WA2028 Mastering JAX-RS RESTful Web Services and AJAX Clients

Student Labs

Web Age Solutions Inc.

Table of Contents

Lab 1 - Configure the Development Environment

Throughout these labs, you will be developing web service components with two main pieces of software: Oracle WebLogic Server v12c and Oracle Enterprise Pack for Eclipse(OEPE).

Oracle WebLogic Server v12c(12.1.1) (WLS) is a JEE application server that serves up JEE applications, and is also a web service container.

OEPE, Provides tools that make it easier to develop applications utilizing specific Oracle Fusion Middleware technologies and Oracle Database. Based on the Eclipse tool, it is the programming environment we will use for this class.

In this lab exercise, we will configure WLS and OEPE to form our development environment.

Part 1 - Oracle WebLogic server Setup

WebLogic Server is already extracted into **C:\Software\WebLogic** folder.

In this part we will setup necessary Environment variables and run installation configuration script.

Follow the instructions to set up JAVA_HOME, MW_HOME and JAVA_VENDOR environment variable

JAVA_HOME=C:\Software\Java

MW_HOME=C:\Software\WebLogic

JAVA_VENDOR=Oracle

__1. Right click on the **My Computer** icon on your computer and select **properties**.

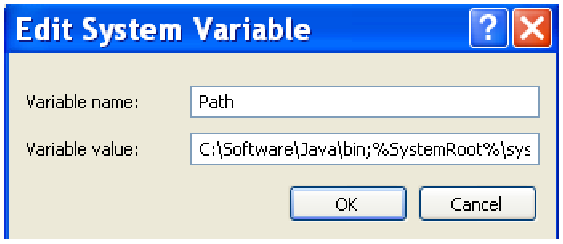
__2. Click the **Advanced** Tab.

__3. Click the **Environment Variables** button.

__4. From the *System Variables* list, select **Path** and click **Edit**.

__5. At the beginning of the line enter the following:

```
C:\Software\Java\bin;
```



Variable name: Path

Variable value: C:\Software\Java\bin;%SystemRoot%\sys

__6. Click **OK**.

__7. Under System Variables, click **New**.

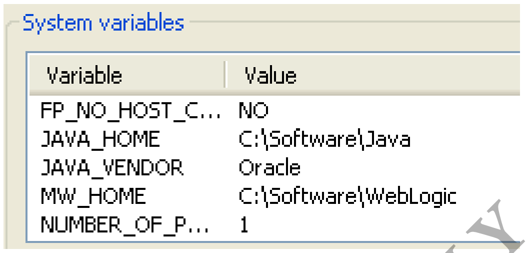
__8. Enter the variable name as **JAVA_HOME**

__9. Enter the variable value as the install path for the Development Kit.

__10. Click **OK**.

__11. Repeat these steps for **MW_HOME** and **JAVA_VENDOR** variables.

Once all done it should look like.



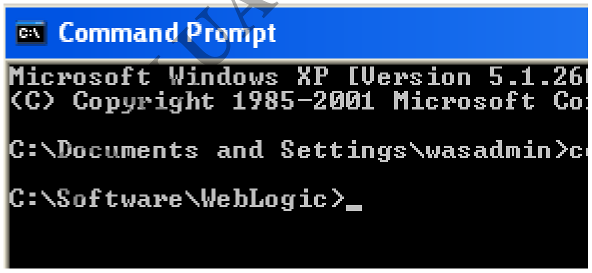
Variable	Value
FP_NO_HOST_C...	NO
JAVA_HOME	C:\Software\Java
JAVA_VENDOR	Oracle
MW_HOME	C:\Software\WebLogic
NUMBER_OF_P...	1

__12. Click OK until you close all windows.

Now you will run the installation configuration script in the MW_HOME directory. It should be run only once.

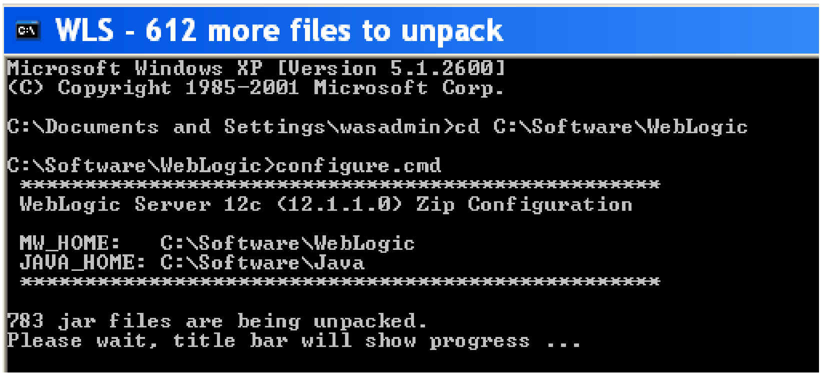
__13. Open a command prompt.

__14. Change the directory to **C:\Software\WebLogic**



```
Microsoft Windows XP [Version 5.1.26
(C) Copyright 1985-2001 Microsoft Co

C:\Documents and Settings\wasadmin>c

C:\Software\WebLogic>_
```

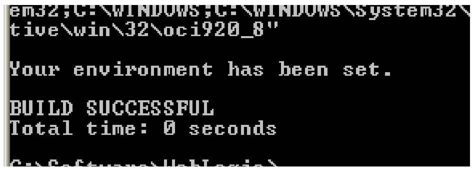
__15. Enter **configure.cmd** and hit Enter.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\wasadmin>cd C:\Software\WebLogic

C:\Software\WebLogic>configure.cmd
 **************************************************
 WebLogic Server 12c (12.1.1.0) Zip Configuration

 MW_HOME:   C:\Software\WebLogic
 JAVA_HOME: C:\Software\Java
 **************************************************

783 jar files are being unpacked.
Please wait, title bar will show progress ...
```

It takes few minutes to complete.

You will get a notification when the installation is done.



```
em32;C:\WINDOWS;C:\WINDOWS\System32\
tive\win\32\oci920_8"

Your environment has been set.

BUILD SUCCESSFUL
Total time: 0 seconds
```

__16. Close the command prompt window.

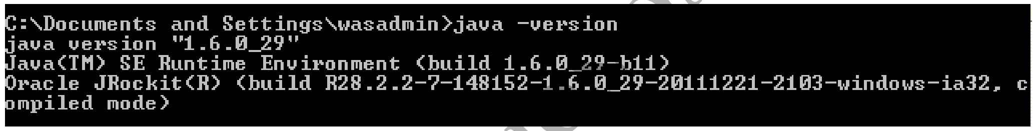
__17. Restart the machine.

__1. Open a Windows command prompt. You can do this by selecting '**Start -> Run**', entering '**cmd**', and then pressing the **OK** button.

__2. Enter the following command:

```
java -version
```

Make sure you see the response shown below.

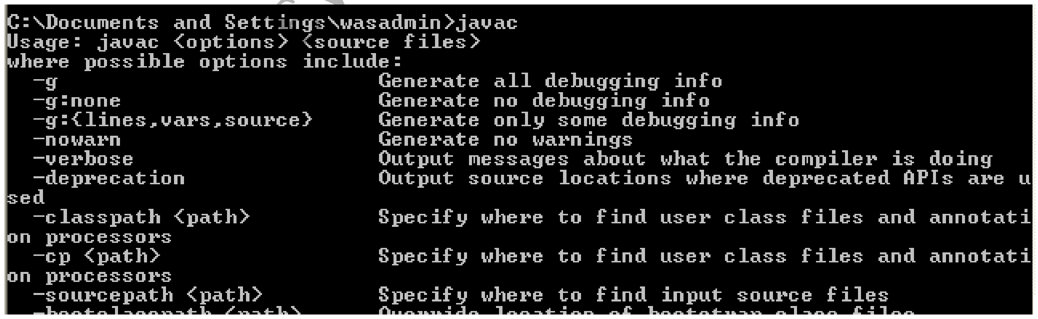


```
C:\Documents and Settings\wasadmin>java -version
java version "1.6.0_29"
Java(TM) SE Runtime Environment (build 1.6.0_29-b11)
Oracle JRockit(R) (build R28.2.2-7-148152-1.6.0_29-20111221-2103-windows-ia32, c
ompiled mode)
```

__3. Enter the following command:

```
javac
```

Verify that you get the options to run the Java compiler:



```
C:\Documents and Settings\wasadmin>javac
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -nowarn                    Generate no warnings
  -verbose                   Output messages about what the compiler is doing
  -deprecation               Output source locations where deprecated APIs are u
sed
  -classpath <path>          Specify where to find user class files and annotati
on processors
  -cp <path>                 Specify where to find user class files and annotati
on processors
  -sourcepath <path>         Specify where to find input source files
```

__4. Close the command prompt window.

Part 2 - Configure a server in OEPE

We now need to configure OEPE to be able to communicate with WLS. Doing so will allow us to perform simple operations with the server (e.g. Starting, stopping and deploying applications). While this is not strictly necessary for a development environment, it significantly simplifies development time. In order to setup the server we should also need to create a domain.

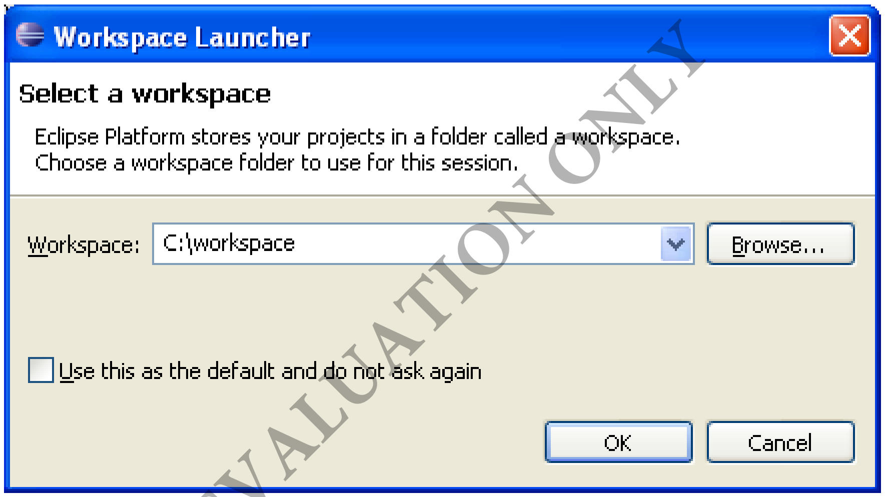
As with WLS, OEPE has already been installed on your computer. We just need to configure it.

__1. Start OEPE by launching **C:\Software\OEPE\eclipse.exe**

OEPE will launch.

You will be prompted to select a Workspace.

__2. Set the *Workspace* to **C:\workspace**



Workspace Launcher

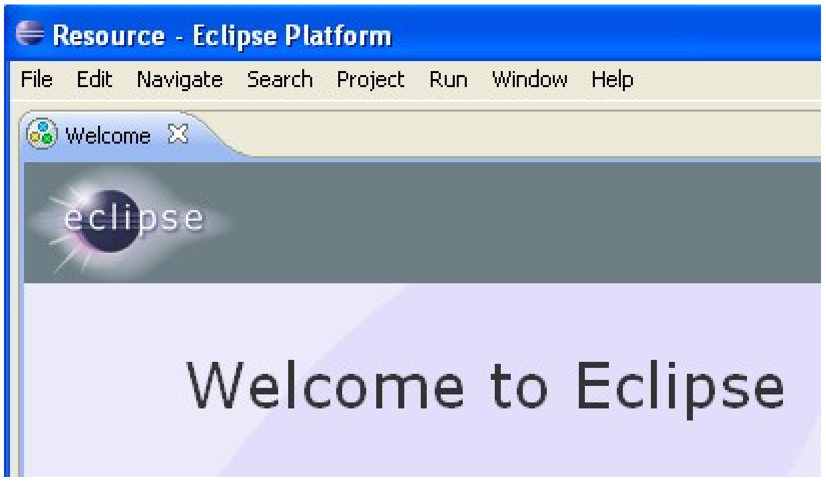
Select a workspace

Eclipse Platform stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\workspace

__3. Click **OK**.

OEPE will start.

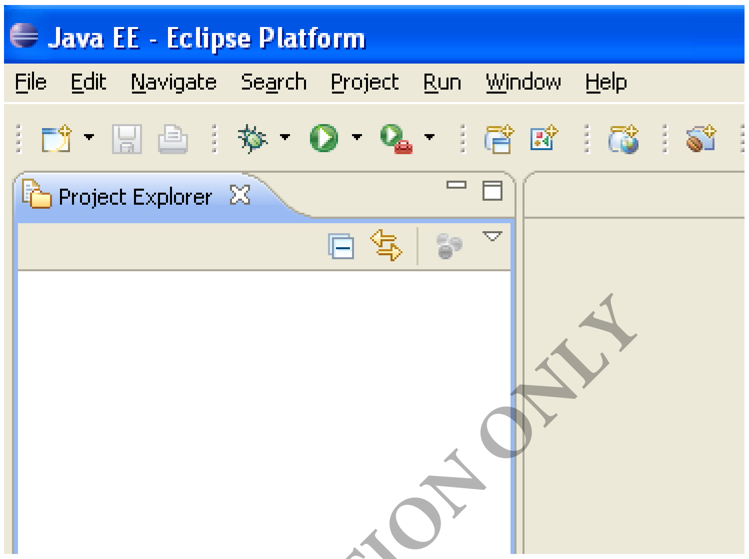

Resource - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Welcome
eclipse
Welcome to Eclipse

As mentioned earlier, OEPE is built on top of Eclipse. If you are familiar with Eclipse, you should have no problems using OEPE.

__4. WTP is displaying the **Welcome** screen (or *view*). Close the view by clicking the **x** in its tab.

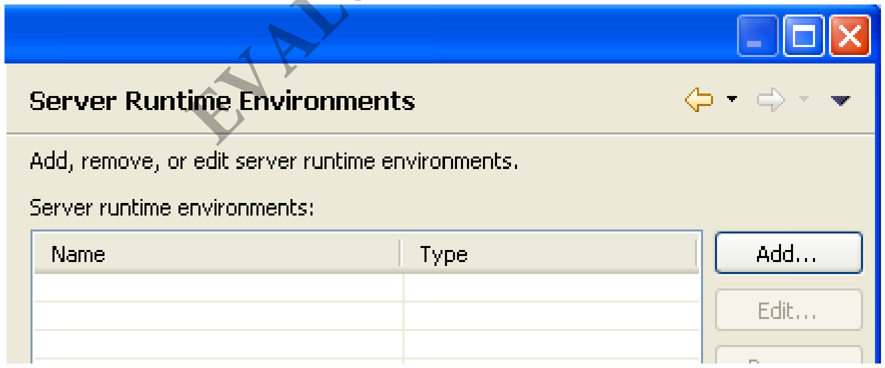
__5. From the menu, select **Window > Open Perspective > Java EE**.

OEPE will then show the **Java EE** *perspective*. (A *perspective* is a collection of views)


Java EE - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Project Explorer

__6. Select **Windows | Preferences**.

__7. Expand **Server | Runtime Environments**.


Server Runtime Environments

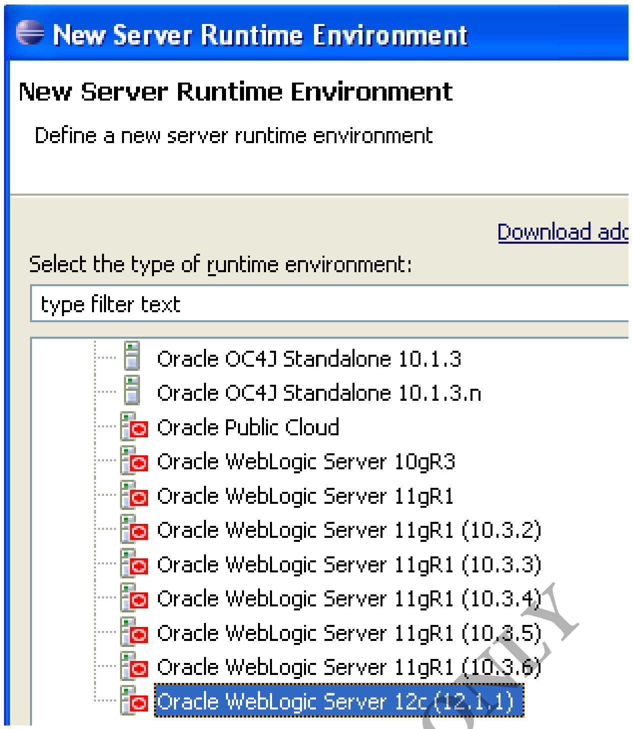
Add, remove, or edit server runtime environments.

Server runtime environments:

Name	Type

__8. Click **Add...**

__9. Expand **Oracle** and select **Oracle Weblogic Server 12c(12.1.1)**


New Server Runtime Environment

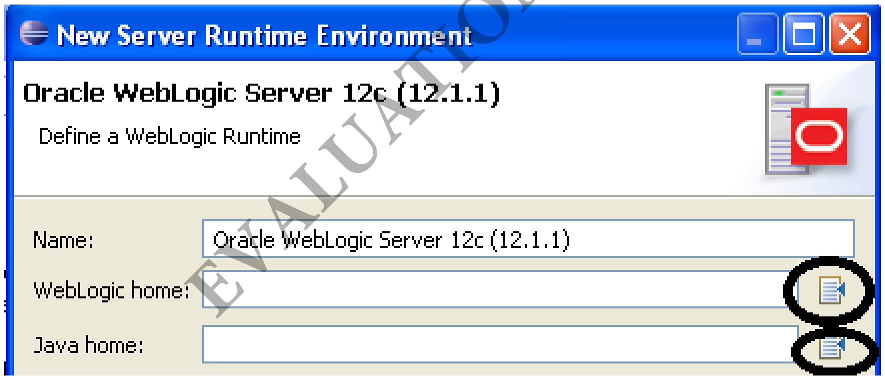
New Server Runtime Environment

Define a new server runtime environment

Select the type of runtime environment:

- Oracle OC4J Standalone 10.1.3
- Oracle OC4J Standalone 10.1.3.n
- Oracle Public Cloud
- Oracle WebLogic Server 10gR3
- Oracle WebLogic Server 11gR1
- Oracle WebLogic Server 11gR1 (10.3.2)
- Oracle WebLogic Server 11gR1 (10.3.3)
- Oracle WebLogic Server 11gR1 (10.3.4)
- Oracle WebLogic Server 11gR1 (10.3.5)
- Oracle WebLogic Server 11gR1 (10.3.6)
- Oracle WebLogic Server 12c (12.1.1)

__10. Click **Next**.


New Server Runtime Environment

Oracle WebLogic Server 12c (12.1.1)

Define a WebLogic Runtime

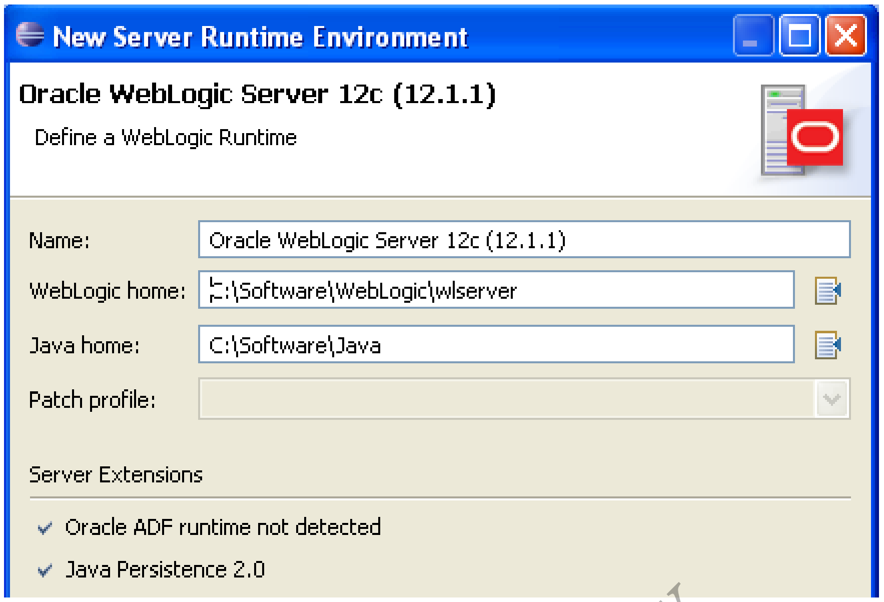
Name: Oracle WebLogic Server 12c (12.1.1)

WebLogic home:

Java home:

__11. For Weblogic home, enter **C:\Software\WebLogic\wlserver** or browse to find the location. Similarly enter **C:\Software\Java** for Java home.

You will see as shown below:


New Server Runtime Environment

Oracle WebLogic Server 12c (12.1.1)

Define a WebLogic Runtime

Name: Oracle WebLogic Server 12c (12.1.1)

WebLogic home: C:\Software\WebLogic\wlserver

Java home: C:\Software\Java

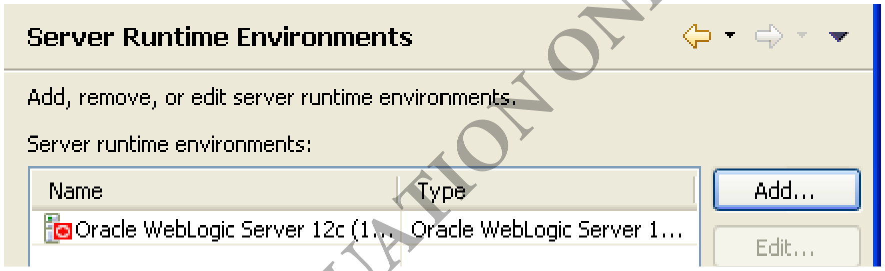
Patch profile:

Server Extensions

✓ Oracle ADF runtime not detected

✓ Java Persistence 2.0

__12. Click **Finish**.


Server Runtime Environments

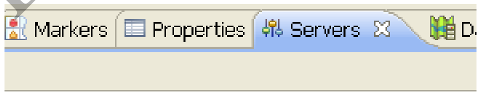
Add, remove, or edit server runtime environments.

Server runtime environments:

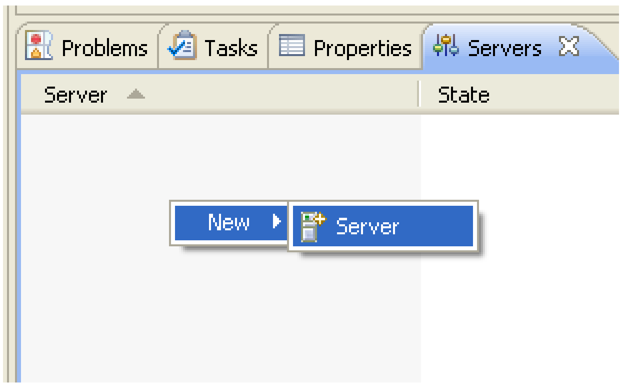
Name	Type
Oracle WebLogic Server 12c (1...	Oracle WebLogic Server 1...

__13. Click **OK**.

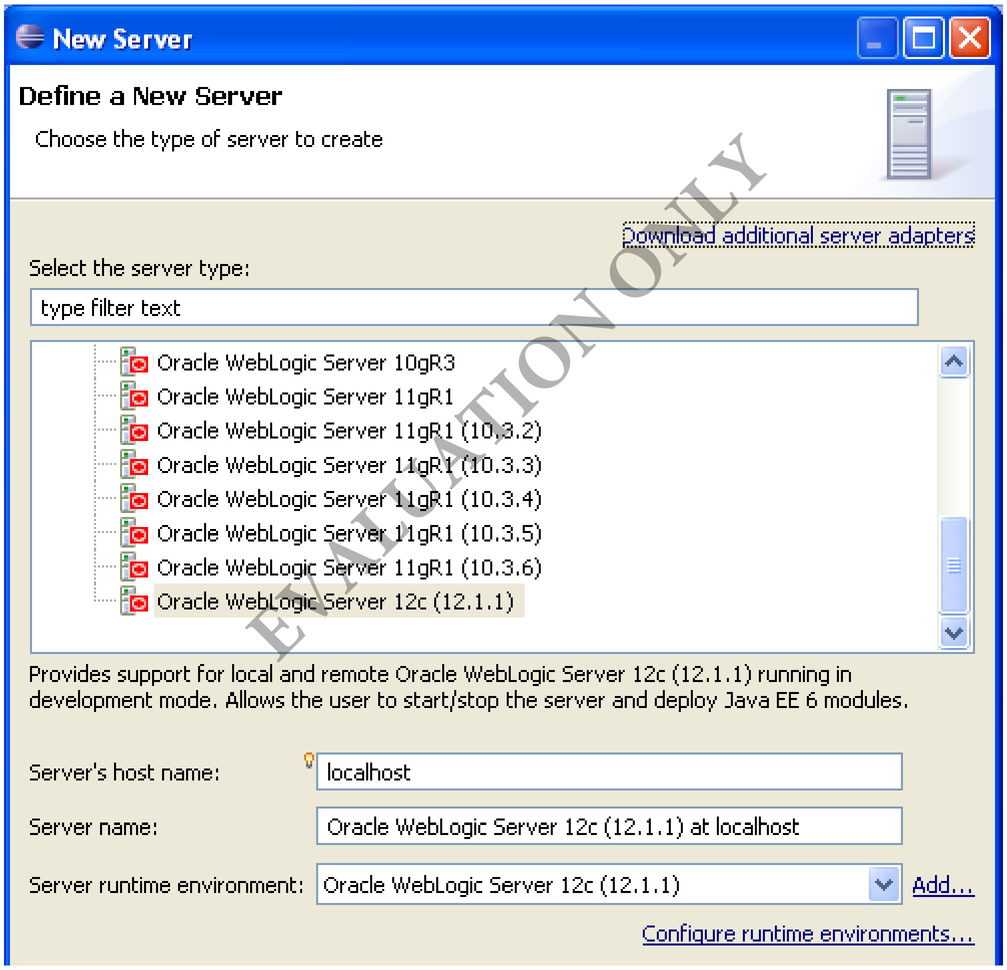
__14. We need to create domain and configure OEPE to point at the domain. Locate the *Server view*. To do this, click the **Servers** tab in the bottom area of OEPE.



__15. Right click anywhere in the area and select **New | Server**.



The *New Server* window will appear.


New Server

Define a New Server

Choose the type of server to create

Download additional server adapters

Select the server type:

type filter text

Oracle WebLogic Server 10gR3
Oracle WebLogic Server 11gR1
Oracle WebLogic Server 11gR1 (10.3.2)
Oracle WebLogic Server 11gR1 (10.3.3)
Oracle WebLogic Server 11gR1 (10.3.4)
Oracle WebLogic Server 11gR1 (10.3.5)
Oracle WebLogic Server 11gR1 (10.3.6)
Oracle WebLogic Server 12c (12.1.1)

Provides support for local and remote Oracle WebLogic Server 12c (12.1.1) running in development mode. Allows the user to start/stop the server and deploy Java EE 6 modules.

Server's host name: localhost

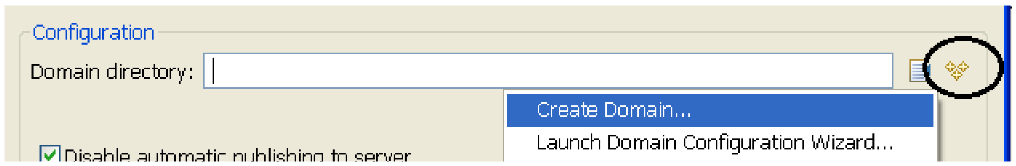
Server name: Oracle WebLogic Server 12c (12.1.1) at localhost

Server runtime environment: Oracle WebLogic Server 12c (12.1.1) Add...

Configure runtime environments...

__16. Select **Oracle WebLogic Server 12c**.

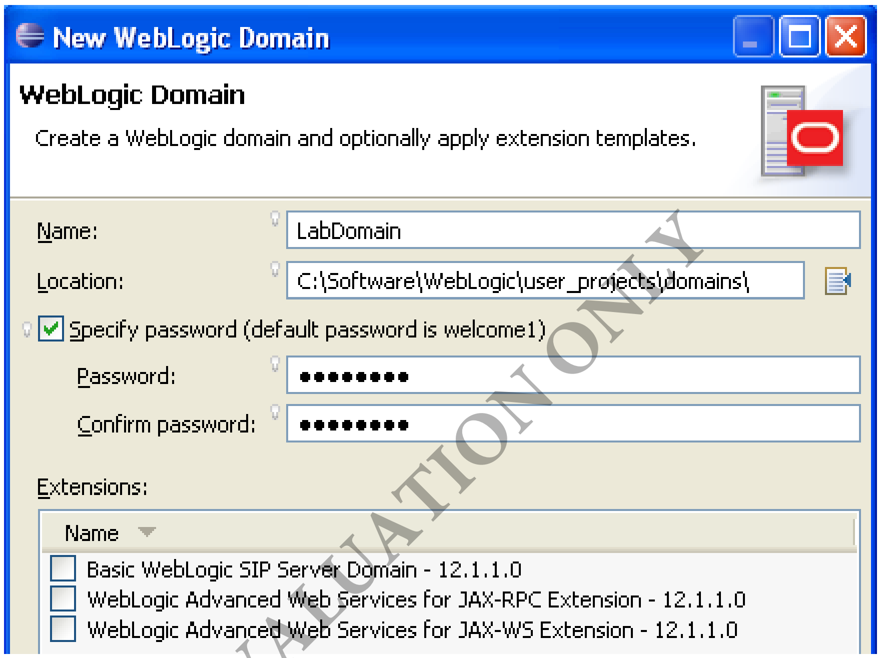
__17. Click **Next**.


Configuration
Domain directory:
Create Domain...
Launch Domain Configuration Wizard...
Disable automatic publishing to server

__18. Next to **Domain directory**, click the **create** icon and then select **Create Domain**.

__19. Enter **LabDomain** as the name.

__20. Check specify password and enter **w1sadmin** as password **where the password contains number one not L.**


New WebLogic Domain

WebLogic Domain

Create a WebLogic domain and optionally apply extension templates.

Name: LabDomain

Location: C:\Software\WebLogic\user_projects\domains\

☑ Specify password (default password is welcome1)

Password: ●●●●●●●●●

Confirm password: ●●●●●●●●●

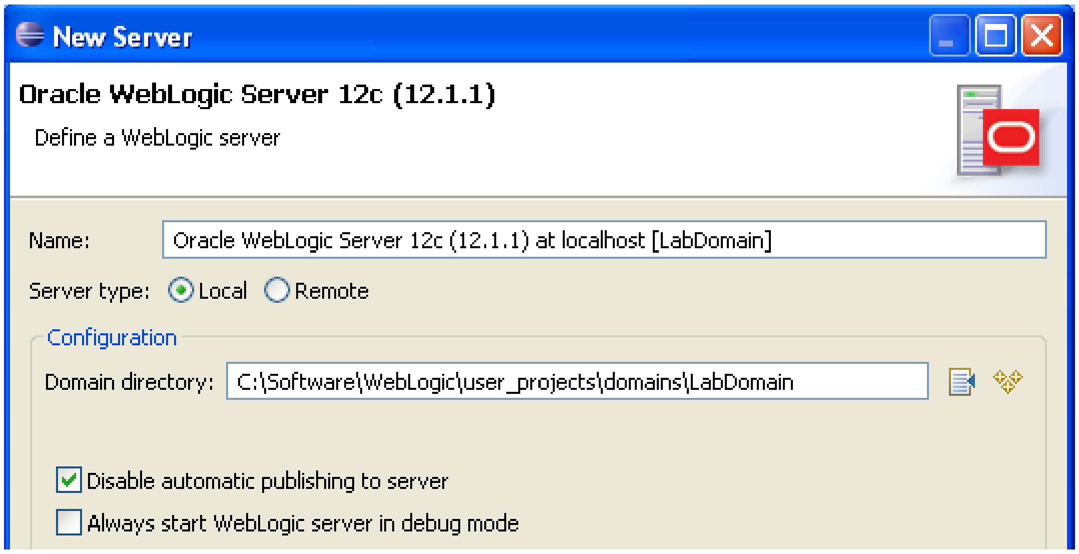
Extensions:

Name
☐ Basic WebLogic SIP Server Domain - 12.1.1.0
☐ WebLogic Advanced Web Services for JAX-RPC Extension - 12.1.1.0
☐ WebLogic Advanced Web Services for JAX-WS Extension - 12.1.1.0

__21. Don't select any extensions, because we will not be using any of the advanced features like WS-Addressing.

__22. Click **Finish**.

__23. After some seconds the below screen will open, make sure **Disable Automatic Publishing to server** is checked.


New Server

Oracle WebLogic Server 12c (12.1.1)

Define a WebLogic server

Name: Oracle WebLogic Server 12c (12.1.1) at localhost [LabDomain]

Server type: [✓ Local] Remote

Configuration

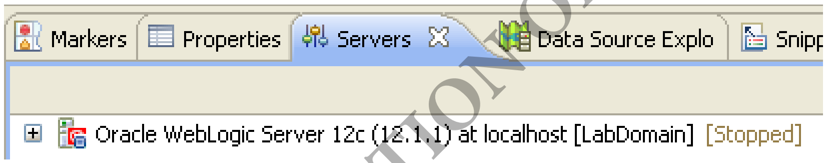
Domain directory: C:\Software\WebLogic\user_projects\domains\LabDomain

[✓] Disable automatic publishing to server

[ ] Always start WebLogic server in debug mode

__24. Click **Finish**.

The server connection has been defined, and the server should now be listed in the *Servers* view.

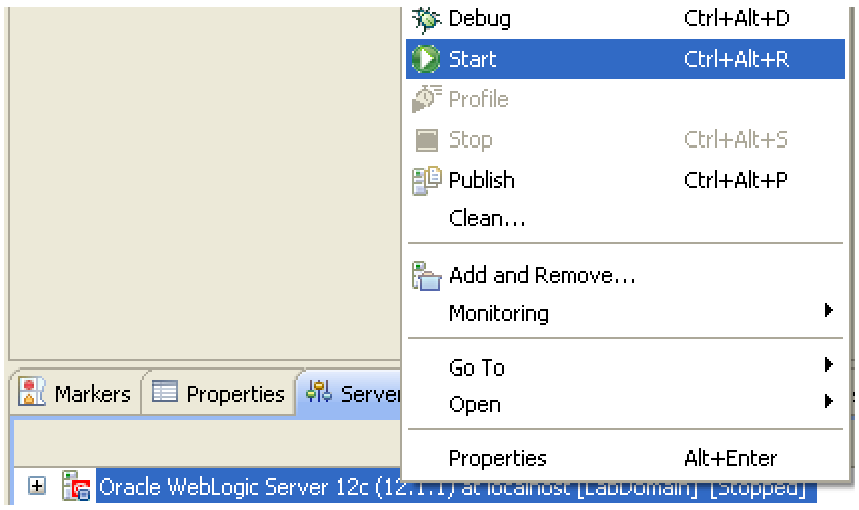


Oracle WebLogic Server 12c (12.1.1) at localhost [LabDomain] [Stopped]

Part 3 - Working with the Server

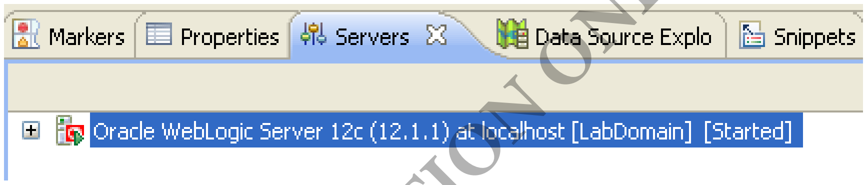
Throughout this class, we will be using this view to perform 3 basic operations: server *stop*, *start* and *publish*. We will discuss publishing in a later lab, so let us now focus on starting and stopping the server.

__1. Let us start things off by starting the server. In the *Servers* view, right click on the server and select **Start**.

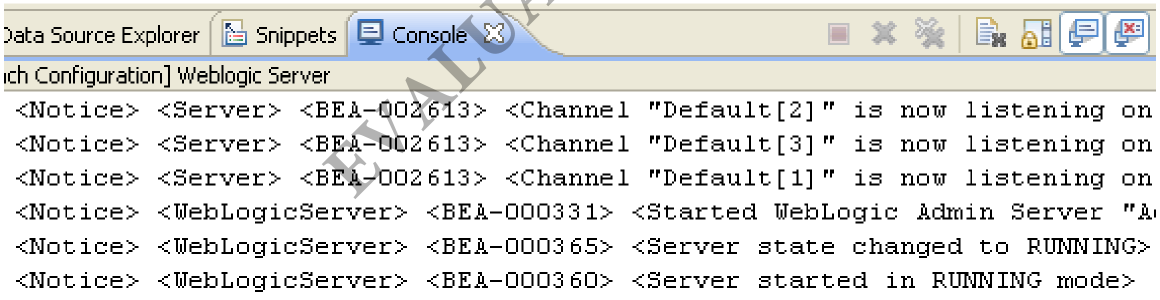


The *Console* view should automatically open. This will allow us to "see" OEPE booting. Eventually, the server will start and the *Servers* view will be brought to the focus again.

After finish publishing the server *State* should be **Started**.


Markers
Properties
Servers
Data Source Explo
Snippets
Oracle WebLogic Server 12c (12.1.1) at localhost [LabDomain] [Started]

__2. Examine the *Console* window by clicking the *Console* tab.

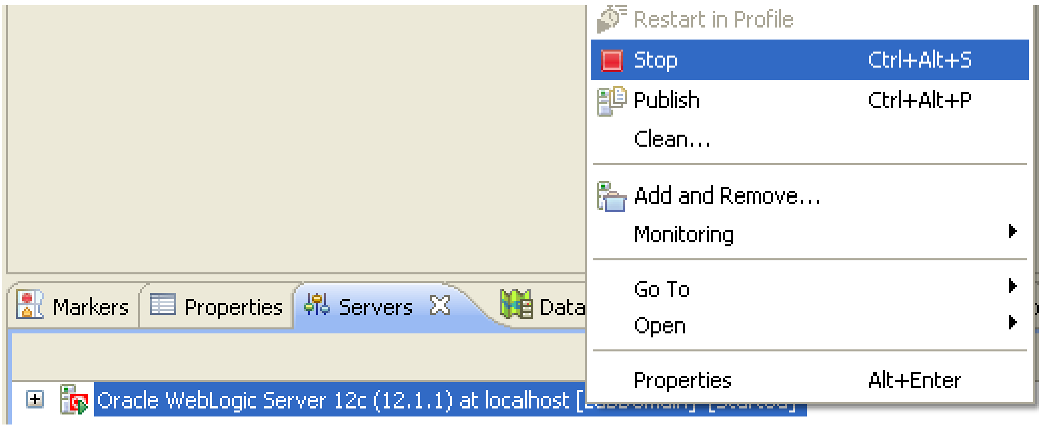


```
<Notice> <Server> <BEA-002613> <Channel "Default[2]" is now listening on
<Notice> <Server> <BEA-002613> <Channel "Default[3]" is now listening on
<Notice> <Server> <BEA-002613> <Channel "Default[1]" is now listening on
<Notice> <WebLogicServer> <BEA-000331> <Started WebLogic Admin Server "A
<Notice> <WebLogicServer> <BEA-000365> <Server state changed to RUNNING>
<Notice> <WebLogicServer> <BEA-000360> <Server started in RUNNING mode>
```

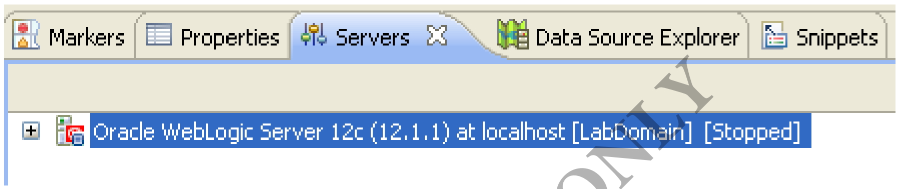
In later lab steps, when we ask you to refer to the *server console*, this is the view we will be referring to.

__3. Let us try stopping the server. Go back to the *Servers* view.

__4. Right click on the server and select **Stop**.



As with starting the server, the console view will open briefly. Then the *Server* view will appear, listing the status as *Stopped*.


Markers
Properties
Servers
Data Source Explorer
Snippets
Oracle WebLogic Server 12c (12.1.1) at localhost [LabDomain] [Stopped]

You have stopped the server.

Part 4 - Review

In this lab, you configured the development environment.

You started things off by configuring environment followed by WebLogic Server domain creation and server setup, which will be used to host our web service applications.

You saw how to start and stop a WebLogic Server instance.

Lab 2 - Designing a RESTful Service

Before we get into actual coding of RESTful services, we need to learn about how to design or architect one. This skill will be independent of the programming model and will apply equally if you choose to use PHP, ASP.NET or JAX-RS as the programming model.

We will begin by gathering business requirements. We will then analyze the requirements and then take various architectural decisions.

You will only need a pen and a notepad for this lab.

Part 1 - The Business Requirements

Acme Inc. is a machine parts manufacturing company. It manufactures components that other companies buy to build machines. Most of the parts are standard and are purchased by many companies. Acme also offers custom parts manufacturing services. A customer can submit the design for a part which Acme manufactures just for that client.

Currently, Acme offers a web site where companies can place orders and check order status. This has worked well for a few years. Now, customers are increasingly asking for a Web Service interface. This will help the clients integrate parts ordering with their own internal software systems and allow them to place orders without human intervention.

Short Theory of Requirements

Usually, requirements are first written using brief unstructured statements. They are captured as feature requests, with unique number and status. In this manner, features are very similar to defects. Features that are within the scope of a project are approved and then further elaborated using use cases. Use cases also have unique identification numbers. Use cases follow a more structured format and are immensely helpful in any software design.

To keep things simple in this lab, we will just write down the basic requirements. In real life, you are encouraged to develop use cases before proceeding with the subsequent stages of service design.

R1. Get price quote and availability – A client organization should be able to get a quote for a part and a confirmation if it can be available within a given date. The quote includes unit price and total applicable price taking into account any discounts.

R2. Place an order for standard parts – A client organization should be able to place an order. An order is a list of part numbers and quantities. After the order is placed successfully, an identifier for the order is returned.

R3. Get order details – Client should be able to get the status of an order by the order ID. The information includes the total cost of the order. If the order is pending shipment then an expected delivery date is also returned.

R4. Get order history – Client should able to get a list of orders by order status.

R5. Cancel an order – Client should able to cancel an order.

R6. Place order for a custom part – Client uploads a CAD (computer aided design) document that captures the design of the part.

Part 2 - Define Project Scope

In Phase I of the project, requirements R1-R5 will be addressed. Ordering of custom parts (R6) will be addressed in Phase II.

Part 3 - Designing the Object Model

This is the first stage in designing a RESTful web service. At this stage, we define the structure of the data that is exchanged between the web service and its consumers. The process is very similar to class design in Object Oriented Programming.

First step in object model definition is the identification of entities. Entities are nouns in business requirements.

__1. Can you go through the business requirements and identify the entities?

Here is the answer:

- Price quote request
- Price quote response
- Order
- Order line item
- Order status

__2. Now that you have identified the entities, can you list the attributes or fields for each entity? Once again, the answer is below. But do your best to do the design yourself.

Price quote request

- Part number
- Quantity
- Client organization ID
- Date the part needs to be available by

Price quote response

- Part number
- Quantity
- Date the part needs to be available by
- Unit price
- Total price

- If order can be shipped within the required date.

Order line item

- Part ID
- Quantity requested

Order

- Order ID
- Client organization ID.
- Order total
- Status
- Expected shipment date
- List of order line items

Part 4 - Identify the Service

Service identification is an important stage in Service Oriented Architecture (SOA). At this point we study the requirements and look for services that need to be implemented.

First, we look for the verbs in the requirements. They signify the actions or operations that the services need to be performed. Then, we group the logically related operations into services.

__1. Can you go through the requirements and list the tasks or operations that various yet to be identified services need to perform? Then compare your findings with the answer below.

Operations:

- Get price quote and availability (R1)
- Place an order for standard parts (R2)
- Get order details (R3)
- Get order history for a client organization (R4)
- Cancel an order (R5)

Note: In our case, each requirement (or use case) yields one operation. In more complex cases, there may be multiple operations identified from a use case.

Next, we need to group the operations in a logical grouping called a service. In REST, they key factor that affects grouping is entity. That means, operations that work on any given entity (like Order, Customer), should be grouped in a single service.

__2. Can you group the operations based on the entity they work with? Name each group as a service accordingly. The answer is below.

Quote Service:

- Get price quote and availability (R1)

Order Service:

- Place an order for standard parts (R2)
- Get order details (R3)
- Get order history for a client organization (R4)
- Cancel an order (R5)

Part 5 - Design the Service Interface

For SOAP based services, designing service interface involves writing the WSDL document and the associated XML schema files. For REST, the process involves assigning a URI structure and HTTP method for each identified operation.

In RESTful web service, a URI uniquely points to an entity. The HTTP method determines what action needs to be performed on that entity. For example, each order is identified by the /order/{orderId} URI. Hence, to get the status of an order, we do a GET on that URI. To place an order, we will do a PUT on the /order URI.

Each operation identified from the business requirement needs to be mapped to a URI and HTTP method pair.

__1. First, let us decide on the URI for each entity. This is a pretty straightforward exercise. **Note:** As a matter convention, plural forms of the entity names are used in URI.

1. Price quote - /quotes.
2. Order - /orders.

There is no need for URIs for the other entities since we do not directly perform any action on them.

These URIs will also form the root URI for the identified services. Individual operations within a service may add extra path elements. For example, to get the status of order, the URI will be /orders/{orderId}. Some operations may need to add URL parameters. This is discussed in more details later.

Now, we are ready to design the service interfaces by assigning a URI/HTTP method pair to each operation. Since you might be new to this, the work has been already done for you. Go through the design below and try to understand the rationale for various decisions taken.

Service: Quote Service **Root URI:** /quotes			
Operation	**URI extension**	**HTTP Method**	**Remarks**
Get price quote and availability (R1)		POST	Normally for such a read-only request, we should use GET. But, we need to send the quote request XML document to the service. This will be difficult to do for a GET request. That is why we choose POST.

Service: Order Service **Root URI:** /orders			
Operation	**URI extension**	**HTTP Method**	**Remarks**
Place an order for standard parts (R2)		PUT	Creation of an entity usually uses the PUT method. There is no need for an URI extension.
Get order details (R3)	/{orderId} E.g.: /orders/1051	GET	The order ID is now added to the root URI to point to a specific order.
Get order history for a client organization (R4)	?clientId={clientId}&status={status} E.g.: /orders?clientId=18&status=P	GET	The client organization ID and status are supplied as URL parameters. We could consider the syntax /order/{clientId}/{status} since that forms a valid identifier for a set of orders. But, that will confuse things with the /order/{orderId} syntax used for other actions.
Cancel an order (R5)	/{orderId}	DELETE	We choose to use the DELETE method to signify cancellation. This is bit of a tricky decision because the entity is not actually deleted.

Design Tip

According to REST methodology, the URI syntax is heavily entity centric, in the sense that a URI points to an entity or a set of entities that match certain criteria. Actual operations performed on the entity are not reflected in the URI. The HTTP methods go a long way to define the actions. But, even that may not be enough.

Consider the situation if we had two separate requirements to cancel and delete an order. In that case, the DELETE method will apply to the deletion case. How will we model cancellation? In situations like this, where HTTP method is not sufficient to model an operation, you need to resort to URL parameters. For example, /orders/{orderId}?action=cancel can be used.

In summary, HTTP methods are sufficient to model CRUD (create, retrieve, update and delete) operations. Anything beyond that, use URL parameters.

Part 6 - Decide on the Data Format

The two prevalent formats in the REST world are JSON and XML. XML is better for applications written in Java, .NET and PHP since they all come with XML parsers. JSON is the most convenient format to work with from JavaScript. JavaScript includes the DOM API. So, in the worst case, you can consume XML from JavaScript.

As stated in our business requirements, the web service will be consumed mainly by software applications of the client organizations. This will steer our decision to XML. In future, if the need arises, we can support JSON.

Part 7 - Unaddressed Architectural Issues

Our web service has to be secured. Otherwise, anyone can place an order or check the status. To keep things simple at this stage, we will not deal with the security requirements. We will get to that in a latter lab.

We do not deal with hypermedia controls either. We will learn about hypermedia later.

Part 8 - Review

Designing a RESTful web service requires a series of steps. We saw some of these steps in play here:

1. Design the object model.
2. Identify services
3. Design the service interface (URI structure and HTTP method for the operations).
4. Decide on the data format (XML or JSON).

Lab 3 - Develop a Simple RESTful Service

Over the next few labs, our goal will be to learn the fundamentals of the JAX-RS API. We will not get into implementing the services for Acme Inc. just yet.

In this lab, we will develop a very simple RESTful web service. This will let us focus on the development process using the OEPE tool and WebLogic server. We will also learn about the basic JAX-RS annotations.

The main goal of this lab is to understand how to create a web service project and configure it properly for JAX-RS development.

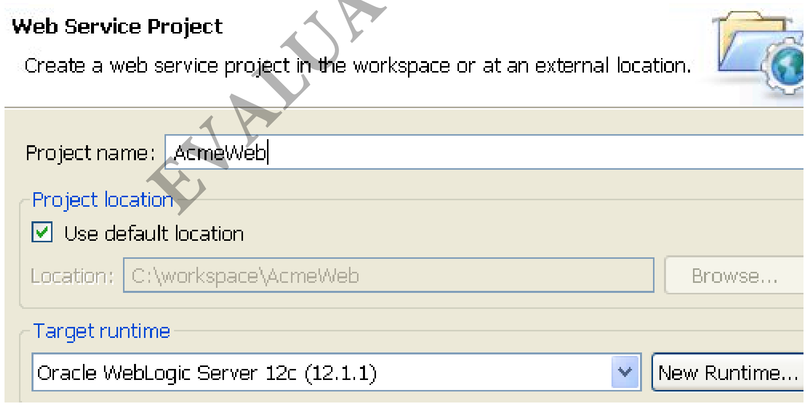
Part 1 - Create the Web Service Project

In OEPE, we can develop a JAX-RS web service inside of a dynamic web project or a web service project. The latter is just an extension of a dynamic web project with a few extra JAR files added to the compiler's build path. These extra JAR files include the Jersey 1.9 JAR files that implement the JAX-RS API. Hence, using the web service project will save us a bit of time. If you have a legacy dynamic web project where you will like to create a JAX-RS service, all you have to do is add the Jersey JAR files to the build path. We will look into that shortly.

__1. Launch OEPE if it is not already running.

__2. Switch to the **Java EE** perspective.

__3. From the menubar, select **File > New > Web Service Project**.



Web Service Project

Create a web service project in the workspace or at an external location.

Project name: AcmeWeb

Project location

☑ Use default location

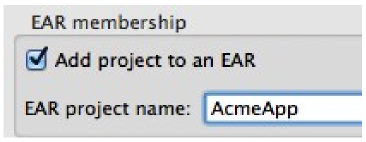
Location: C:\workspace\AcmeWeb

Target runtime

Oracle WebLogic Server 12c (12.1.1)

__4. Enter **AcmeWeb** as the project name.

__5. Ensure that **Oracle WebLogic Server 12c** as the targeted runtime.



EAR membership

☑ Add project to an EAR

EAR project name: AcmeApp

__6. Add the project to an enterprise application archive project called **AcmeApp**

Tip

You must add the web service project to a EAR. Otherwise, it appears, the JAX-RS services don't work.

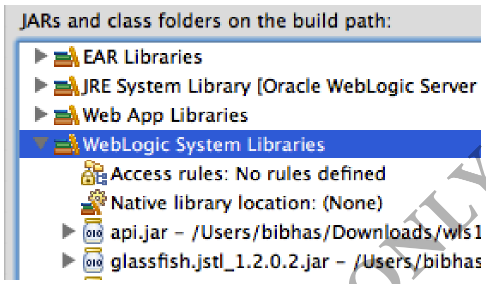
__7. Click **Finish**.

We will briefly review the project.

__8. Right click the newly created AcmeWeb project and select **Properties**.

__9. Select the **Java Build Path** properties.

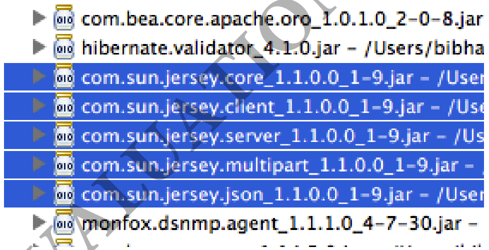
__10. Click the **Libraries** tab.



JARs and class folders on the build path:

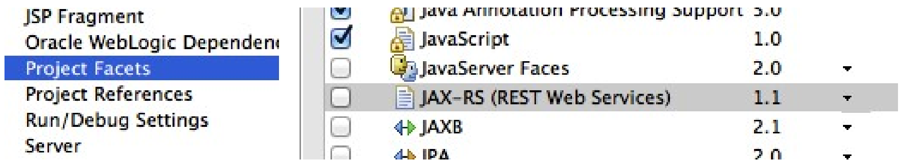
- EAR Libraries
- JRE System Library [Oracle WebLogic Server
- Web App Libraries
- WebLogic System Libraries
  - Access rules: No rules defined
  - Native library location: (None)
  - api.jar – /Users/bibhas/Downloads/wls1
  - glassfish.jstl_1.2.0.2.jar – /Users/bibhas

__11. Expand **WebLogic Systems Library**.



- com.bea.core.apache.oro_1.0.1.0_2-0-8.jar
- hibernate.validator_4.1.0.jar – /Users/bibha
- com.sun.jersey.core_1.1.0.0_1-9.jar – /User
- com.sun.jersey.client_1.1.0.0_1-9.jar – /Use
- com.sun.jersey.server_1.1.0.0_1-9.jar – /Us
- com.sun.jersey.multipart_1.1.0.0_1-9.jar –
- com.sun.jersey.json_1.1.0.0_1-9.jar – /User
- monfox.dsnmp.agent_1.1.1.0_4-7-30.jar –

__12. Scroll down and locate the **Jersey JAR** files. If you use a plain dynamic web project, you will need to manually add these JAR files to the build path.



JSP Fragment
Oracle WebLogic Dependen
Project Facets
Project References
Run/Debug Settings
Server

✓	Java Annotation Processing Support	5.0
✓	JavaScript	1.0
☐	JavaServer Faces	2.0
☐	JAX-RS (REST Web Services)	1.1
☐	JAXB	2.1
☐	JPA	2.0

__13. Now, select the **Project Facets** property.

__14. Note that the JAX-RS facet is not selected. This facet plays no role in developing JAX-RS services for WebLogic.

__15. Click **Cancel** to close the dialog.

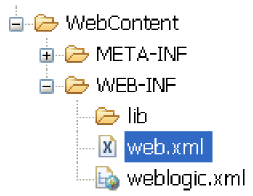
Part 2 - Register the REST Handler Servlet

A REST servlet handles all incoming request for RESTful services. In our case, this servlet is available from the Jersey implementation. All we have to do is register it in web.xml. To save time, this file is given to you. You will simply import it.

__1. Open Windows file explorer.

__2. Go to **C:\LabFiles\**

__3. Copy **web.xml**



__4. In Eclipse OEPE, paste the file inside the **WebContent > WEB-INF** folder of **AcmeWeb**.

__5. Open **web.xml** and study how the servlet is registered. Specifically, note the servlet mapping:

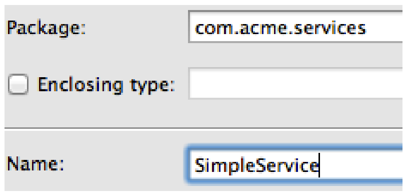
```
<servlet-mapping>
    <servlet-name>RestServlet</servlet-name>
    <url-pattern>/svc/*</url-pattern>
</servlet-mapping>
```

This means, the URL for every REST request will start with http://host:port/AcmeWeb/svc/. You can choose any other path for the REST servlet. But, we will stick to the short and sweet "svc".

Part 3 - Create the Resource Class

The Java class that implements a RESTful service is called a resource. We will now develop a Java class for a simple service.

__1. Right click **AcmeWeb** project and select **New > Class**.



Package: com.acme.services

Enclosing type:

Name: SimpleService

__2. Enter **com.acme.services** as the package name and **SimpleService** as the class name.

__3. Click **Finish**.

__4. Add a member variable that will help us do logging.

```
Logger logger = Logger.getLogger("SimpleService");
```

We will now add the testGET() method. It will not have any business logic. Later, we will map this method to a GET request.

__5. Add the method as follows.

```
public String testGET() {
          logger.info("Got a GET request");

          return "OK";
}
```

__6. Organize imports (Control+Shift+O) and select **java.util.logging.Logger**.

__7. Save changes.

Part 4 - Configure the Resource

We will now configure the URI and HTTP method for the service resource and its methods. We will use the "/simple" root URI for the service.

__1. Define the URI of the root resource, by adding the @Path annotation above the class. This is shown in bold face below.

```
@Path("/simple")
public class SimpleService {
```

The testGET() method does not need any path extension. All we have to do is set GET as the HTTP method.

We will also set "text/plain" as the content MIME type of the reply. That is good enough for this method. For XML data type, the MIME will be "text/xml".

__2. Set the HTTP method and reply MIME type for the testGET() method sub-resource as follows.

```
@GET
@Produces("text/plain")
public String testGET() {
```

__3. Organize imports. Select **javax.ws.rs.Produces**.

__4. Save changes.

Part 5 - Unit Test

We will now exercise the web service from a browser.

__1. Start the server if it is not running.

__2. Right click the server and select **Add and Remove**.

__3. Add the **AcmeApp** project to the server.

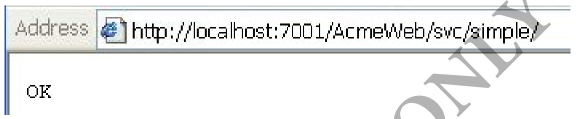
__4. Click **Finish**.

__5. Make sure the server is synchronized.

__6. Open a web browser and enter the URL:

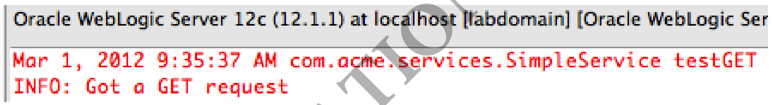
```
http://localhost:7001/AcmeWeb/svc/simple/
```

__7. You should see OK in the browser.



```
OK
```

__8. The Console view in Eclipse will show the log output.


Oracle WebLogic Server 12c (12.1.1) at localhost [labdomain] [Oracle WebLogic Ser

```
Mar 1, 2012 9:35:37 AM com.acme.services.SimpleService testGET
INFO: Got a GET request
```

This proves that our project has been setup correctly and JAX-RS is working fine.

Part 6 - Use Path Extension for Sub-resource

A sub-resource – Java method – can be mapped to an URI extension path. Any HTTP request for that path will be handled by that method. We will now create a method that will respond to the /simple/**mypath** URI.

__1. First, add this method.

```
public String testGETWithPath() {
          logger.info("Got a GET request with path extension.");

          return "OK";
}
```

__2. Annotate the method as follows.

```
@GET
@Produces("text/plain")
@Path("/mypath")
public String testGETWithPath() {
```

__3. Save changes.

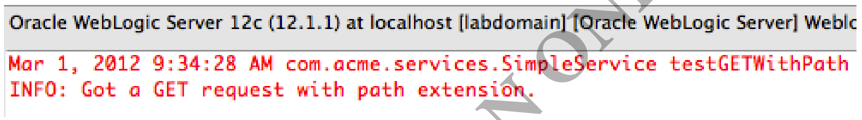
Part 7 - Unit Test

__1. Right click the server and select **Publish**.

__2. In a web browser, enter the URL:

```
http://localhost:7001/AcmeWeb/svc/simple/mypath
```

__3. Make sure that the Console shows the log output.



```
Oracle WebLogic Server 12c (12.1.1) at localhost [labdomain] [Oracle WebLogic Server] Weblc
Mar 1, 2012 9:34:28 AM com.acme.services.SimpleService testGETWithPath
INFO: Got a GET request with path extension.
```

__4. Open a new browser and re-test the testGET method for regression using the URL:

```
http://localhost:7001/AcmeWeb/svc/simple/
```

__5. Close all open files.

__6. Close all browsers.

Part 8 - Review

In this lab, we created and configured a web service project. We created a very simple JAX-RS web service. At this point, you should know how to configure the URI of the root resource – the Java class - using the @Path annotation. Also, we configured the HTTP method of a sub-resource – a method – using the @GET annotation.