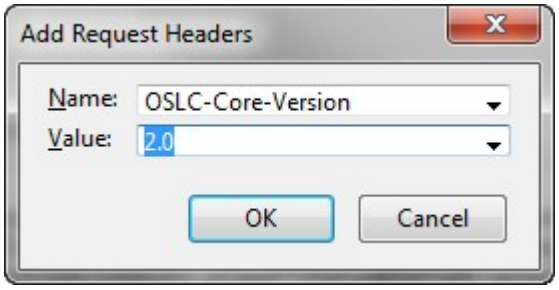
- **application/rdf+xml** RM Providers MUST respond with RDF/XML representation without restrictions.
- **application/json** RM Providers MUST respond with JSON representation as defined in the OSLC Core Representations Guidance.
- **application/xml** RM Provider MUST respond with OSLC-defined abbreviated XML representation as defined in the OSLC Core Representations Guidance
- **application/atom+xml** RM Provider MUST respond with Atom Syndication Format XML representation as defined in the OSLC Core Representations Guidance

The Atom Syndication Format XML representation SHOULD use RDF/XML representation without restrictions for the atom:content entries representing the resource representations.

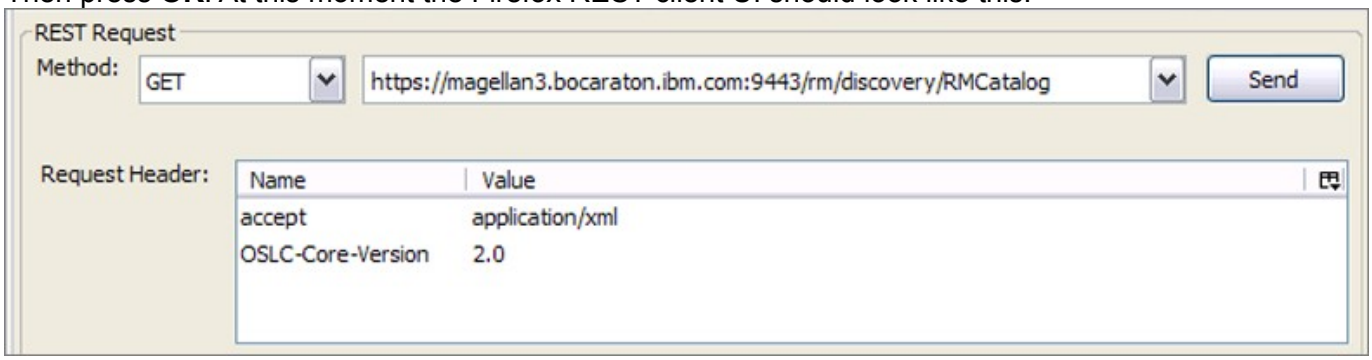
For the readability of this workshop we require as often as it is possible the **application/xml** media type. Feel free to use another resource format if it is more convenient for you.

NOTE: RRC only supports RDF/XML and XML resource formats.

Press the **Add Request Header** button a second time and declare a new header **OSLC-Core-Version: 2.0**. The header indicates to the OSLC producer that you are expecting a response based on the latest release of the OSLC-RM specification (http://open-services.net/bin/view/Main/OslcCoreSpecification#Specification_Versioning).

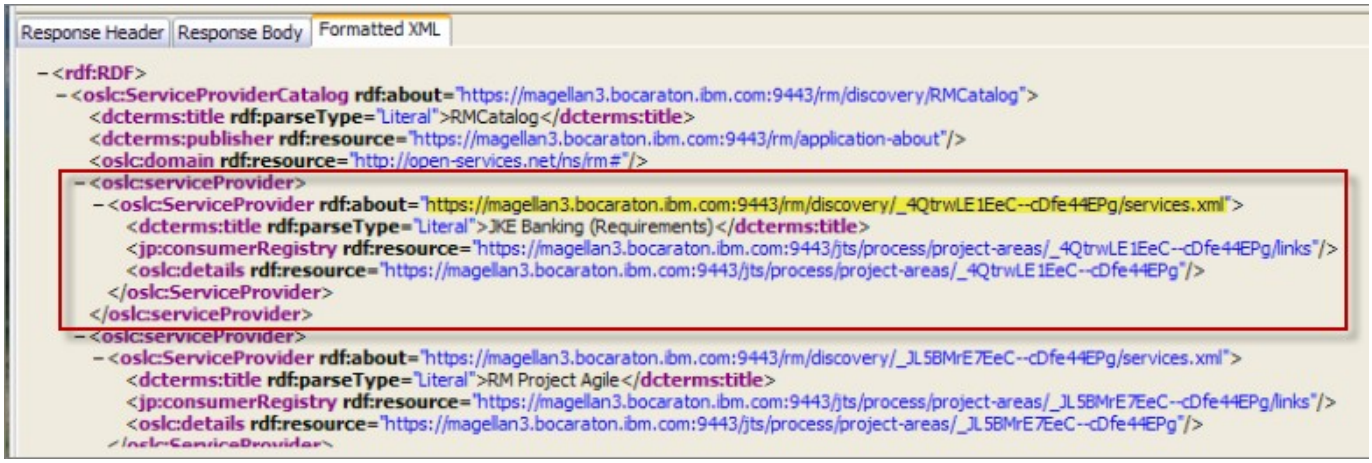


Then press **OK**. At this moment the Firefox REST client UI should look like this:



__3. 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 headers you have provided.

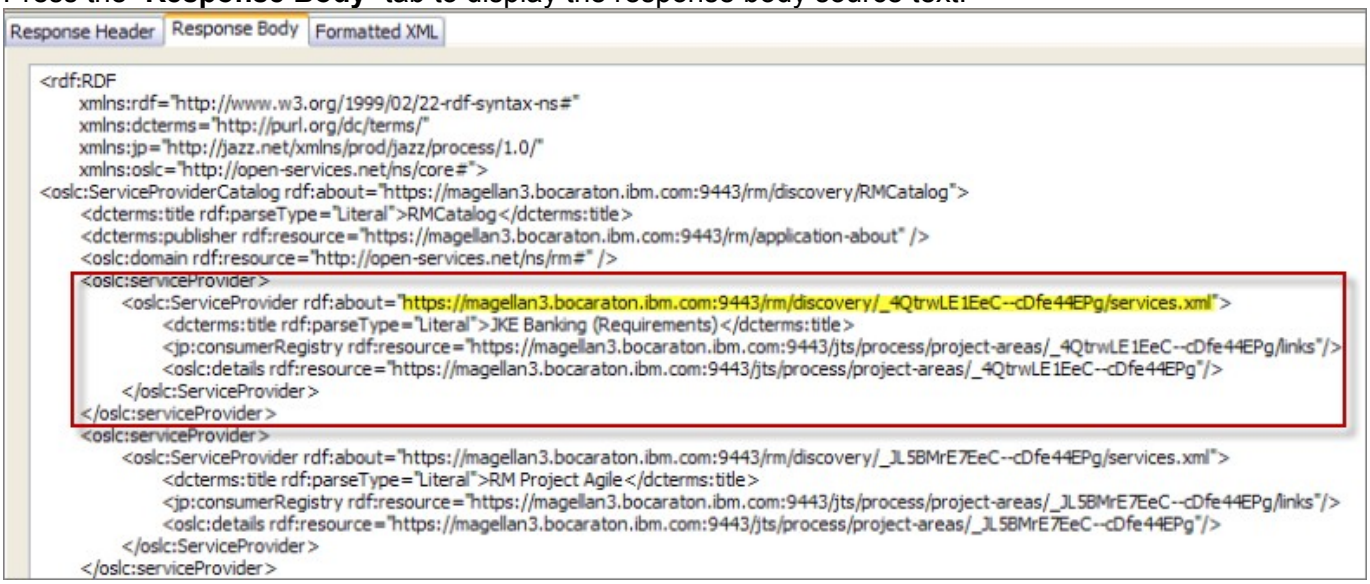
__4. Press the "**Formatted XML**" tab to display the response body.



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

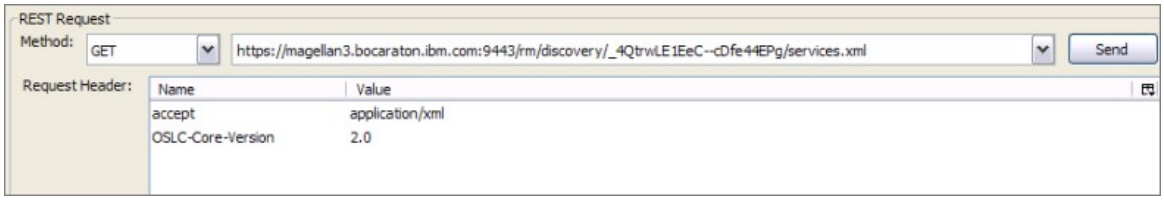
In the case of RRC, there is one ServiceProvider element for each **RM Project**.
Typically, an application would use the title of this element to allow the user to choose between the RM projects.

__5. Press the "**Response Body**" tab to display the response body source text.

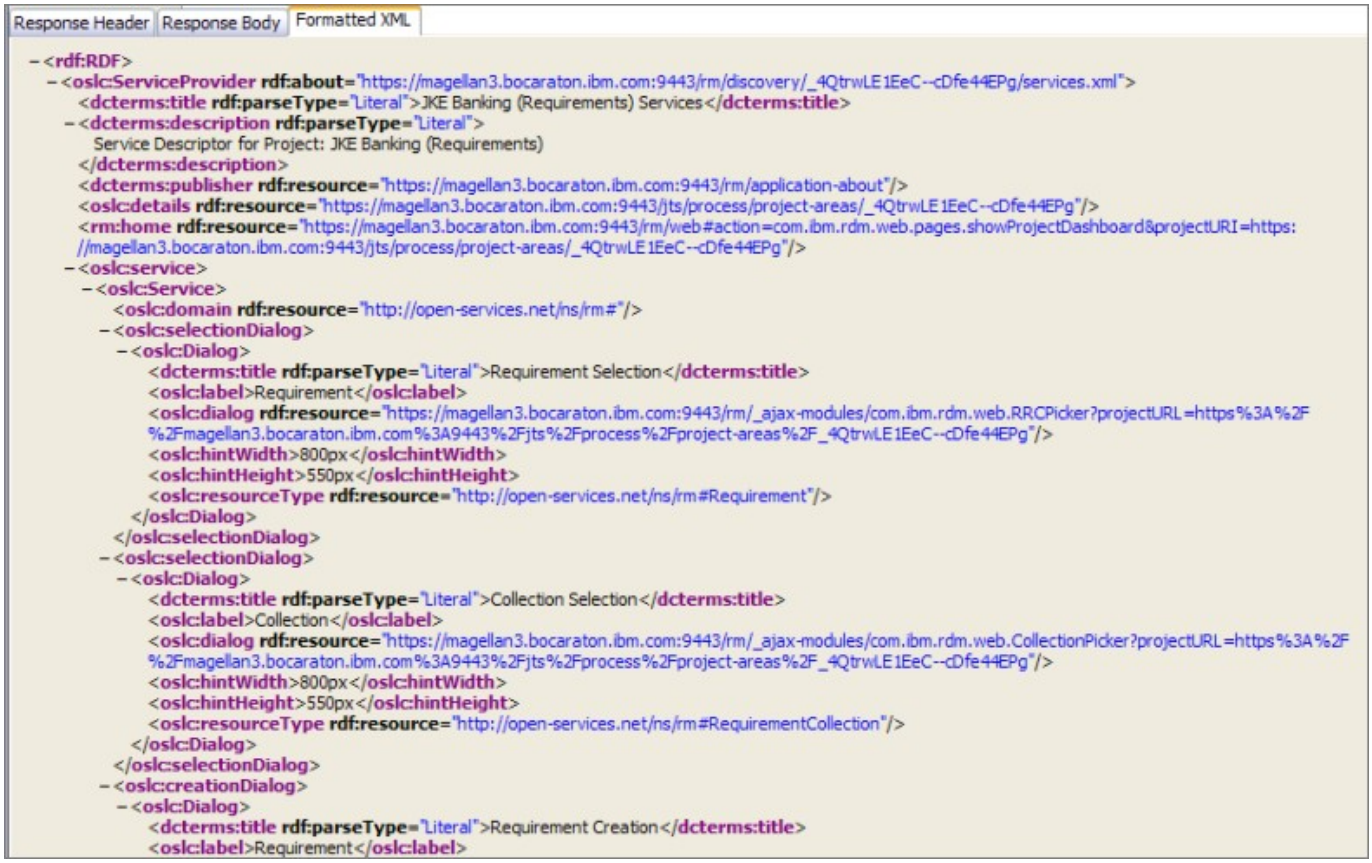


__6. From this view, you should be able to retrieve the Service Provider element for one of the RM Projects already created in the Jazz Team Server such as the **JKE Banking (Requirements)** Project and copy the URL associated to the attribute **rdf:about** defined in the element **oslc:ServiceProvider** .

__7. Paste the Service Provider URL in the URL field of the REST client and keep the headers.

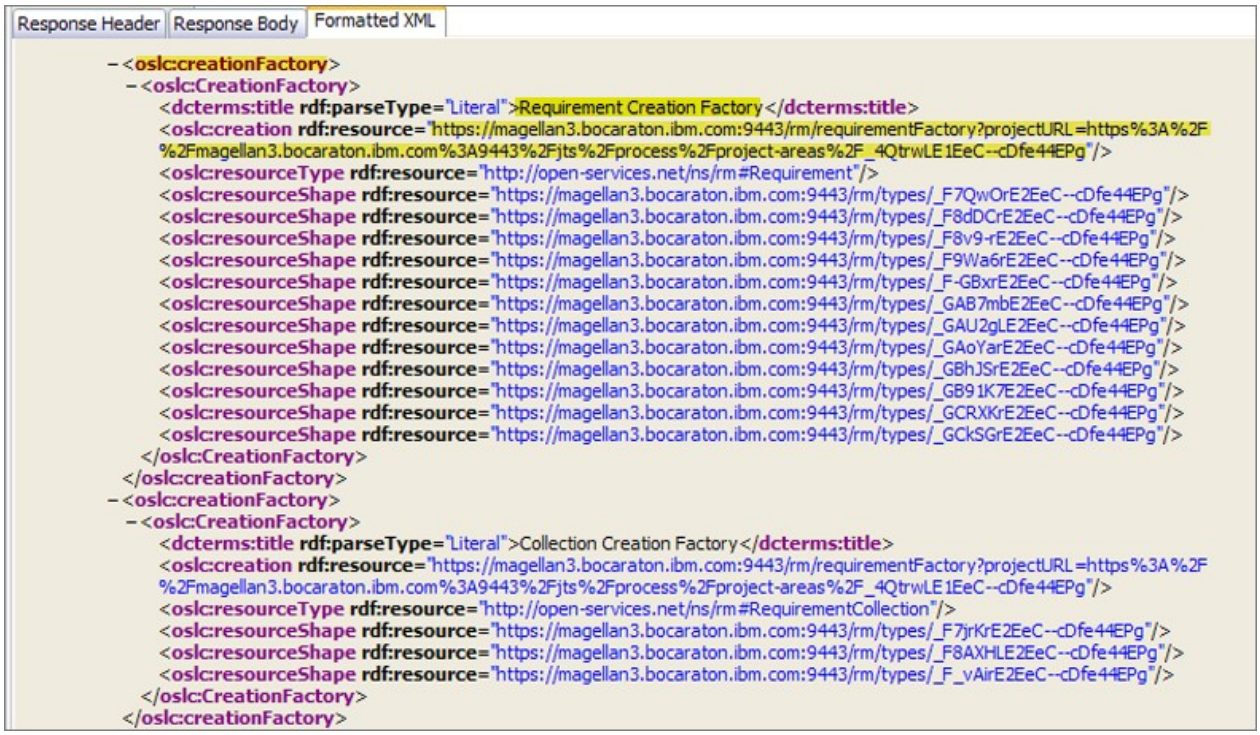


__8. Press the **Send** button. The response body will display a Service Provider document (http://open-services.net/bin/view/Main/OslcCoreSpecification#Service_Provider_Resources) listing all the REST services available for this Service Provider (alias RM Project):

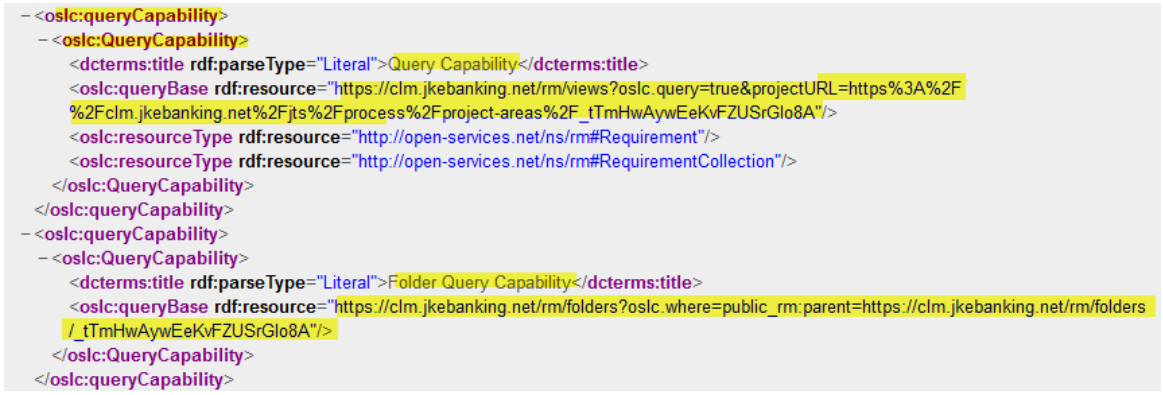


__9. You will find 4 kinds of services:

__a. **Creation Factory**: Enables clients to create new resources:



__b. **Query Capability:** Enables clients to query across a collection of resources. **Note:** For RRC v4.0 and above, there is a new capability for query of folders.



__c. **Selection Dialog**: Enables clients to select a resource via UI

```
- <oslc:selectionDialog>
  - <oslc:Dialog>
      <dcterms:title rdf:parseType="Literal">Requirement Selection</dcterms:title>
      <oslc:label>Requirement</oslc:label>
      <oslc:dialog rdf:resource="https://magellan3.bocaraton.ibm.com:9443/rm/_ajax-modules/com.ibm.rdm.web.RRCPicker?projectURL=https%3A%2F
      %2Fmagellan3.bocaraton.ibm.com%3A9443%2Fjts%2Fprocess%2Fproject-areas%2F_4QtrwLE1EeC--cDfe44EPg"/>
      <oslc:hintWidth>800px</oslc:hintWidth>
      <oslc:hintHeight>550px</oslc:hintHeight>
      <oslc:resourceType rdf:resource="http://open-services.net/ns/rm#Requirement"/>
    </oslc:Dialog>
  </oslc:selectionDialog>
- <oslc:selectionDialog>
  - <oslc:Dialog>
      <dcterms:title rdf:parseType="Literal">Collection Selection</dcterms:title>
      <oslc:label>Collection</oslc:label>
      <oslc:dialog rdf:resource="https://magellan3.bocaraton.ibm.com:9443/rm/_ajax-modules/com.ibm.rdm.web.CollectionPicker?projectURL=https%3A%2F
      %2Fmagellan3.bocaraton.ibm.com%3A9443%2Fjts%2Fprocess%2Fproject-areas%2F_4QtrwLE1EeC--cDfe44EPg"/>
      <oslc:hintWidth>800px</oslc:hintWidth>
      <oslc:hintHeight>550px</oslc:hintHeight>
      <oslc:resourceType rdf:resource="http://open-services.net/ns/rm#RequirementCollection"/>
    </oslc:Dialog>
  </oslc:selectionDialog>
```

__d. **Creation Dialog**: Enables clients to create a resource via UI

```
- <oslc:creationDialog>
  - <oslc:Dialog>
      <dcterms:title rdf:parseType="Literal">Requirement Creation</dcterms:title>
      <oslc:label>Requirement</oslc:label>
      <oslc:dialog rdf:resource="https://magellan3.bocaraton.ibm.com:9443/rm/_ajax-modules/com.ibm.rdm.web.CreateArtifactPicker?projectURL=https
      %3A%2F%2Fmagellan3.bocaraton.ibm.com%3A9443%2Fjts%2Fprocess%2Fproject-areas%2F_4QtrwLE1EeC--cDfe44EPg"/>
      <oslc:hintWidth>520px</oslc:hintWidth>
      <oslc:hintHeight>425px</oslc:hintHeight>
      <oslc:resourceType rdf:resource="http://open-services.net/ns/rm#Requirement"/>
    </oslc:Dialog>
  </oslc:creationDialog>
- <oslc:creationDialog>
  - <oslc:Dialog>
      <dcterms:title rdf:parseType="Literal">Collection Creation</dcterms:title>
      <oslc:label>Collection</oslc:label>
      <oslc:dialog rdf:resource="https://magellan3.bocaraton.ibm.com:9443/rm/_ajax-modules/com.ibm.rdm.web.CreateCollectionPicker?projectURL=https
      %3A%2F%2Fmagellan3.bocaraton.ibm.com%3A9443%2Fjts%2Fprocess%2Fproject-areas%2F_4QtrwLE1EeC--cDfe44EPg"/>
      <oslc:hintWidth>520px</oslc:hintWidth>
      <oslc:hintHeight>425px</oslc:hintHeight>
      <oslc:resourceType rdf:resource="http://open-services.net/ns/rm#RequirementCollection"/>
    </oslc:Dialog>
  </oslc:creationDialog>
```

5.3 Search for Requirements

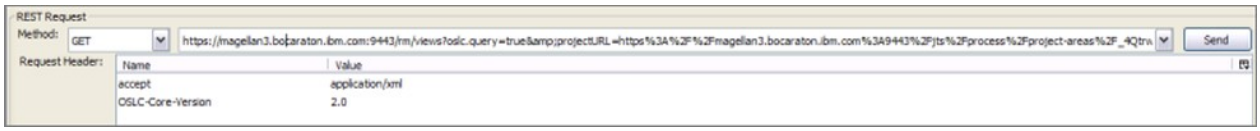
In this section, we will describe how we can query for requirements using the corresponding OSLC-RM REST service.

__1. Scroll back to the **oslc:QueryCapability** element in the Service Provider services document.

__2. From this element, look for the **oslc:queryBase** element and copy the URL associated to the attribute **rdf:resource** defined.

```
-<oslc:queryCapability>
  -<oslc:QueryCapability>
      <dcterms:title rdf:parseType="Literal">Query Capability</dcterms:title>
      <oslc:queryBase rdf:resource="https://magellan3.bocaraton.ibm.com:9443/rm/views?oslc.query=true&projectURL=https%3A%2F
      %2Fmagellan3.bocaraton.ibm.com%3A9443%2Fjts%2Fprocess%2Fproject-areas%2F_4QtrwLE1EeC--cDfe44EPg"/>
      <oslc:resourceType rdf:resource="http://open-services.net/ns/rm#Requirement"/>
      <oslc:resourceType rdf:resource="http://open-services.net/ns/rm#RequirementCollection"/>
    </oslc:QueryCapability>
  </oslc:queryCapability>
```

__3. Paste this URL into in the URL field of the REST client and keep the headers.



__4. Press the **Send** button, the response body will display **ALL** the artifacts (**rdfs:member** elements) listed for the corresponding project.

__5. Locate the **dcterms:title** element. It indicates the number of Query Results.

This REST service supports the following parameters (http://open-services.net/bin/view/Main/OSLCCoreSpecQuery): **oslc.searchTerms**, **oslc.where, oslc.select, oslc.properties and oslc.prefix.**

These parameters offer the possibility to filter the artifacts you are looking for and which data to fetch.

__6. Lets try to retrieve all the requirements which contain the word “donor” in it.
To do so, complete the Query URL with the following parameter:

```
&oslc.searchTerms="donor"
```

So the URL should look like this:

```
https://jazz.server.com:9443/rm/views?
oslc.query=true&amp;amp;projectURL=https%3A%2F
%2Fmagellan3.bocaraton.ibm.com%3A9443%2Fjts%2Fprocess%2Fproject-areas
%2F_4QtrwLE1EeC--cDfe44EPg&oslc.searchTerms="donor"
```

__7. Press the **Send** button. the response body will display a subset of these requirements: The dcterms:title element displays how many results were found.

__8. If we complete the previous request with the **oslc.select** parameter, we will be able to specify the subset of attributes/elements we want to fetch from the server.

For example, let's say you are only interested in the resource name (**dcterms:title**). In this case, complete the previous URL with the following expression:

```
&oslc.prefix=dcterms=<http://purl.org/dc/terms/>&oslc.select=dcterms:title
```

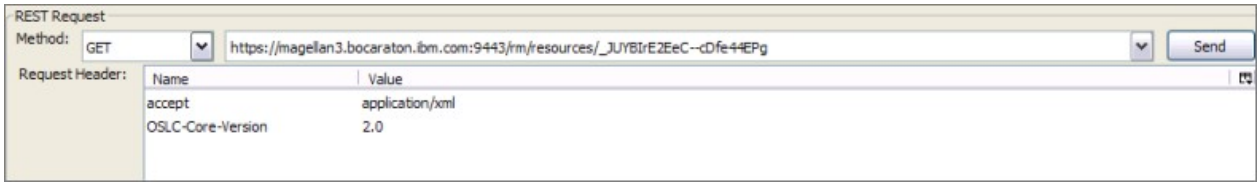
Note how we had to define the namespace context with **oslc.prefix** prior to being able to use **dcterms.title** in the query.

So the URL should look like this:

```
https://magellan3.bocaraton.ibm.com:9443/rm/views?
oslc.query=true&projectURL=https%3A%2F%2Fmagellan3.bocaraton.ibm.com
%3A9443%2Fjts%2Fprocess%2Fproject-areas%2F_4QtrwLE1EeC--
cDfe44EPg&oslc.searchTerms="donor"&oslc.prefix=dcterms=<http://purl.org
/dc/terms/>&oslc.select=dcterms:title
```

__9. Press the **Send** button. The REST client will only display the requested attributes:

__10. Now, we want to demonstrate how to retrieve all the attributes of a requirement. Copy the URL of a **rdfs:member** previously listed and paste it in the URL field:

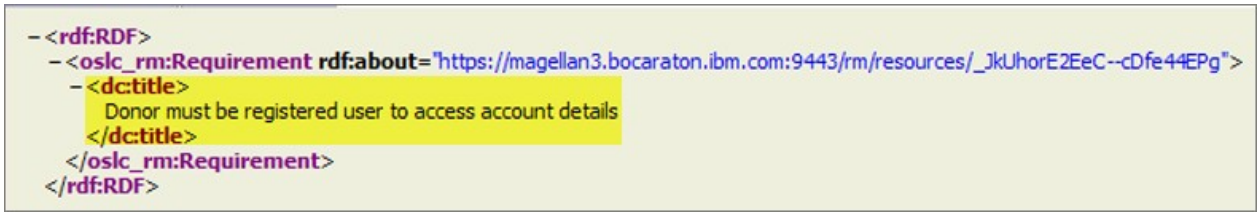


__11. And press **Send**, the **Formatted XML** tab should display all the attributes of the requirement

__12. When retrieving a resource,it is possible to do a partial retrieval using a HTTP URL key=value pair with **oslc.properties.** Let's try an example using the same URL from the previous step. Append the following to the URL:

```
?oslc.prefix=dcterms=<http://purl.org/dc/terms/>

&oslc.properties=dcterms:title
```

__13. Press **Send,** and the **Formatted XML** tab should display only the title of the requirement.



5.4 Requirement Selection Dialog

In this section, we will describe how we can invoke the selection dialog for requirement artifacts using the corresponding OSLC-RM REST service.

__1. Scroll back to the **oslc:selectionDialog** element in the Service Provider services document.

__2. From this element, look for the **oslc:dialog** element and copy the URL associated to the attribute **rdf:resource** defined.

__3. Paste this URL into in a browser and append the following to the URL: `#oslc-core-windowName-1.0`

__4. The following dialog will be displayed which is the Requirement Selection dialog.

5.5 Creation Dialog

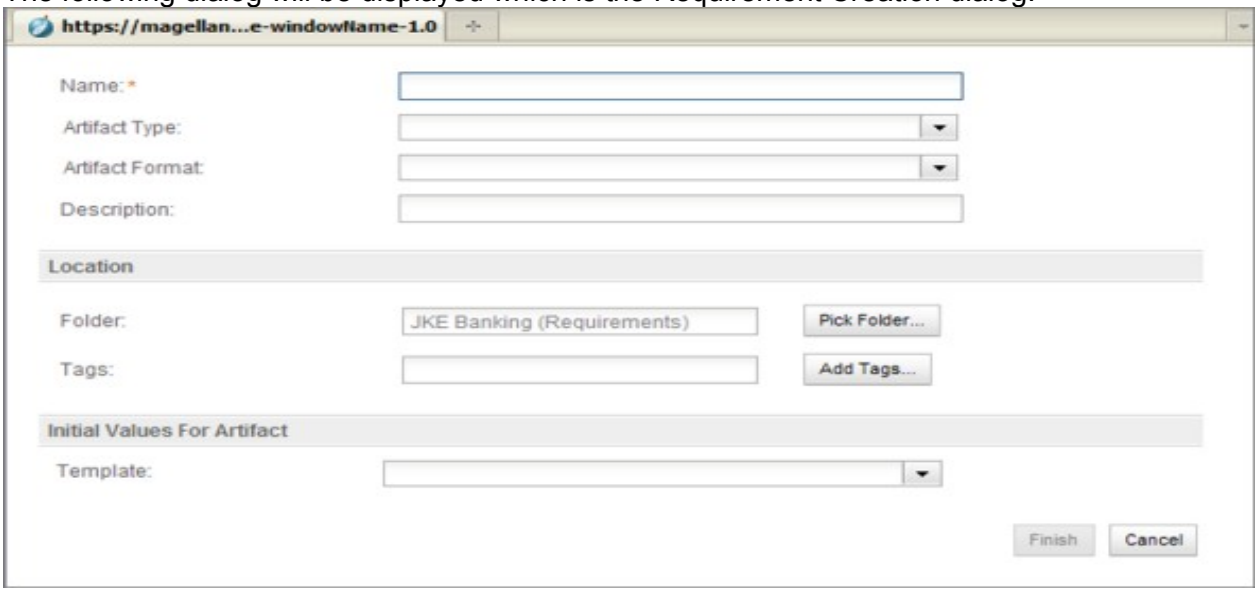
In this section, we will describe how we can invoke the creation dialog for requirements using the corresponding OSLC-RM REST service.

__11. Scroll back to the **oslc:creationDialog** element in the Service Provider services document.

__12. From this element, look for the **oslc:dialog** element and copy the URL associated to the attribute **rdf:resource** defined.

__13. Paste this URL into in a browser and append the following to the URL: `#oslc-core-windowName-1.0`

__14. The following dialog will be displayed which is the Requirement Creation dialog.



OSLC Traceability

Note how these last 2 dialogs look familiar. They are the same dialogs that are used by Rational Team Concert when it plays the role of an OSLC consumer. RTC presents these dialogs to a user when it is either 1) searching for existing or 2) creating a new RM resource for the purpose of establishing a traceability link between a workitem and that RM resource.

5.6 Appendix

Here are some more examples related to query function. As we learned in an earlier section, you can filter the artifacts you are looking for and which data to fetch.

Append the following to the QueryCapability URI to further refine a query. **Note**: The special character # must be URL encoded as %23.

Find artifact by ID, display all fields

&oslc.prefix=dcterms=<http://purl.org/dc/terms/>&oslc.select=*&oslc.where=dcterms:identifier=4

Find artifact by title, using fulltext search and display all fields

&oslc.prefix=dcterms=<http://purl.org/dc/terms/>&oslc.select=*&oslc.searchTerm=”donor”

Query for all requirements in a specific project

&oslc.prefix=rdf=<http://www.w3.org/1999/02/22-rdf-syntax-ns%23>,dcterms=<http://purl.org/dc/terms/>&oslc.select=*&oslc.where=rdf:type=<http://open-services.net/ns/rm%23Requirement>

Query for all artifacts that were modified at or later than 08/01/2012 at 21:50:40.979. This must specify the date in the following format : ^^xsd:dateTime

&oslc.prefix=dcterms=<http://purl.org/dc/terms/>,oslc_rm=<http://open-services.net/ns/rm%23>&oslc.select=dcterms:title,dcterms:modified&oslc.where=dcterms:modified>="2012-08-01T21:51:40.979Z"%5E%5Exsd%3AdateTime

Conclusion

You have completed lab 5. You now have an initial understanding of the OSLC RM APIs.

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

Lab 6 Access OSLC RM APIs programmatically

Lab Scenario

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

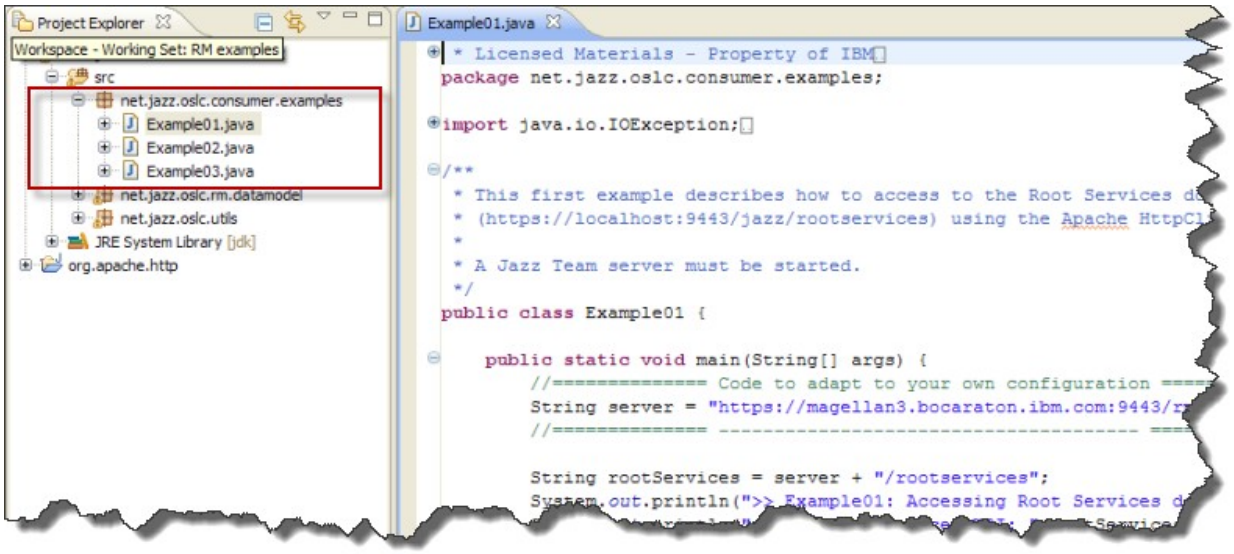
If your Jazz Team Server is not running, start it now (<JazzTeamServer_Root_Folder>`\server\server.startup.bat`).

6.1 Accessing 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. If not already running, start the RTC Eclipse client (<TeamConcert_Root_Folder> \eclipse.exe). When prompted, select the Eclipse workspace used previously in Lab 2 such as **C:\RTC30Dev\DevWS**.

__2. In the **Package Explorer** view, expand the **src/net.jazz.oslc.consumer.examples** source package of the **net.jazz.oslc.consumer.rm.client** Eclipse project and then double click the **Example01.java** file.



__3. First change the RRC Server URL, if necessary, by changing the value of the **String server** variable. For example, if your RM server is accessed as **https://jazz.server.com:9443/rm** then the initialization of the **server** variable would be

```
String server = "https://jazz.server.com:9443/rm";
```

__4. Make sure you save the file (**Ctrl-S**).

__5. The following snippet of code is extracted from the `main` method.

```
// Setup the HttClient
HttpClient httpclient = new DefaultHttpClient();
// Disabling SSL Certificate Validation
HttpUtils.setupLazySSLSupport(httpclient);
// Setup the HTTP GET method
HttpGet rootServiceDoc = new HttpGet(rootServices);
rootServiceDoc.addHeader("Accept", "application/rdf+xml");
rootServiceDoc.addHeader("OSLC-Core-Version", "2.0");

HttpResponse response;
try {
    // Execute the request
    response = httpclient.execute(rootServiceDoc);
    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();
}
```

__6. 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 HttClient
HttpClient httpclient = new DefaultHttpClient();
```

__7. 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.

__8. 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 RM Root Services document. The request is completed with the header specifying the expected media type result and expected format base of the OSLC-RM Release 2.0 specifications.

```
// Setup the HTTP GET method
HttpGet rootServiceDoc = new HttpGet("https://jazz.server.com:9443/rm");
rootServiceDoc.addHeader("Accept", "application/rdf+xml");
rootServiceDoc.addHeader("OSLC-Core-Version", "2.0");
```

__9. 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(rootServiceDoc);
```

__10. The next line prints out the **status code** of the HTTP response

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

__11. 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);
```

__12. If the response is an error then the code releases any created resources:

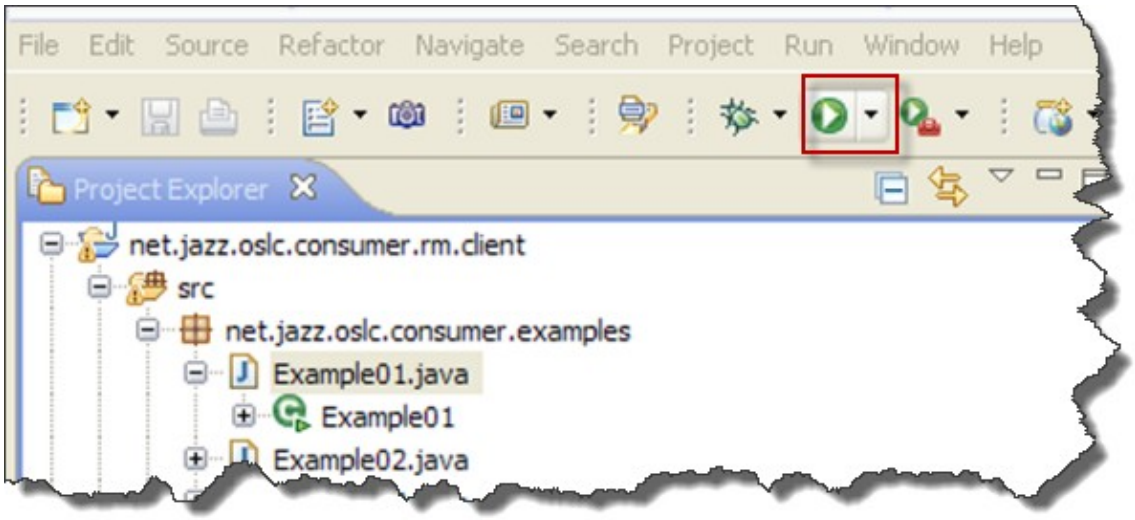
```
// Release allocated resources
response.getEntity().consumeContent();
```

__13. Finally, the last line shuts down the HTTP client by releasing the connections.

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

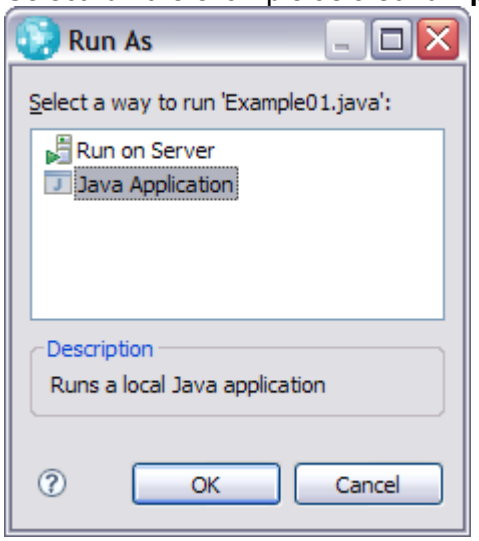
__14. Now that we have a good understanding of the code, lets run it.

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

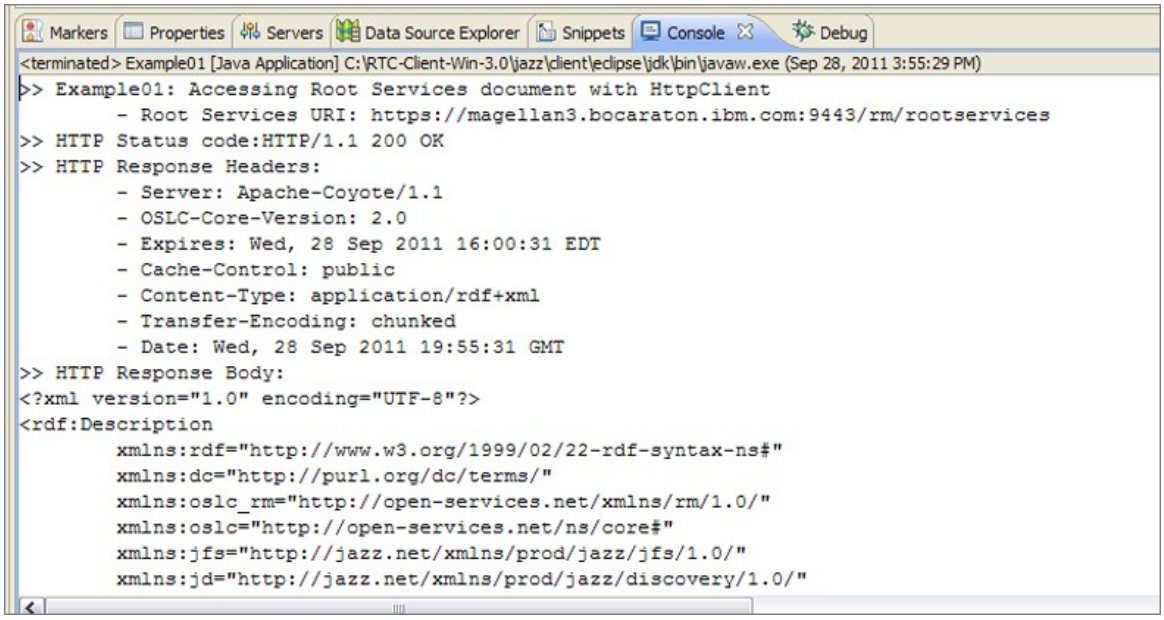


__b. Press the **Run as…** button located on toolbar

__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.



The log should look like this:

```
>> Example01: Accessing Root Services document with HttpClient
    - Root Services URI: https://magellan3.bocaraton.ibm.com:9443/rm/rootservices
>> HTTP Status code:HTTP/1.1 200 OK
>> HTTP Response Headers:
    - Server: Apache-Coyote/1.1
    - OSLC-Core-Version: 2.0
    - Expires: Wed, 28 Sep 2011 15:08:48 EDT
    - Cache-Control: public
    - Content-Type: application/rdf+xml
    - Transfer-Encoding: chunked
    - Date: Wed, 28 Sep 2011 19:03:48 GMT
>> HTTP Response Body:
<?xml version="1.0" encoding="UTF-8"?>
<rdf:Description
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/terms/"
    xmlns:oslc_rm="http://open-services.net/xmlns/rm/1.0/"
    xmlns:oslc="http://open-services.net/ns/core#"
    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/"
    xmlns:jdb="http://jazz.net/xmlns/prod/jazz/dashboard/1.0/"
    xmlns:ju="http://jazz.net/ns/ui#"
    xmlns:rm="http://www.ibm.com/xmlns/rdm/rdf/"
    xmlns:fp="http://jazz.net/xmlns/prod/jazz/frontingapp/process/1.0/"
    xmlns:jp="http://jazz.net/xmlns/prod/jazz/presentation/1.0/"
```

```
    xmlns:jp10="http://jazz.net/xmlns/prod/jazz/process/1.0/"
    rdf:about="https://magellan3.bocaraton.ibm.com:9443/rm/rootservices">
    <dc:title>Requirements Management</dc:title>
    <dc:description>The Requirements Management application implements the services and web interfaces for
Requirements Management.  In some cases only a subset of these capabilities may be available to you depending
on the particular product license key(s) you have been assigned by the server administrator.</dc:description>
    <oslc_rm:majorVersion>3</oslc_rm:majorVersion>
    <oslc_rm:version>3.0.1.0</oslc_rm:version>
    <oslc_rm:buildVersion>3.0.1 (I20110602_0919)

…/...

</rdf:Description>
```

6.2 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_rm:cmServiceProviders`.

The W3C XPath language (__) 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
foo	**Selects all the child nodes named foo.**
/foo	**Selects from the root node the nodes named foo.**
//foo	**Selects nodes named foo no matter where they are in the document**
@att	**Selects the attribute node named att.**
foo/bar	**Selects all the nodes named bar having a parent node named foo.**
//foo[@att]	**Select all the nodes named foo no matter where they are having an attribute named att.**
//foo [@att="val”]	**Select all the nodes named foo no matter where they are having an attribute named att with the value val.**

XPath tester

if you need to test an XPath expression against some XML code, there are several interesting testers on the web.

- We found a first one which requires to upload the XML file to parse: http://www.whitebeam.org/library/guide/TechNotes/xpathtestbed.rhtm
- We found another one which accepts a copy/paste of the XML content to parse: http://www.futurelab.ch/xmlkurs/xpath.en.html

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

```
<?xml version="1.0"?>
<rdf:Description …>
…/…

<!-- Catalog of Requirements Management Service Providers on this server -->
    <oslc_rm:rmServiceProviders rdf:resource="https://jazz.server.com:9443/rm/discovery/RMCatalog" />
…/…
    </rdf:Description>
```

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

```
/rdf:Description/oslc_rm:rmServiceProviders/@rdf:resource
```

This expression means: "Select the attribute node named `rdf:resource` from the element node named `oslc_rm:rmServiceProviders`, child of the root 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
XPathFactory factory = XPathFactory.newInstance();
XPath xpath = factory.newXPath();
xpath.setNamespaceContext(
            new NamespaceContextMap(new String[]
                    {    "rdf", "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
                         "oslc_rm","http://open-services.net/xmlns/rm/1.0/"}));

// Parse the response body to retrieve the catalog URI
InputSource source = new InputSource(response.getEntity().getContent());
Node attribute = (Node) (xpath.evaluate("/rdf:Description/oslc_rm:rmServiceProviders/@rdf:resource",
                                        source,   XPathConstants.NODE));

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

__2. The first lines create an instance of an XPath evaluation environment. This environment is set up to be able to parse and understand nodes using the rdf and oslc-rm namespaces.

```
XPathFactory factory = XPathFactory.newInstance();
XPath xpath = factory.newXPath();
```

__3. The next instruction sets the namespace context. This interface is used to retrieve the namespaces corresponding to the prefixes used by the XPath. In this example, the XPath is `/rdf:Description/oslc_rm:rmServiceProviders/@rdf:resource`. It references two prefixes "**rdf**" and "**oslc_rm**". So, the NamespaceContext defines the mapping between these prefixes and the namespaces (xlmns declarations) used in the document that the XPath will parse.

```
xpath.setNamespaceContext(
        new NamespaceContextMap(new String[]
                { "rdf",        "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
                  "oslc_rm",    "http://open-services.net/xmlns/rm/1.0/"}));
```

__4. The next lines parse the response body (response.getEntity().getContent()) using the `XPath.evaluate` method. This method takes 3 arguments:

```
// Parse the response body
InputSource source = new InputSource(response.getEntity().getContent());
Node attribute = (Node) (xpath.evaluate(
                        "/rdf:Description/oslc_rm:rmServiceProviders/@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_rm:rmServiceProviders/@rdf:resource"
```

__b. The source to parse.

This source can be either an `org.xml.sax.InputSource` 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. 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.

__5. 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());
```

__6. Now that we have a good understanding of the code, lets run it.

__a. Double-click the **Example02.java** file in the **Package Explorer**.

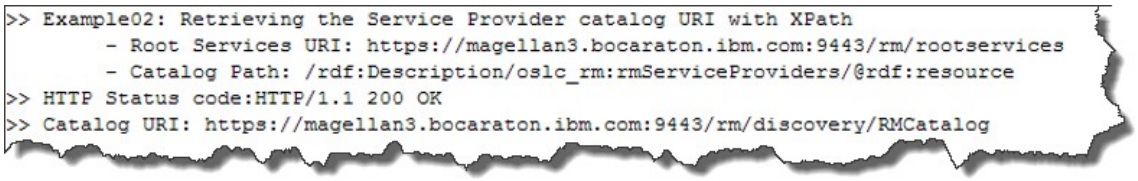
__b. First change the RRC Server URL if necessary by changing the value of the String **server** variable. For example, if your RRC server is accessed as https://jazz.server.com:9443/rm then the initialization of the **server** variable would be

```
String server = "https://jazz.server.com:9443/rm";
```

__c. Make sure you save the file (**Ctrl-S)**.

__d. Run the sample as a Java application

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



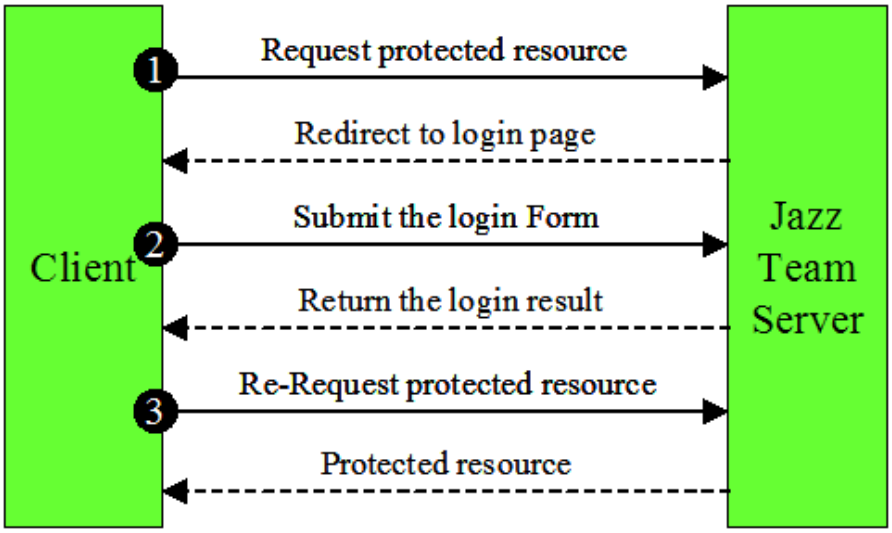
6.3 Jazz Form-based Authentication

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` and `doRRCOAuth` method stored in the **HttpUtils** class. If you don't want to dig into this implementation, feel free to directly skip to step (30).

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

```
        static String AUTHURL = "X-jazz-web-oauth-url";
        // Step (1): Request the protected resource
        HttpResponse documentResponse = httpClient.execute(request);

        if (documentResponse.getStatusLine().getStatusCode() == 200 ||
    documentResponse.getStatusLine().getStatusCode() == 401) {
            Header header = documentResponse.getFirstHeader(AUTHURL);

            if ((header!=null)  {
                documentResponse.getEntity().consumeContent();
                //First GET
                HttpGet request2 = new HttpGet(header.getValue());
                HttpClientParams.setRedirecting(request2.getParams(), false);
                documentResponse = httpClient.execute(request2);
                documentResponse.getEntity().consumeContent();

                //Second GET
                Header location = documentResponse.getFirstHeader("Location");
                HttpGet request3 = new HttpGet(location.getValue());
                HttpClientParams.setRedirecting(request3.getParams(), false);
                documentResponse = httpClient.execute(request3);
                documentResponse.getEntity().consumeContent();

                //POST to login form
                // The server requires an authentication: Create the login form
                HttpPost formPost = new HttpPost(jtsURI+"/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);
                formResponse.getEntity().consumeContent();

                //Third GET
                HttpGet request4 = new HttpGet(location.getValue());
                HttpClientParams.setRedirecting(request4.getParams(), false);
                documentResponse = httpClient.execute(request4);
                documentResponse.getEntity().consumeContent();

                //Second Post
                Header location3 = documentResponse.getFirstHeader("Location");
                Map<String,String> oAuthMap = getQueryMap(location3.getValue());
                String oauthToken = oAuthMap.get("oauth_token");
                String oauthCallback = oAuthMap.get("oauth_callback");;
```

```
                // The server requires an authentication: Create the login form
                HttpPost formPost2 = new HttpPost(jtsURI+"/j_security_check");
                formPost2.getParams().setParameter("oauth_token", oauthToken);
                formPost2.getParams().setParameter("oauth_callback", oauthCallback);
                formPost2.getParams().setParameter("authorize", "true");
                formPost2.addHeader("Content-Type","application/x-www-form-urlencoded;charset=UTF-8");

                formResponse = httpClient.execute(formPost2);
                formResponse.getEntity().consumeContent();

                header = formResponse.getFirstHeader("Content-Length");
                if ((header!=null) && (!("0".equals(header.getValue())))) {
                    // The login failed
                    throw new InvalidCredentialsException("Authentication failed");
                } else {
                    // The login succeed
                    // Step (3): Request again the protected resource
                    formResponse.getEntity().consumeContent();
                    return true; //REDO YOUR REQUEST
                }
            }
        }
        return  false;
```

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

```
// 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-jazz-web-oauth-url* HTTP response header. If the value of this header is not NULL 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 ||
    documentResponse.getStatusLine().getStatusCode() == 401)  {
    Header header = documentResponse.getFirstHeader("X-jazz-web-oauth-url");

    if ((header!=null) {
        …
    } else  {
        return documentResponse;
    }
```

__7. The next step consists of retrieving some information via 2 GET calls and then filling in and POSTing the authentication form:

```
        //First GET
        HttpGet request2 = new HttpGet(header.getValue());
        HttpClientParams.setRedirecting(request2.getParams(), false);
        documentResponse = httpClient.execute(request2);
        documentResponse.getEntity().consumeContent();

        //Second GET
        Header location = documentResponse.getFirstHeader("Location");
        HttpGet request3 = new HttpGet(location.getValue());
        HttpClientParams.setRedirecting(request3.getParams(), false);
        documentResponse = httpClient.execute(request3);
        documentResponse.getEntity().consumeContent();
        //POST to login form
        // The server requires an authentication: Create the login form
        HttpPost formPost = new HttpPost(jtsURI+"/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 retrieve some information from thefirst POST's response header:location URI. Then, do a GET on this location URI to retrieve the following: oauth_token, oauth_callback. Using this token and the callback, prepare a 2nd POST to the login form.

```
        //Third GET
        HttpGet request4 = new HttpGet(location.getValue());
        HttpClientParams.setRedirecting(request4.getParams(), false);
        documentResponse = httpClient.execute(request4);
        documentResponse.getEntity().consumeContent();
```

__9. Send the second POST to the location URI retrieved from the last GET along with the oAuth token and the oAuthCallback as parameters.

```
//Second Post
Header location3 = documentResponse.getFirstHeader("Location");
Map<String,String> oAuthMap = getQueryMap(location3.getValue());
String oauthToken = oAuthMap.get("oauth_token");
String oauthCallback = oAuthMap.get("oauth_callback");;

// The server requires an authentication: Create the login form
HttpPost formPost2 = new HttpPost(jtsURI+"/j_security_check");
formPost2.getParams().setParameter("oauth_token", oauthToken);
formPost2.getParams().setParameter("oauth_callback", oauthCallback);
formPost2.getParams().setParameter("authorize", "true");
formPost2.addHeader("Content-Type","application/x-www-form-urlencoded;charset=UTF-8");
formResponse = httpClient.execute(formPost2);
formResponse.getEntity().consumeContent();
```

__10. If the login didn't fail, then the client can request the protected document a second time, and should receive the expected response:

```
header = formResponse.getFirstHeader("Content-Length");
if ((header!=null) && (!("0".equals(header.getValue())))) {
    // The login failed
    throw new InvalidCredentialsException("Authentication failed");
} else {
    // The login succeed
    // Step (3): Request again the protected resource
    formResponse.getEntity().consumeContent();
    return true; //REDO YOUR REQUEST
}
```

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

__11. In the **Package Explorer** view, open the **Example03.java** file and look at the `main` method:

```
// Setup the catalog request
HttpGet catalogDoc = new HttpGet(serviceProvidersCatalog);
catalogDoc.addHeader("Accept", "application/xml");
catalogDoc.addHeader("OSLC-Core-Version", "2.0");

// Access to the Service Providers catalog
HttpResponse catalogResponse
        = HttpUtils.sendGetForSecureDocument(server, catalogDoc, login, password, httpclient);
if (catalogResponse.getStatusLine().getStatusCode() == 200) {
    // Define the XPath evaluation environment
    XPath xpath2 = factory.newXPath();
    xpath2.setNamespaceContext(
                new NamespaceContextMap(new String[]
                        {       "oslc",         "http://open-services.net/ns/core#",
                                "dcterms",      "http://purl.org/dc/terms/"}));

    // Parse the response body to retrieve the Service Provider
    source = new InputSource(catalogResponse.getEntity().getContent());
    NodeList titleNodes = (NodeList) (xpath2.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/xml`.

```
// Access to the Service Providers catalog
HttpResponse catalogResponse
        = HttpUtils.sendGetForSecureDocument(server, catalogDoc, 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:

```
// Define the XPath evaluation environment
XPath xpath2 = factory.newXPath();
xpath2.setNamespaceContext(
            new NamespaceContextMap(new String[]
                  {     "oslc",          "http://open-services.net/ns/core#",
                        "dcterms",       "http://purl.org/dc/terms/"}));

// Parse the response body to retrieve the Service Provider
source = new InputSource(catalogResponse.getEntity().getContent());
NodeList titleNodes
    = (NodeList) (xpath2.evaluate(
                        "//oslc:ServiceProvider/dcterms: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"?>
<rdf:RDF xmlns:dcterms="http://purl.org/dc/terms/" xmlns:ns1="http://jazz.net/xmlns/prod/jazz/process/1.0/"
   xmlns:oslc="http://open-services.net/ns/core#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <oslc:ServiceProviderCatalog rdf:about="https://jazz.server.com:9443/ccm/oslc/workitems/catalog">
      <dcterms:title rdf:parseType="Literal">Project Areas</dcterms:title>
      <dcterms:publisher>
         .../...
      </dcterms:publisher>
      <oslc:domain rdf:resource="http://open-services.net/ns/cm#"/>
      <oslc:serviceProvider>
         <oslc:ServiceProvider rdf:about= "https://jazz.server.com:9443/ccm/oslc/.../services.xml">
            <dcterms:title rdf:parseType="Literal">JUnit Project</dcterms:title>
            <oslc:details
               rdf:resource="https://jazz.server.com:9443/ccm/process/..."/>
            <ns1:consumerRegistry rdf:resource="https://jazz.server.com:9443/rm/process/.../links"/>
         </oslc:ServiceProvider>
      </oslc:serviceProvider>
   </oslc:ServiceProviderCatalog>
</rdf:RDF>
```

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

```
"/rdf:RDF/oslc:ServiceProviderCatalog/oslc:serviceProvider/oslc:ServiceProvider/dcterms:title"
```

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

```
"//oslc:ServiceProvider/dcterms:title"
```

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

```
//dcterms:title
```

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

__12. Now that we have a good understanding of the code, lets run it.

__a. Select the **Example03.java** file in the **Package Explorer**

__b. *If necessary* first change the values of the following variables to match your setup:

```
String server = https://jazz.server.com:9443/rm;
String login = "clmadmin";
String password = "clmadmin";
```

__c. Make sure you save the file (**Ctrl-S)**.

__d. Run it as Java application.

The Console view will print out the titles of the RM Projects currently stored in the Jazz Team Server:.

```
>> Example03: Print out the content of the Service Providers catalog
        - Root Services URI: https://magellan3.bocaraton.ibm.com:9443/rm/rootservices
        - Service Providers catalog XPath expression: /rdf:Description/oslc_rm:rmServiceProviders/@rdf:resource
        - Service Provider title XPath expression: //oslc:ServiceProvider/dcterms:title
        - Login: clmadmin
        - Password: clmadmin
Sep 28, 2011 4:32:18 PM org.apache.http.impl.client.DefaultRequestDirector handleResponse
WARNING: Authentication error: Unable to respond to any of these challenges: {oauth=WWW-Authenticate: OAuth realm="J
>> Project Areas:
>>       - JKE Banking (Requirements)
>>       - RM Project Agile
>>       - RM Project Usecase Driven Reqs
>>       - RM Project - Traditional
```

6.4 Requirement Create/Modify

In this example, we will describe how to create a requirement resource using the one of the REST services. The REST service we should use is called CreationFactory. Remember that we introduced these REST services in Lab 5.

NEW for RRC 4.0 – Folder support was added in this release. Example04 has been modified to use this new folder API if the sample is run against a 4.0 server.

Once the requirement resource is created, it will be modified and stored back on 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 fetch the creation factory, create a folder and create a requirement. **NOTE:** A new step has been added to check for the version of the server we are using to run the sample. If the server is 4.0, Step 3 creates a folder where the new requirement will be created.

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

// Step (2) : Retrieve the designated Service Provider (Project Area)
String serviceProviderURI = getServiceProvider(catalogURI, projectAreaName);
System.out.println(">> RM Project ["+projectAreaName+"]: "+projectAreaURI);

// Step (3) : (Optional) Create folders
// should only be used with RM version 4.0 or above
String baseFolder = null;
String sourceFolderUrl = null;
String targetFolderUrl = null;
if ( this.version >= 4.0f ) {
    basefolder = createFolder(serviceProviderURI, "TestBaseOSLCWS", null);
    sourceFolderUrl = createFolder(serviceProviderURI, "SourceFolder" baseFolder);
    targetFolderUrl = createFolder(serviceProviderURI, "TargetFolder", baseFolder);
}

// Step (4) : Find a creation factory and create requirement
String requirementURL = createRequirement(serviceProviderURI);
System.out.println(">> Requirement ["+requirementURL+"]");
```

__2. We will not spend time on the 2 first steps which have been explained during the previous labs. Step (4) is interesting because the code finds the creation factory and creates a requirement. Let's examine the createRequirement(...) method.

__a. First, we have to locate the creationFactory URL for the specific service provider or RM project we are interested in as the destination of this new artifact.

```
// Define the XPath evaluation environment
XPath xpath = getXpathNamespace();

//Resource Type
String resourceType = "http://open-services.net/ns/rm#Requirement";

//This is the URL we can use to post requirements to (note there could be more than one, we just pick the first
String requirementFactoryUrl ="//oslc:CreationFactory/oslc:resourceType[@rdf:resource=\"" + resourceType
+"\"]/../oslc:creation/@rdf:resource";

//This path tells us all the different types we can use to create our requirement
String requirementfactoryShapes = "//oslc:CreationFactory/oslc:resourceType[@rdf:resource=\"" +
resourceType +"\"]/../oslc:resourceShape/@rdf:resource";

// Retrieve the designated Service Provider
InputSource source = new InputSource(response.getEntity().getContent());
        Document doc = parse(source);
        response.getEntity().consumeContent();

        //Get the Creation Factory URL
        Node evaluate = (Node)xpath.evaluate(requirementFactoryUrl, doc, XPathConstants.NODE);
        String factoryURI = evaluate.getTextContent();
        System.out.println("Creation Factory URI->> " + evaluate.getTextContent());
```

__b. Then, get the available shapes. Shapes are analogous to Artifact Types in the RRC UI. Each RM project based on its template has a specific set of Artifact types. See Project Properties of a specific RM project to understand its defined artifact types. For this example, we are just going to choose the first one on the list. **Note**: It may vary depending on the RM project you are using in your workshop experience.

```
//Get the available shapes for this creation factory
NodeList paNode = (NodeList)(xpath.evaluate(requirementfactoryShapes, doc, XPathConstants.NODESET));
for (int i = 0; i < paNode.getLength(); i++) {
    //These all correspond to an artifact type URI in RRC
    System.out.println("Resource Shape URI->> " + paNode.item(i).getTextContent());
}
```

__c. Now, we pass the shape (or type of requirement) URL that we want to create to another method: **createRequirementContentFromShape(..)**

```
//For this example just use the first shape
String shapeURL = paNode.item(0).getTextContent();
```

```
//Create our content from the shape
final String content = createRequirementContentFromShape(shapeURL);
System.out.println("New Requirement Content ->>\n" + content);
```

__3. In the **createRequirementContentFromShape(..)** method, we add code that will build the requirement request which will be sent to the server.

__a. In this next block of code, we are deciding to include a title, description (which are both required) and an optional property, the primaryText. Primary Text is the 'rich' text that contains the body of a requirement document.

```
//For this example, lets assume we just want to have a title, description and
        //some basic content to add to primary text.
        String title = "MyDocument";
        String description = "This is a test document";
        //Note: primary text must be in xhtml compliant format
        String primaryText = "<div xmlns=\"http://www.w3.org/1999/xhtml\"
id=\"_Nf2cQJKNEd25PMUBGiN3Dw\"><h1 id=\"_DwpWsMueEd28xKN9fhQheA\">Test
Document</h1></div>";
```

__b. Next, we will use the RequirementRequest class to help us put all of this together in a XML document. It handles a subset of properties that are required by the OSLC specification and helps to generate the content for the creationFactory POST request.

__4. The new Requirement Content is returned from the **createRequirementContentFromShape** method, and we are now ready to build the Post. The POST request is sent to the CreationFactory URL we saved earlier. The content is added using a p**ost.setEntity(..)** method call.

```
        //Post to the Requirement Factory
        HttpPost post = new HttpPost(factoryURI);
        post.addHeader("Accept", "application/xml");
        post.addHeader("Content-Type", "application/xml");
        post.addHeader("OSLC-Core-Version", "2.0");
        post.setEntity(new StringEntity(content, HTTP.UTF_8));
        HttpResponse sendPostForSecureDocument = HttpUtils.sendPostForSecureDocument(server, post, login,
password, httpclient);
        Header requirementLocation = sendPostForSecureDocument.getFirstHeader("Location");
        sendPostForSecureDocument.getEntity().consumeContent();
```

__5. The last part of Step(4) prints out the response of the HTTP PUT. So, let's run the example and check out the results.

__a. Select the **Example04.java** file in the **Package Explorer**.

__b. *If necessary ,* first change the values of the following variables to match your setup: **NOTE:** Set the version to 4.0 if you want to exercise the new folder API that has been added to this sample.

```
        String server = https://jazz.server.com:9443/rm;
```

```
        String jts_server = https://jazz.server.com:9443/jts;
        String login = "ADMIN";
        String password = "ADMIN";
        String projectAreaName = "JKE Banking (Requirements)";
       String version = "4.0";    // specify the version of the RM server, if no
version specified, the default is 3.0
```

__c. Make sure you save the file (**Ctrl-S)**.

__d. Run the example as a Java application.

__e. The Console view should provide the following similar output:

```
>> Created Requirement
[https://jazz.server.com:9443/rm/resources/_j5Viau3PEeCJqrnKYnBQVA]
```

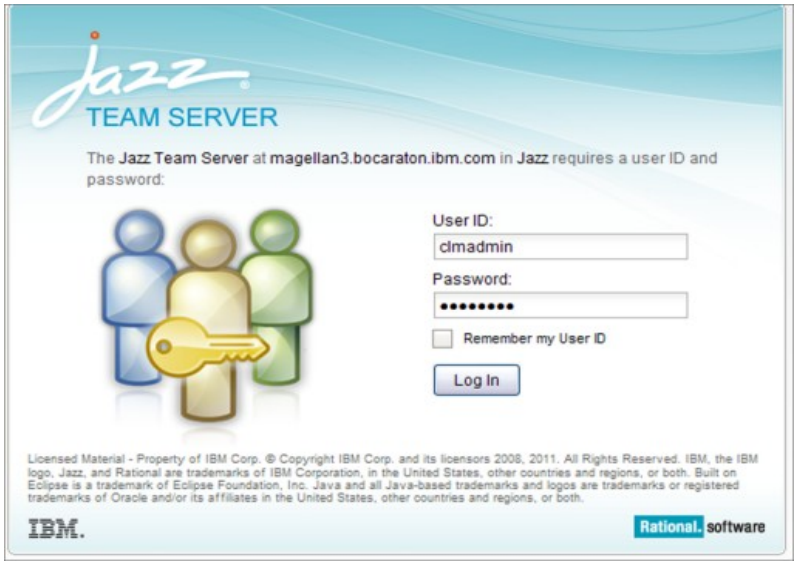
__6. Note the URL of the created Requirement from the bottom of the Console view.

__7. Open the **Firefox** internet browser by double-clicking the **Mozilla Firefox** shortcut 

on the **Windows Desktop.**

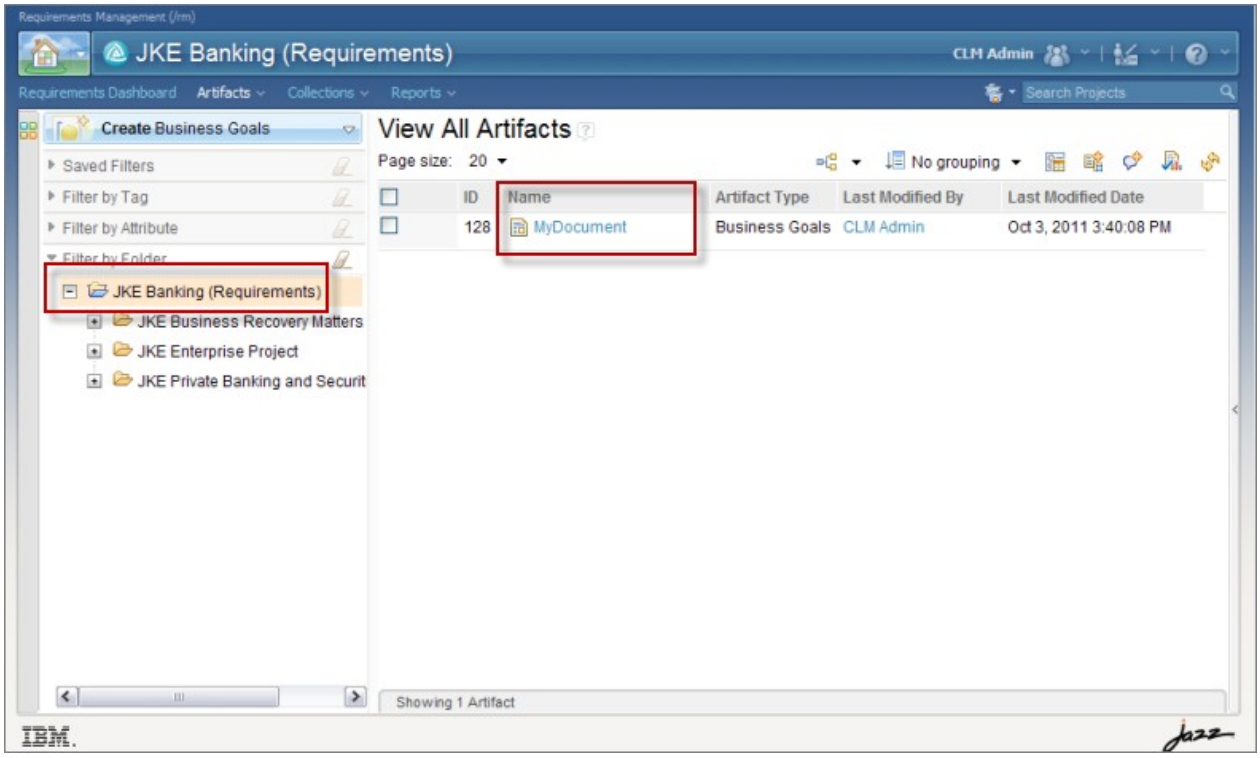
__8. Open the RM application: https://jazz.server.com:9443/rm.

__9. If you were not already logged in, the web UI will display the login dialog.
Login with your current admin login.

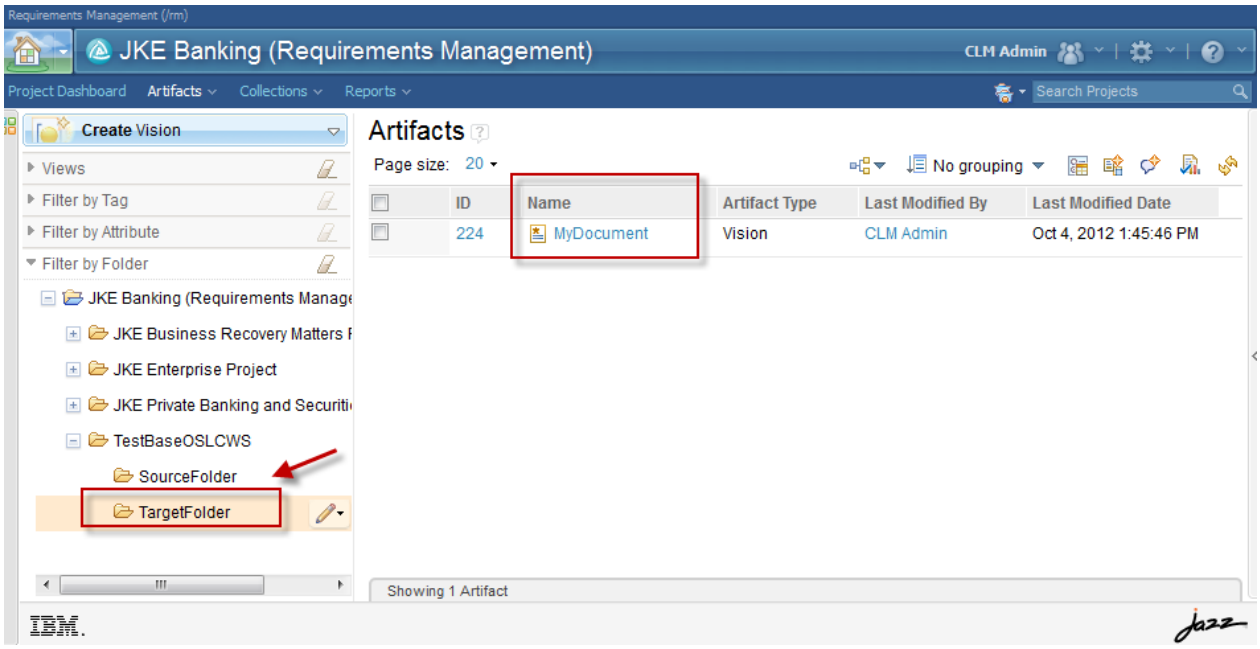


__10. From the Requirements dashboard, click on the RM project used for Example04 to open the Artifact dashboard.

__11. On the Artifact dashboard, in the root folder of the project, you should find the newly created requirement.



If the version has been set to “4.0” and the sample is run against a 4.0 server, then the newly created requirement will be created in a folder called 'TestBaseOSLCWS\TargetFolder'.



Folder support

Let's explore the changes that were added to Example04 to support folder creation and creating a requirement in a folder.

In order to create a folder programmatically via the OSLC RM API, it is necessary to

- construct the folderCreationFactory URI
- assemble the folder xml which includes the folderName and the parentFolder
- POST the xml to the folderCreationFactory URI

__1. Scroll to the **createFolder** method of Example04. Notice how we construct the folderCreationFactory URI: **https://<server:port>/rm/folders** + **?projectURL=** + **https://<server:port>/jts** + **/process/project-areas/** + **<project-id>**

The projectID is retrieved from the serviceProvider URI.

```
// Create the URL Creation factory
String targetProject = "?projectURL=" + this.JTS_Server + "/process/project-areas/" + projectID;
String folderCreationFactory = this.server + "/folders" + targetProject;
```

__2. If the parentFolder is to be the root of the project, the parentFolder URI is constructed as follows, where this.server is equal to **https://<server>:<port>/rm**

```
if ( parentfolder == null ) {
   //lets create it on the root
   parentfolder = this.server + "/folders/" + projectID;
}
```

__3. Now, we need to construct the folder XML body which will include the folderName and the parentFolder.

```
   // Create the body content
   StringBuffer folderBody = new StringBuffer();
   folderBody.append("<rdf:RDF\n")
   .append("xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns%23\"\n")
   .append("xmlns:dcterms=\"http://purl.org/dc/terms/\"\n")
   .append("xmlns:oslc=\"http://open-services.net/ns/core%23\"\n")
   .append("xmlns:nav=\"http://jazz.net/ns/rm/navigation%23\">\n")
   .append("<nav:folder rdf:about=\"\">\n")
   .append("<dcterms:title>" + folderName + "</dcterms:title>\n")
   .append("<nav:parent rdf:resource=\"" + parentfolder + "\"/>")
   .append("</nav:folder>\n")
   .append("</rdf:RDF>");
```

__4. Next, we do a POST to the foldercreationfactoryURI and retrieve the folder location from the **Location** header.

```
final String content = folderBody.toString();
```

```
//Post to the Requirement Factory
HttpPost post = new HttpPost(folderCreationFactory);
post.addHeader("Accept", "application/xml");
post.addHeader("Content-Type", "application/xml");
post.addHeader("OSLC-Core-Version", "2.0");
post.setEntity(new StringEntity(content, HTTP.UTF_8));
HttpResponse sendPostForSecureDocument = HttpUtils.sendPostForSecureDocument(server, post, login,
password, httpclient);

Header folderLocation = sendPostForSecureDocument.getFirstHeader("Location");
sendPostForSecureDocument.getEntity().consumeContent();
return folderLocation.getValue();
```

5.7 Query

In this example, we will describe how to use the Query Capability OSLC-RM REST service to query for a requirement resource. Remember that we introduced these REST services in Lab 5.

__1. In the **Package Explorer** view, open the **Example05.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 the steps to fetch the query capability element in the Service Provider services document, and query for a requirement.

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

// Step (2) : Retrieve the designated Service Provider (Project Area)
String serviceProviderURI = getServiceProvider(catalogURI, projectAreaName);
System.out.println(">> RM Project ["+projectAreaName+"]: "+projectAreaURI);

// Step (3) : Find the query capability element
String query = getQueryCapability(serviceProviderURI);
```

__2. We will not spend time on the 2 first steps which have been explained during the previous labs. Step (3) is interesting because the code finds the Query Capability URI. Let's examine the **getQueryCapability(...)** method.

__a. First, we have to locate the queryCapability URL for the specific service provider or RM project we are interested in. This would be the target of the query operation, meaning, we would be searching in this project.

```
// Define the XPath evaluation environment
XPath xpath = getXpathNamespace();

//Resource Type
String resourceType = "http://open-services.net/ns/rm#Requirement";

String requirementQueryUrl ="//oslc:QueryCapability/oslc:resourceType[@rdf:resource=\"" + resourceType
+"\"]/../oslc:queryBase/@rdf:resource";
Node evaluate = (Node)xpath.evaluate(requirementQueryUrl, source, XPathConstants.NODE);
String query = evaluate.getTextContent();
System.out.println("Query URI->> " + query);
response.getEntity().consumeContent();

return query;
```

__3. Now that we have the QueryCapability URI, we can now proceed to invoke the query in the **performQuery(...) method.**

__a. In our example, we want to search for all resources that contain the text '**Vision**' and in the results we only want to see the title of the resource. In constructing the portion of the POST request that includes the search string, we must use **oslc.prefix, oslc.searchTerms,** and **oslc.select.**

The **oslc.prefix** will define the namespace for the attributes: **dcterms** and **types.** We must include this **oslc.prefix** because we want to use **dcterms:title** and **types:PrimaryText** in the Query request. Remember, from Lab 5, we learned that **oslc.searchTerms** where we provide the text we are searching for and **oslc.select** is where we indicate that we only want to display title in the query results.

The code below puts together the string to append to the queryCapabilityURI.

```
// Free text query for "Vision"
String oslcSearchForVisionQuery = queryCapabilityURI +
    "&oslc.prefix=" + URLEncoder.encode("dcterms=<http://purl.org/dc/terms/>", "UTF8") + "," +
    URLEncoder.encode("types=<http://www.ibm.com/xmlns/rdm/types/>", "UTF8") + "," +
    "&oslc.select=" + URLEncoder.encode("dcterms:title", "UTF8") + "," +
    URLEncoder.encode("types:PrimaryText", "UTF8") +
    "&oslc.searchTerms=" + URLEncoder.encode("\"Vision\"", "UTF8");

String[] lhsrhs = oslcSearchForVisionQuery.split("\\?");
String url = lhsrhs[0];
String body = lhsrhs[1];
```

In the code above, we are adding the following to the QueryCapabilityURI

```
&oslc.prefix=dcterms=<http://purl.org/dc/terms/>,types=<http://www.ibm.com/xmlns/rdm/types/>&oslc.select=dcterms:title,types=PrimaryText&oslc.searchTerms=”Vision”
```

Note that we are also using URLEncoder to properly encode these parameters.

__b. After constructing the query URI, we will perform a POST to the QueryCapability service using the QueryCapability URI. In this POST, we will include the **oslcSearchForVisionQuery** string.

```
HttpPost post = new HttpPost(url);
post.addHeader("Accept", "application/rdf+xml");
post.addHeader("OSLC-Core-Version", "2.0");
post.addHeader("Content-type", "application/x-www-form-urlencoded");

HttpEntity entity = new ByteArrayEntity(body.getBytes("UTF8"));
post.setEntity(entity);
HttpResponse postResponse = HttpUtils.sendPostForSecureDocument(server, post, login, password, httpclient,
200);
if (postResponse.getStatusLine().getStatusCode() == 200) {
    InputStream responseStream = postResponse.getEntity().getContent();
```

```
    byte[] responseBytes = toBytes(responseStream);
    String responseString = new String(responseBytes, "UTF8");
    System.out.println(responseString);
    else{
    System.out.println(postResponse.getStatusLine());
}
```

__c. The POST response will include a listing of all the requirements that match the fulltext string search (“donor”) and the query results will display the title and primaryText of the requirements.

__4. Let's run the example and check out the results.

__a. Select the **Example05.java** file in the **Package Explorer**.

__b. *If necessary* first change the values of the following variables to match your setup:

```
String server = https://jazz.server.com:9443/rm;
String jts_server = https://jazz.server.com:9443/jts;
String login = "ADMIN";
String password = "ADMIN";
String projectAreaName = "JKE Banking (Requirements)";
```

__c. Make sure you save the file (**Ctrl-S)**.

__d. Run the example as a Java application.

__e. The Console view should provide the following similar output:

```
    >> POST(1) https://magellan3.bocaraton.ibm.com:9443/rm/views
>> Response Headers:
    - Server: Apache-Coyote/1.1
    - X-Last-Modified-XSD: 2011-10-03T14:59:48.546Z
    - Content-Type: application/atom+xml;charset=UTF-8
    - Content-Length: 2919
    - Date: Wed, 05 Oct 2011 01:29:57 GMT
<rdf:RDF xmlns:oslc="http://open-services.net/ns/core#"
   xmlns:oslc_rm="http://open-services.net/ns/rm#"
   xmlns:dcterms="http://purl.org/dc/terms/"
   xmlns:rmTypes="http://www.ibm.com/xmlns/rdm/types/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
>
   <oslc:ResponseInfo rdf:about="https://magellan3.bocaraton.ibm.com:9443/rm/views">
      <dcterms:title>Query Results: 3</dcterms:title>
   </oslc:ResponseInfo>
   <rdf:Description rdf:about="http://example.com/query">
      <rdfs:member>
         <oslc_rm:Requirement
rdf:about="https://magellan3.bocaraton.ibm.com:9443/rm/resources/_JH2oCrE2EeC--cDfe44EPg">
            <dcterms:title>Business Recovery Matters Vision</dcterms:title>
```

<rmTypes:PrimaryText>**Business Recovery Matters Capabilities**

Stakeholder Needs

Dividends for a Cause Capabilities

Donors can specify contribution criteria

Donors can specify when to start and stop contributing</rmTypes:PrimaryText>
</oslc_rm:Requirement>
</rdfs:member>
<rdfs:member>
<oslc_rm:Requirement
rdf:about="https://magellan3.bocaraton.ibm.com:9443/rm/resources/_I_D4k7E2EeC--cDfe44EPg">
<dcterms:title>**Vision Template**</dcterms:title>
<rmTypes:PrimaryText>Vision: <<Project Name>>

1. Introduction

<< Provide a brief introduction of the project>>

2. Positioning

2.1 Problem Statement

<<Provide a statement summarizing the problem being solved by this project. The following format may be used>>
The problem of <<describe the problem>>

Affects <<the stakeholders affected by the problem>>

The impact of which is <<what is the impact of the problem>>

A successful solution would be <<list some key benefits of a successful solution>>

2.2 Product Position Statement

Provide an overall statement summarizing, at the highest level, the unique position the product intends to fill in the marketplace. A product position statement communicates the intent of the application and the importance of the project to all concerned personnel. The following format may be used:
For <<target customer>>

Who <<statement of the need or opportunity>>

The <<product name>&</rmTypes:PrimaryText>
</oslc_rm:Requirement>
</rdfs:member>
<rdfs:member>
<oslc_rm:Requirement

```
rdf:about="https://magellan3.bocaraton.ibm.com:9443/rm/resources/_JM7VVLE2EeC--cDfe44EPg">
            <dcterms:title>JKE Requirements Process Overview</dcterms:title>
          </oslc_rm:Requirement>
       </rdfs:member>
    </rdf:Description>
</rdf:RDF>
```

5.8 Create OSLC Servlet for RM

At this point of the workshop, let's create a **MyRMPortal** application to take advantage of the OSLC-RM Consumer API.

In this example, we will reuse code from the simple Web portal application created in Lab 3 so that it lists all the project areas / RM Projects of a designated OSLC-RM server.

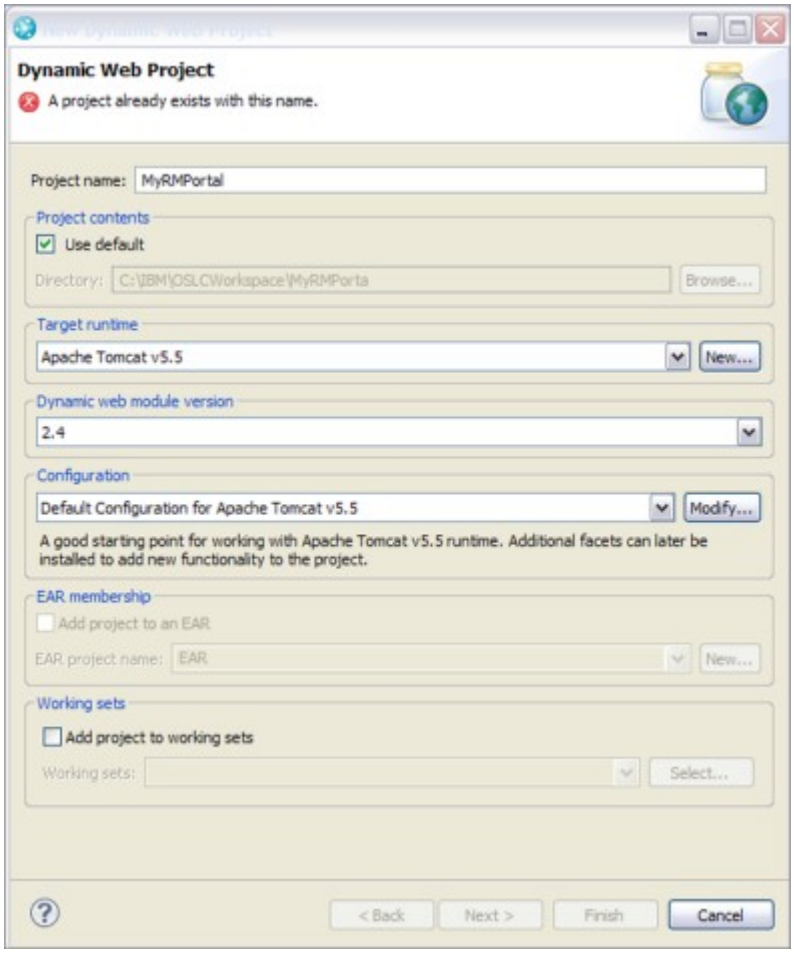
__15. Launch the Eclipse client

__16. Create an Eclipse Dynamic Web project.

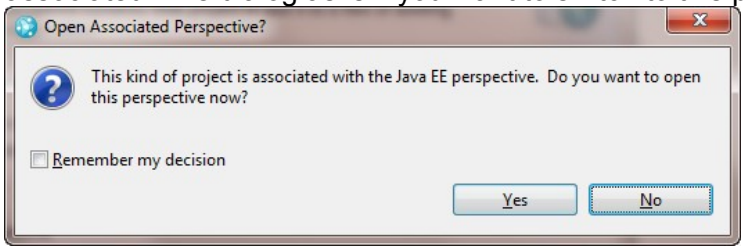
__a. From the **File** menu, select **New > Other...**

__b. From the **New** dialog wizard select the **Dynamic Web Project** item and press **Next**.

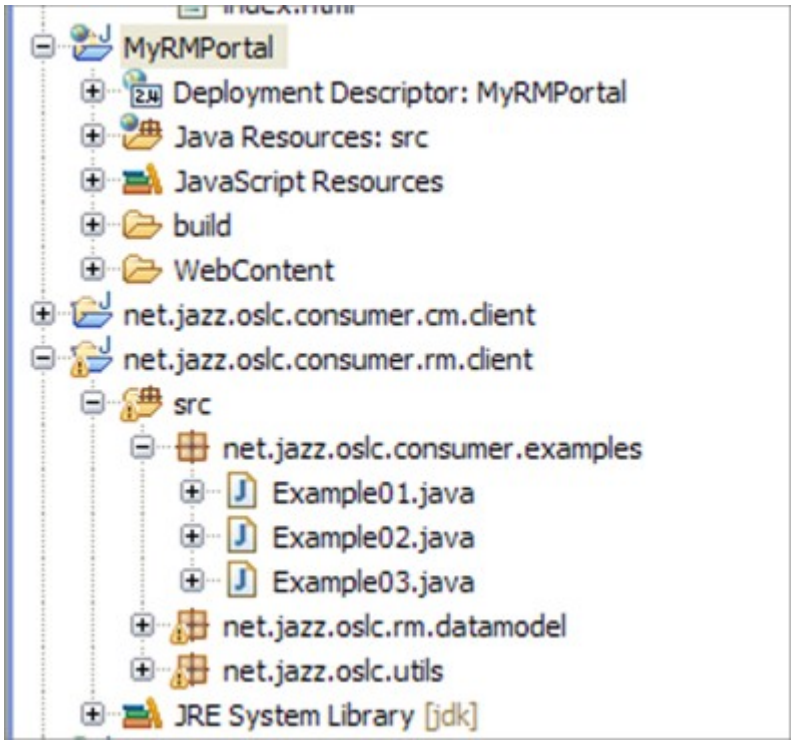
__c. For the project name, type **MyRMPortal** and press **Finish**.



A dialog will pop up telling you that for this kind of project, the Java EE perspective is associated. The dialog asks if you want to switch to this perspective. Answer **Yes**.



At this point, you should have a new eclipse project named **MyRMPortal** listed in the **Project Explorer** view of the **Java EE** perspective.

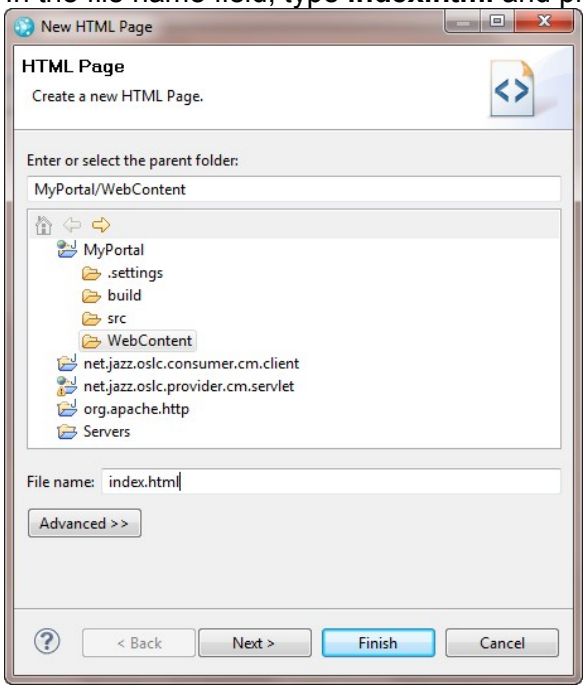

MyRMPortal
Deployment Descriptor: MyRMPortal
Java Resources: src
JavaScript Resources
build
WebContent
net.jazz.oslc.consumer.cm.client
net.jazz.oslc.consumer.rm.client
src
net.jazz.oslc.consumer.examples
Example01.java
Example02.java
Example03.java
net.jazz.oslc.rm.datamodel
net.jazz.oslc.utils
JRE System Library [jdk]

__17. Create an HTML page to connect to an OSLC RM server and display its Project Areas. For that we will need to know the Public URI of the server, and the login / password for authentication.

__a. Click on the **WebContent** folder of your new Eclipse project.

__b. From the context menu, select **New > HTML Page** to create an HTML page

__c. In the file name field, type **index.html** and press **Finish**



This wizard creates and HTML page using a default template and open the HTML editor on it.

__d. Between the **<title>** tags replace **Insert title here** by **My RM Project portal**

__e. Between the **<body>** tags insert the following HTML snipped code:

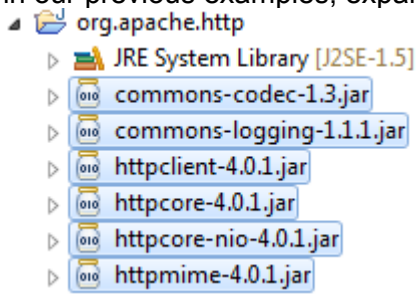
```
<h2>Connect to:</h2>
<form action="ListOfRMProjects" method="get">
    <table>
        <tr><td>Public URI:</td> <td><input type="text" name="publicURI" size="40"/></td></tr>
        <tr><td>Login:</td> <td><input type="text" name="login" size="20"/></td></tr>
        <tr><td>Password:</td> <td><input type="password" name="password" size="20"/></td></tr>
        <tr></tr>
        <tr><td><input type="submit" value="Submit"/></td></tr>
    </table>
</form>
```

Please notice the **ListOfRMProjects** value of the **action** attribute of the form. It designates the name of the servlet we will implement in the next step.

Please notice also the method we will use to connect to the server (**get**), it designates the method called in the servlet.

__18. Add the libraries that your servlet will require.

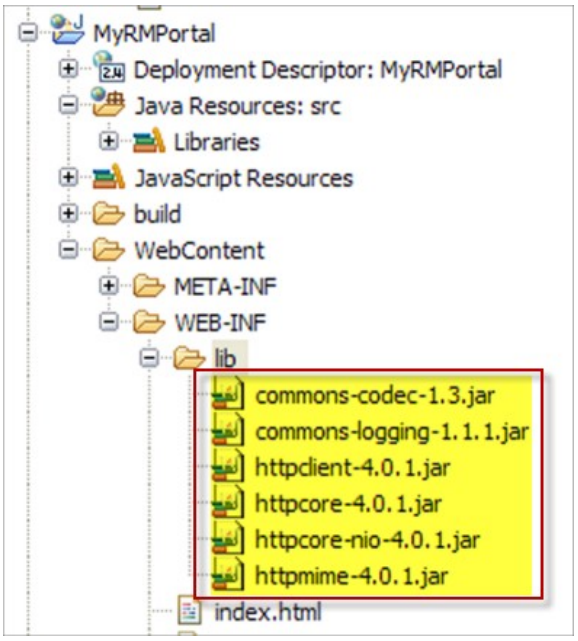
__a. Because we will use in this new servlet the same Apache HTTP client APIs that we used in our previous examples, expand the org.apache.http project and select the listed jars:



__b. From the context menu, select **Copy**.

__c. In the Web project (**MyRMPortal**), select the **WebContent/WEB-INF/lib** folder.

__d. From the context menu, select **Paste**. At this point you should retrieve all the jars you have copied into the **lib** folder:



__19. It is time to create the servlet which will connect to the OSLC RM server, retrieve the list of RM Projects and return an HTML page showing this list.

__a. From the context menu of the **MyRMPortal** Web project, select **New > Servlet**.

__b. For the **Java package** of the servlet, type: **net.jazz.oslc.consumer.rm.servlets**

__c. For the Class name of the servlet, type: **ListOfRMProjects**. As mentioned earlier, this name should be the same as the one that you specify as the value of the action attribute in the form.

__d. Press **Finish**. This wizard creates a new class, subclass of `javax.servlet.http.HttpServlet` and binds the URL **/ListOfRMProjects** to the designated class: **net.jazz.oslc.consumer.cm.servlets.ListOfRMProjects**.

FYI, this binding is described in the file **WebContent/WEB-INF/web.xml**.

__20. Write the code of your servlet

A Java editor is now open against this new servlet class. This class implements 2 default methods: **doGet**, to support the HTTP GET method and **doPost** to support the HTTP POST method. As we mentioned earlier, our form will submit its parameters using the HTTP GET method, which means that the doGet method will be called.

__21. The servlet we want to implement is very similar to the servlet in the MyPortal web application from Lab 3. So, we will use this opportunity to copy/paste the code of the ListOfProjectAreas.java doGet() methos into our servlet and adapt it to our needs.

__a. Edit the List example located in MyPortal->net.jazz.oslc.consumer.cm.servlets/ListOfProjectAreas.java

__b. Copy all the code contained in the **doGet()** method and paste it in the **doGet** method of this servlet.

__c. In the pasted code, instead of using the cmServiceProviders, use the rmServiceProviders.

```
String catalogXPath = "/rdf:Description/oslc_rm:rmServiceProviders/@rdf:resource";
```

__d. Now, instead of using these NamespaceContextMap String array values, replace with the following:

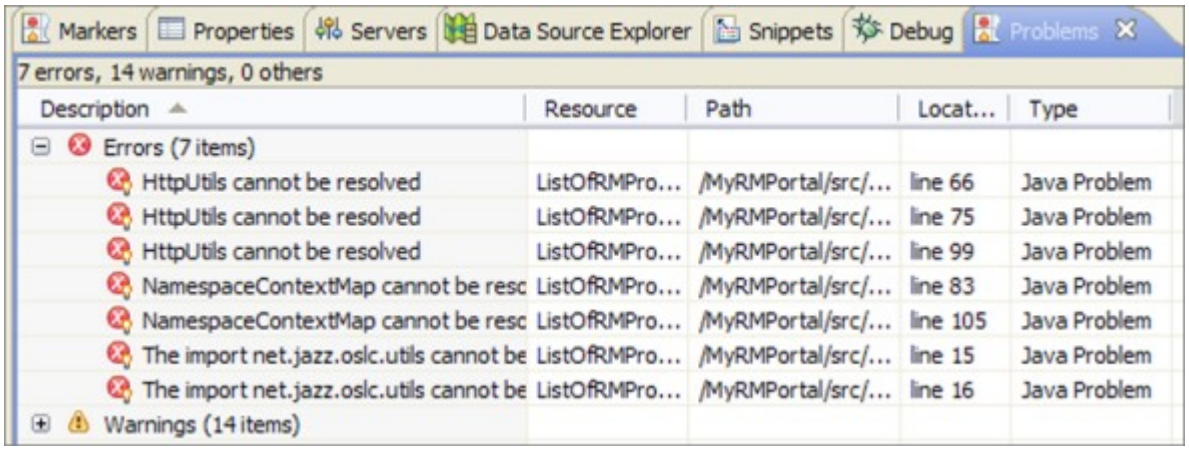
```
new NamespaceContextMap(new String[]
        {       "rdf", "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
                "oslc_rm","http://open-services.net/xmlns/rm/1.0/"}));
```

__e. Scroll down to the HTML response section and edit it to look as follows:

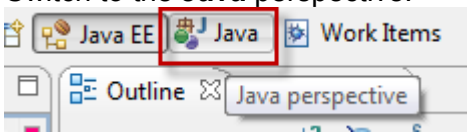
```
// Build the HTML response
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();

out.println("<html>");
out.println("<head>");
out.println("<title>RM Project List</title>");
out.println("</head>");
out.println("<body>");
out.println("<h1>RM Projects in: "+server+"</h1>");
 out.println("<ul>");

    // Print out the title of each Service Provider
    int length = titleNodes.getLength();
    Node node;
    Element element;
    String name;
    String url;
    for (int i = 0; i < length; i++) {
        node = titleNodes.item(i);
        name = node.getTextContent();
        element = (Element)(node.getParentNode());
        url = element.getAttribute("rdf:about");
        out.println("<li><a href=\""+url+"\">"+name+"</a></li>");
    }

 out.println("</ul>");
 out.println("</body>");
 out.println("</html>");

 out.close();
```

__f. Save the file.

__22. At the point you should still have few errors. To fix them, you need to copy the package containing the classes `HttpUtils` and `NamespaceContextMap` into your Web project.



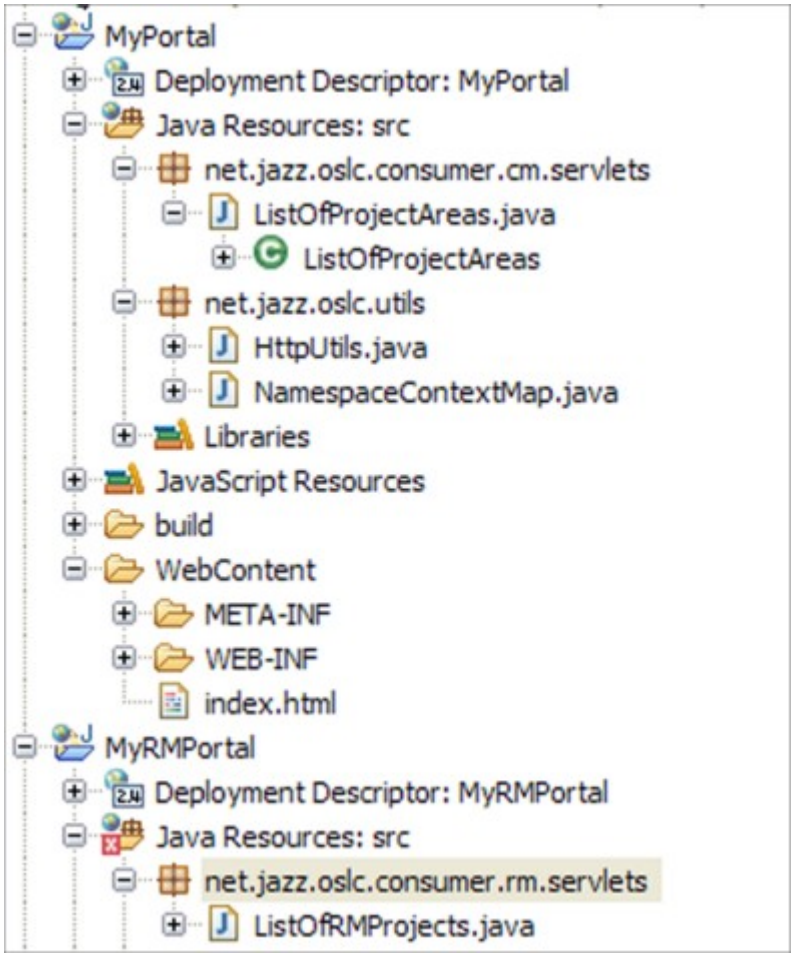
__a. Switch to the **Java** perspective.



__b. In the **MyPortal** Eclipse project, select the package **net.jazz.oslc.utils** package under the **src** source folder.

__c. Access the context menu and select **Copy**.

__d. In the **MyRMPortal** Web project, select the **net.jazz.oslc.consumer.rm.servlets** package folder.



__e. In the context menu select **Paste**.
This action should fix all the errors that you had in your servlet.

__f. Switch back to the **Java EE** perspective.

__23. It is time to test the servlet.

__a. Your Jazz Team Server should be running. If it is not the case, start it.

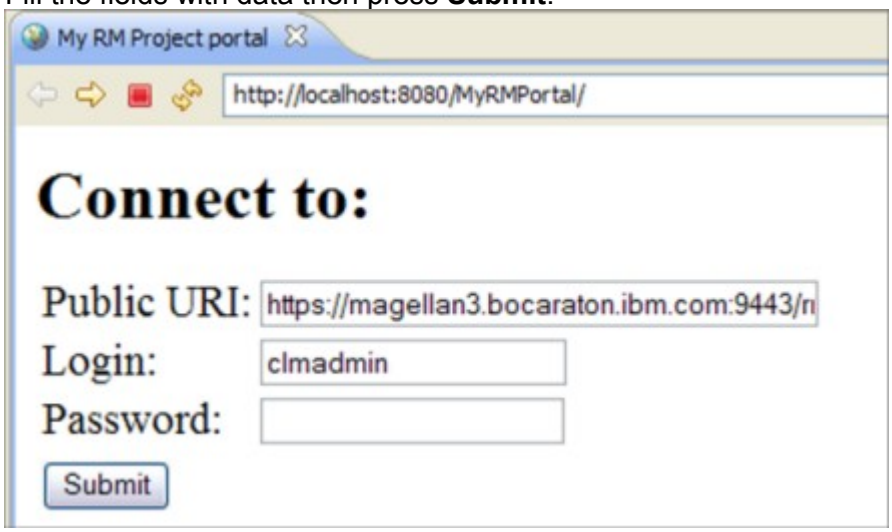
__b. Select the **MyRMPortal** web project.

__c. From the context menu select **Run As > Run on Server**

__d. The workbench will ask to select the Application Server to run against.
Select your Tomcat server and press **Finish**.

The server should start and the **index.html** page should appear into the Eclipse embedded web browser.

Fill the fields with data then press **Submit**.



__e. After few seconds, a new page will appear with the list of the Project Areas contained in the referenced repository.

__f. At this point, using this servlet-based pattern you can build any type of web application consuming OSLC RM APIs.

__24. Shut down the Jazz Team Server by running:
`(<JazzTeamServer_Root_Folder>\server\server.shutdown.bat).`

Conclusion

This last example concludes our two labs on how to consume the OSLC-RM 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…

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.

Appendix B Trademarks and copyrights

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

IBM	AIX	CICS	ClearCase	ClearQuest	Cloudscape
Cube Views	DB2	developerWorks	DRDA	IMS	IMS/ESA
Informix	Lotus	Lotus Workflow	MQSeries	OmniFind	
Rational	Redbooks	Red Brick	RequisitePro	System i	
System z	Tivoli	WebSphere	Workplace	System p	

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.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

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

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

ITIL is a registered trademark and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

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

NOTES

NOTES

Please Recycle

IBM Software