ORACLE

Oracle Workshop for WebLogic 10g R3 Hands on Labs

Workshop for WebLogic extends Eclipse and Web Tools Platform for development of Web Services, Java, JavaEE, Object Relational Mapping, Spring, Beehive, and Web Applications. It installs as a plug-in to your existing Eclipse, or will install Eclipse for you.

Workshop for WebLogic is used to develop, build, assemble, deploy, debug and test Java SE, Java EE, Web Services, Java Web Applications, Object Relational Mapping on Eclipse. If you are an Oracle WebLogic Server user, this is the free tool for you.

10g R3 Release Notes

Workshop for WebLogic introduces new tools in support of Java EE 5.0 standards. The support for Java EE5 includes the following technologies:

Java EE5 Standards Support

- Servlet 2.5
- JSP 2.1
- JSF 1.2
- JSTL 1.2
- Unified Expression Language
- JAX-WS
- JAXB 2.0
- EJB 3
    - EE5 EAR
    - EJB 3 Session Beans
    - EJB 3 Message Driven Beans

Built on Eclipse 3.3.2 and Web Tools Platform 2.0.3

Workshop for WebLogic 10g R3 is built on the Eclipse Platform, an open source framework that is now widely used for Java development. Workshop extends **Eclipse 3.3.2** and the Web Tools Platform **2.0.3**

Supported by Windows Vista

Workshop for WebLogic 10g R3 is supported by **Windows Vista**.

XML Beans

Workshop for WebLogic 10g R3 supports **XMLBeans 2.3**.

Workshop for WebLogic IDE Launcher

The **WorkSpace Studio launcher** has been discontinued. The Workshop for WebLogic IDE launcher is WORKSHOP_HOME/workshop[.exe].

ORACLE

Getting Familiar with Workshop for WebLogic 10g R3

The core components of this Eclipse based development environment are defined by the following functional areas:

- Enhanced server plug-ins for multiple versions of Oracle WebLogic Server
- Visual Oracle WebLogic Server Web Service and XML IDE
- WYSIWYG Web and presentation tier tools for portable Java Web applications
- Object-relational mapping workbench and database tools
- Apache Beehive IDE for Java Page Flow and controls
- AppXRay support for the above components
- Spring IDE Project and Spring code generation wizards
- Core IDE features for Java SE and Java EE
- Built in Web Application and Web Service test client
- Upgrade tools for Workshop 8.1, 9.2 and 10 users

Getting hands on with Workshop for WebLogic 10g R3

This document contains instructions for the following labs:

1. Creation of a simple JAX-WS web service
2. Using the Workshop ClientGen functionality
3. JAX-WS Web Service with custom bindings
4. JAXB
5. FastSwap Java Class Redefinition

LAB1: JAX-WS Service

Note:
The lab resources and completed projects can be found in the companion 10gR3Labs.zip file. The folder names match the lab titles in this document.

Objective: Create a simple JAX-WS web service in WLW and deploy it to a running server.

This release of WebLogic Server supports both Java API for XML-Based Web Services 2.1 (JAX-WS) and Java API for XML-Based RPC 1.1 (JAX-RPC) Web Services. JAX-RPC, an older specification, defined APIs and conventions for supporting XML Web Services in the Java Platform as well support for the WS-I Basic Profile 1.0 to improve interoperability between JAX-RPC implementations. JAX-WS is a follow up to JAX-RPC 1.1 and it implements many of the new features in Java EE 5. For additional documentation and examples about programming the features described in the following sections in a JAX-WS Web Service, see the JAX-WS documentation available at https://jax-ws.dev.java.net.

Java API for XML-based Web Services (JAX-WS) 2.1 is supported in this release, adding the following features to those found in JAX-WS 2.0:

- Support for the JAXB 2.1 (JSR 222) Data Binding API
- WS-Addressing support
- Dynamic publishing of endpoints

- APIs for EndpointReference creation and propagation
- Annotations and APIs to enabled/disable features, such as MTOM and Addressing

The WebLogic Server implementation of JAX-WS is based on the JAX-WS Reference Implementation (RI), Version 2.1.4, and includes enhancements to the tool layer to simplify the building and deployment of JAX-WS services and to ease the migration from JAX-RPC to JAX-WS. The following features and enhancements are available from the JAX-WS RI 2.1.4:

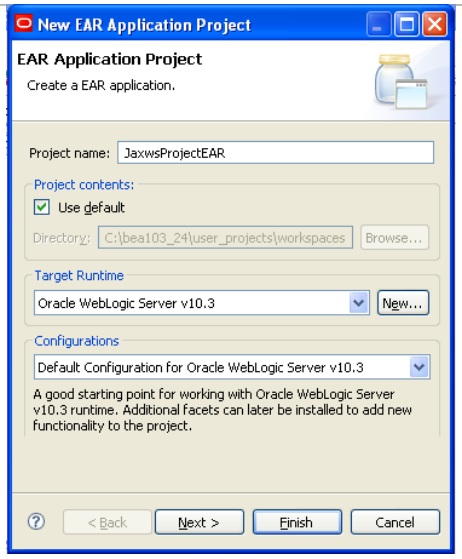
- .NET 2.0/WSF 3.0 MTOM interoperability support
- Significant performance improvements through the use of Woodstox StAX Parser
- SOAPAction- based dispatching
- Integration of JAXB RI 2.1.5
- JAXB type substitution support
- WS-Addressing support for both W3C (1.0) and Member Submission (2004/08)
- Asynchronous client/server support
- Dispatch and provider support:
  - Dispatch<Message> and Provider<Message> support
  - Development of non-WSDL or non-SOAP endpoints, such as REST

As with WebLogic Server 10.0, developers may begin development with either a Java source file or WSDL file. The WebLogic Server Ant tasks <jwsc> and <clientgen> automate the generation of portable data binding classes, creation of deployment descriptors, and packaging.

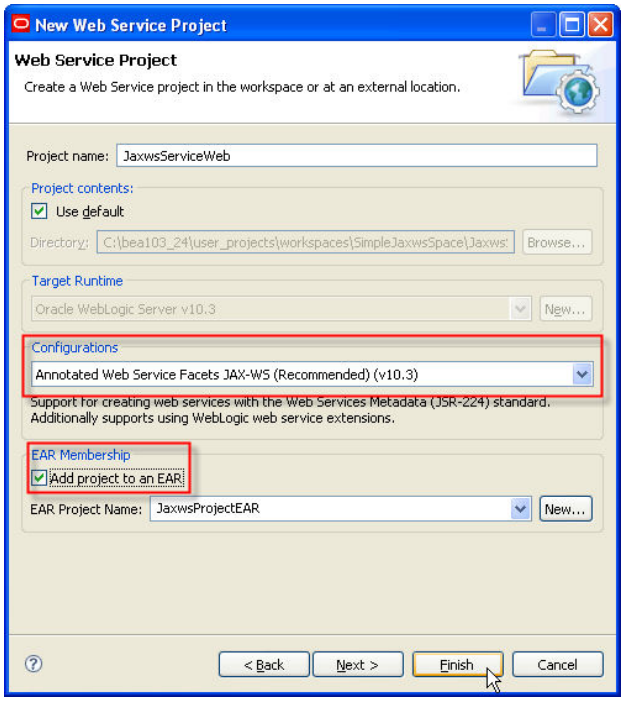
Steps:

Note: The JAX-WS Service and ClientGen projects are combined in the JaxWsService.zip archive. The Service is 'JaxwsServiceWeb' and the client is 'JaxwsClientWeb'.

1. Open Workspace and create a new space called 'SimpleJaxwsSpace'
2. Select File -> New -> Project -> J2EE
3. Create a new Enterprise Application called 'JaxwsProjectEAR'
4. Keep the default configuration
5. Click 'Finish'



6. Select File -> New -> Project
7. Find 'Web Service Project' under 'Web Services'. Select and click 'Next'
8. Name the project 'JaxwsServiceWeb'
9. Under 'Configurations' select 'Annotated Web Services Facets JAX-WS (Recommended) (v10.3)'
10. Select 'Add project to an EAR'
11. Click 'Finish'

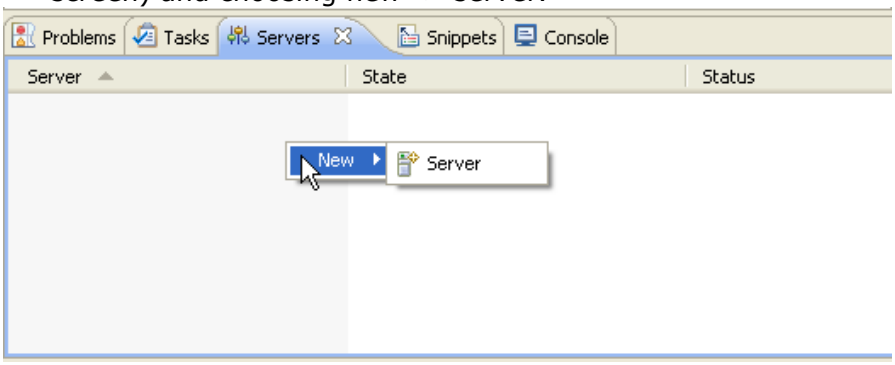


12. In the 'Project Explorer' View right click the 'Java Resources/src' folder in the JaxwsServiceWeb project and select New -> package.
13. Name the package 'webservice'
14. Right click the 'webservice' package folder
15. Select New -> WebLogic Web Service
16. Enter the class name 'HelloJaxWs' and click 'Finish'
17. You'll get the source with the '@WebService' annotation

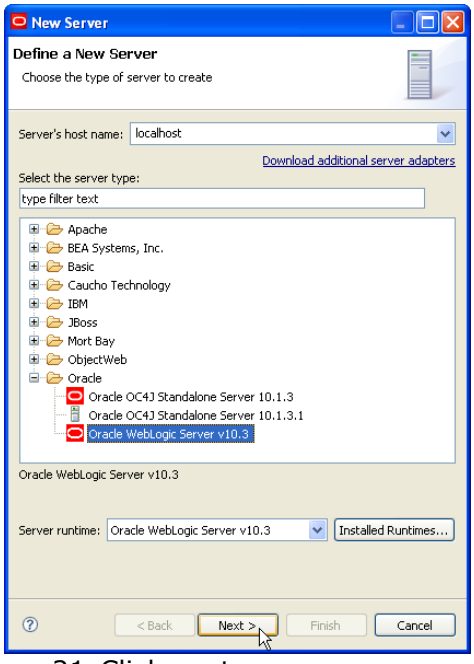
18. Fill out the 'hello' method with the following:

```
@WebMethod
public String hello(String name) {
        System.out.println("Service invoked with name: " + name);
        return "Hello " + name + " from JAX-WS service";
    }
```

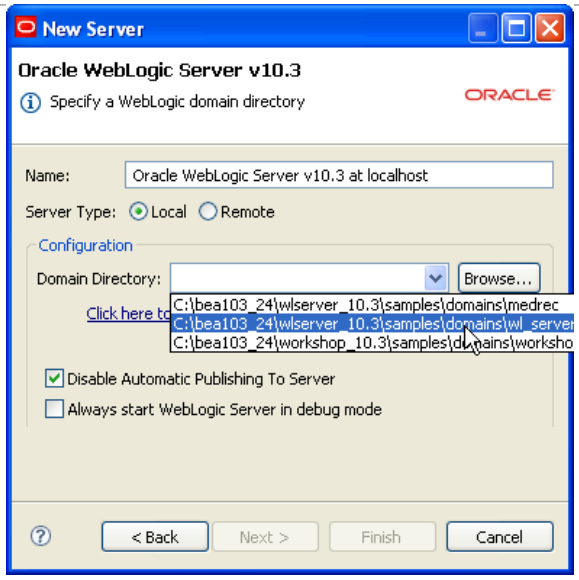
19. Create a server by right clicking in the servers tab (towards the bottom of the screen) and choosing new -> server.



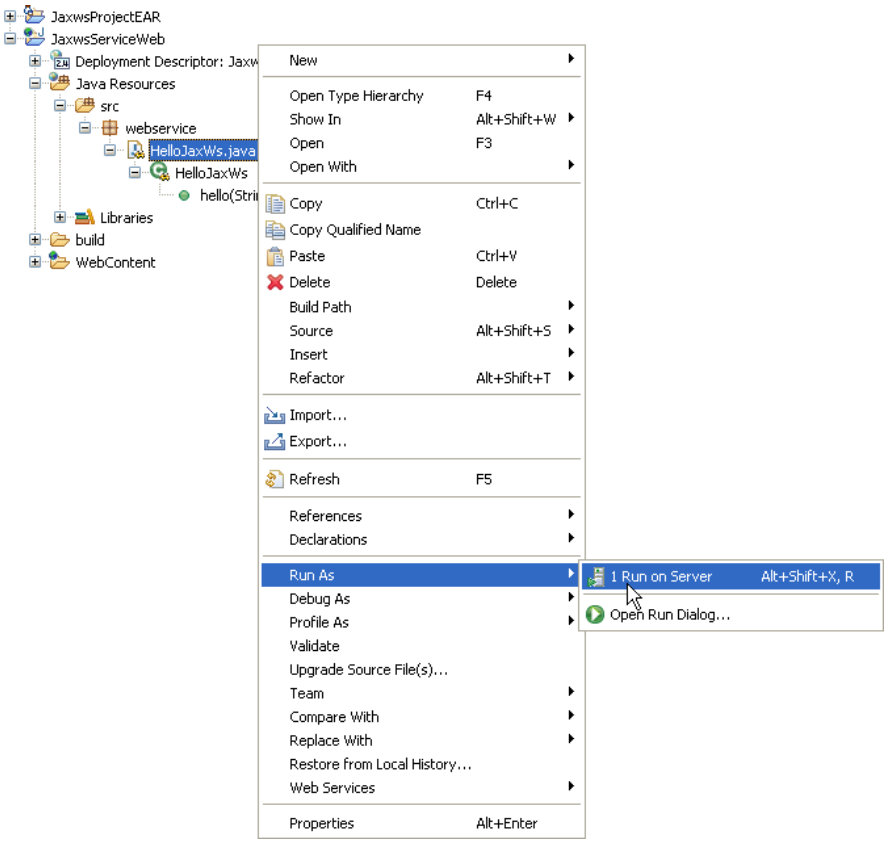
20. Choose the Oracle WebLogic Server 10.3 (default)



21. Click next.
22. When creating the server, you can use the sample domain that ships with the product by pointing to the directory <bea_home>\wlserver_10.3\samples\domains\wl_server. Choose the domain and then click finish.

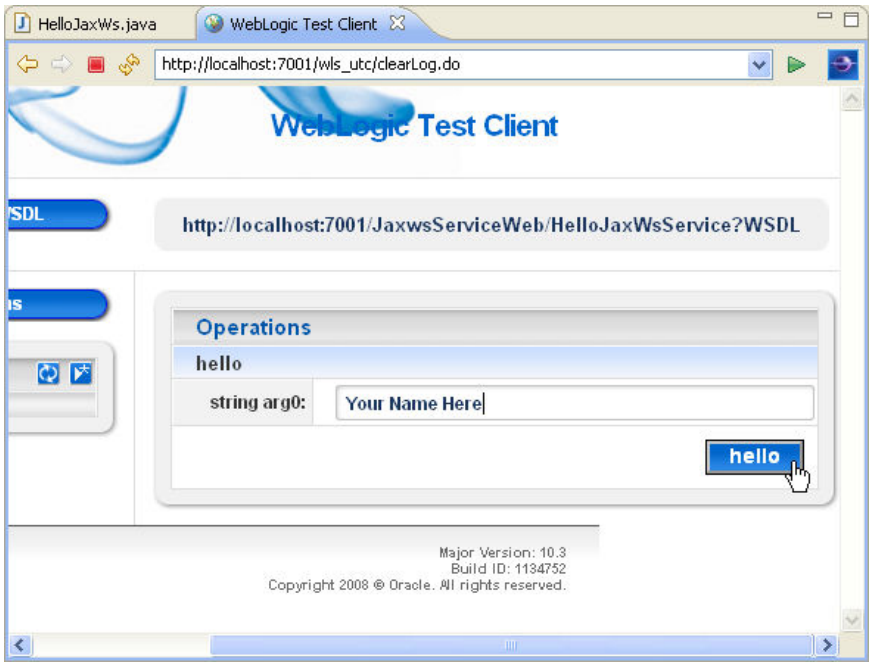


23. Right click on the HelloJaxWs.java as shown below, and choose run as.. run on server to build and deploy and test the application onto Oracle WebLogic Server 10gR3. Click finish on the resulting dialog to run.



Troubleshooting Tip: If you have an issue deploying the app where the error says that someone else owns the edit lock on the server go to the WLS console -> Preferences and disable the automatic locking. Then click 'Release Edit Lock' to fix it.

24. The test client should launch in the embedded Eclipse web browser. This test harness allows you to interact with your JAX-WS web service. Enter your name and click the "Hello" button, then scroll down to see the response.



25. Leave Workshop open, server running and application deployed, as you will need it in the next lab.

LAB2: Client Gen – depends on LAB1

Note:
The lab resources and completed projects can be found in the companion 10gR3Labs.zip file, located on the OTN. The folder name for this lab is the same as lab 1: "JAX-WS Service".

Objective: Create a client for the previously created JAX-WS web service using the ClientGen functionality of WLW. This should be done in sequence after lab1 to work properly.

The clientgen Ant task generates, from an existing WSDL file, the client component files that client applications use to invoke both WebLogic and non-WebLogic Web Services. This allows your web service to be called from a web or non-web client.
When generating a **JAX-WS** Web Services, the output includes:

- The Java class for the Service interface implementation for the particular Web Service you want to invoke.
- JAXB data binding artifacts.
- The Java class for any user-defined XML Schema data types included in the WSDL file.

When generating a **JAX-RPC** Web Services, the output includes:

- The Java class for the Stub and Service interface implementations for the particular Web Service you want to invoke.

- The Java source code for any user-defined XML Schema data types included in the WSDL file.
- The JAX-RPC mapping deployment descriptor file which contains information about the mapping between the Java user-defined data types and their corresponding XML Schema types in the WSDL file.
- A client-side copy of the WSDL file.

Client applications can use the generated artifacts of clientgen to invoke Web Services, for example with:

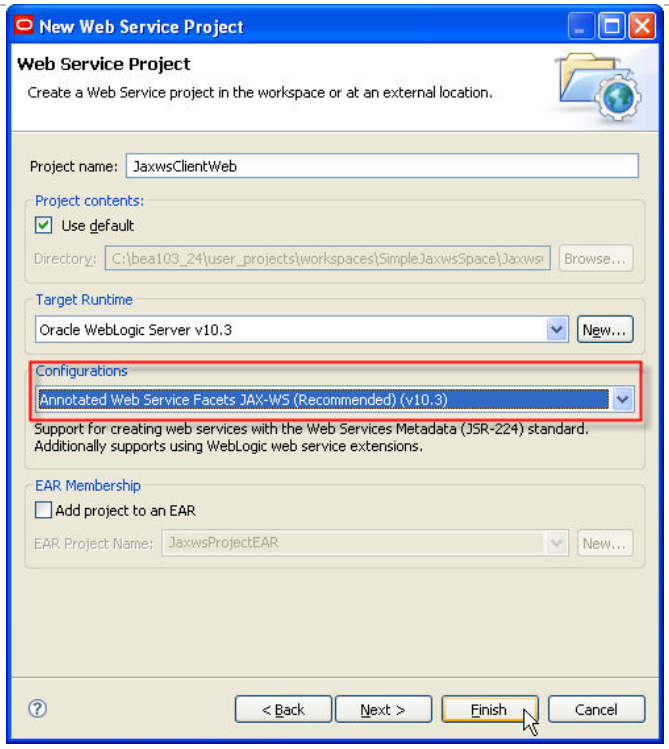
- Stand-alone Java clients that do not use the Java Platform, Enterprise Edition (Java EE) Version 5 client container.
- Java EE clients, such as EJBs, JSPs, and Web Services, that use the Java EE client container.

Client Gen's output libraries can also be used or declared as a Shared Java EE library on WebLogic. Java EE library support in WebLogic Server provides an easy way to share one or more Java EE modules or JAR files among multiple Enterprise Applications. A Java EE library is a stand-alone Java EE module, multiple Java EE modules packaged in an Enterprise Application (EAR), or a plain JAR file that is registered with the Java EE application container upon deployment. After a Java EE library has been registered, you can deploy Enterprise Applications that reference the library. Each referencing application receives a copy of the shared Java EE library module(s) on deployment, and can use those modules as if they were packaged as part of the application itself.

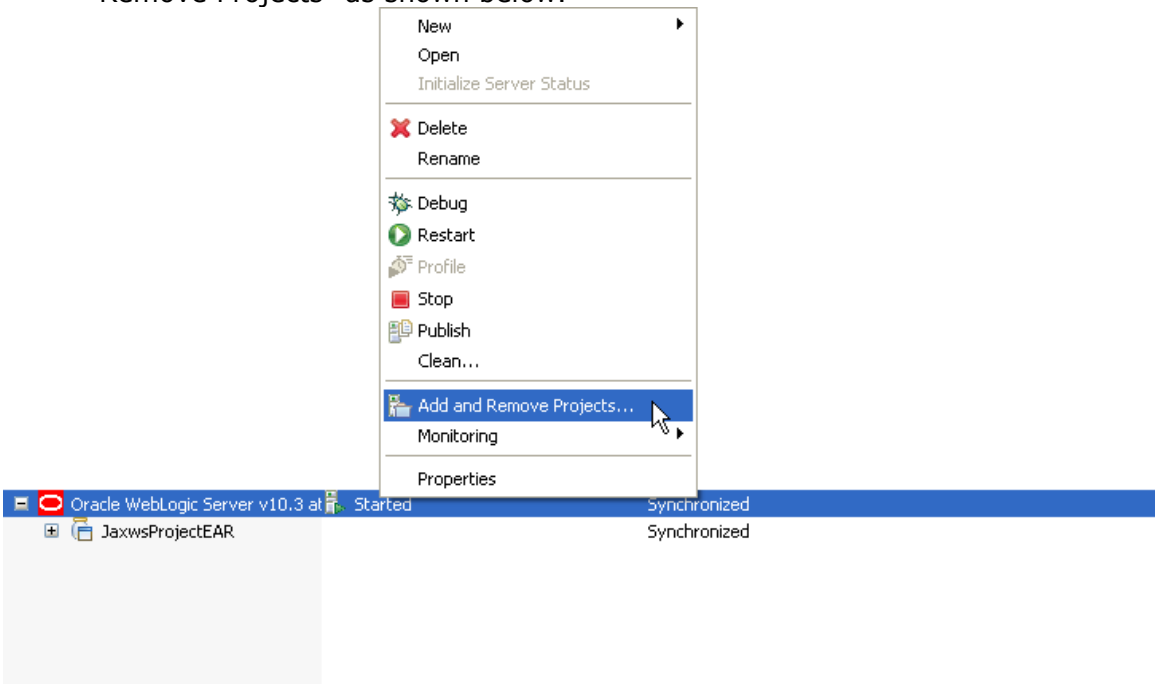
Steps:

Note: The JAX-WS Service and ClientGen projects are combined in the JaxWsService.zip archive. The Service is 'JaxwsServiceWeb' and the client is 'JaxwsClientWeb'.

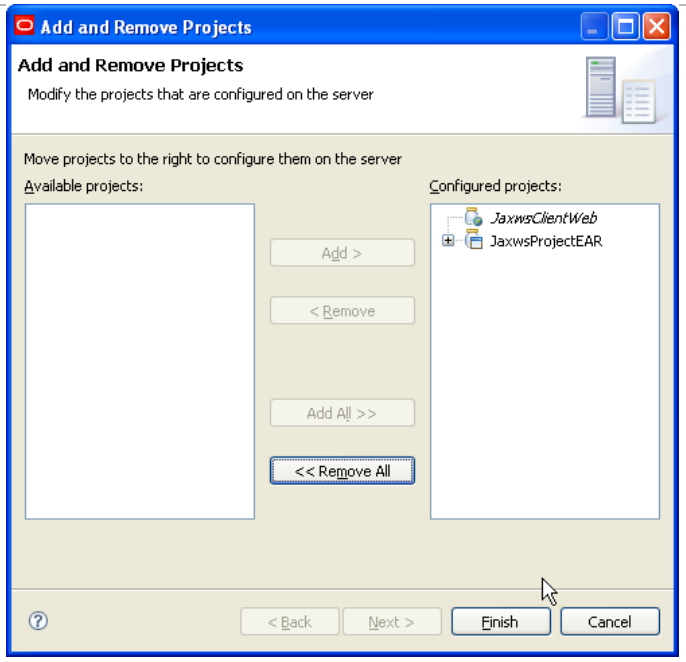
1. Select File -> New -> Project -> Web Services
2. Select 'Web Service Project'
3. Click 'Next'
4. Name the project 'JaxwsClientWeb' **and under 'Configurations' select 'Annotated Web Services Facets JAX-WS (Recommended) (v10.3)' as shown below:**



5. Click 'Finish'
6. Now deploy your new project to server that's already running. In the server tab, right click "Oracle WebLogic Server v10.3 …." and choose "Add and Remove Projects" as shown below.



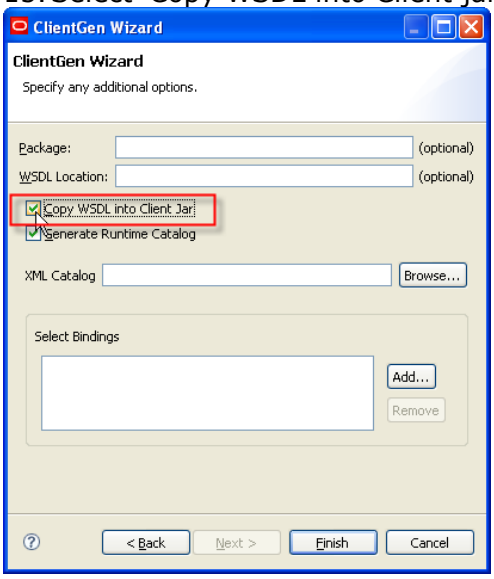
7. Click next on the resulting dialog.
8. Add the project by clicking add all, then finish. Do not remove existing projects.



9. Right click the 'JaxwsClientWeb/WebContent' folder
10. Select New -> Other -> Web Service -> ClientGen Web Service Client
11. Select the 'Remote' radio button and enter the following URL:
    http://localhost:7001/JaxwsServiceWeb/HelloJaxWsService?WSDL
12. Click 'Validate WSDL'

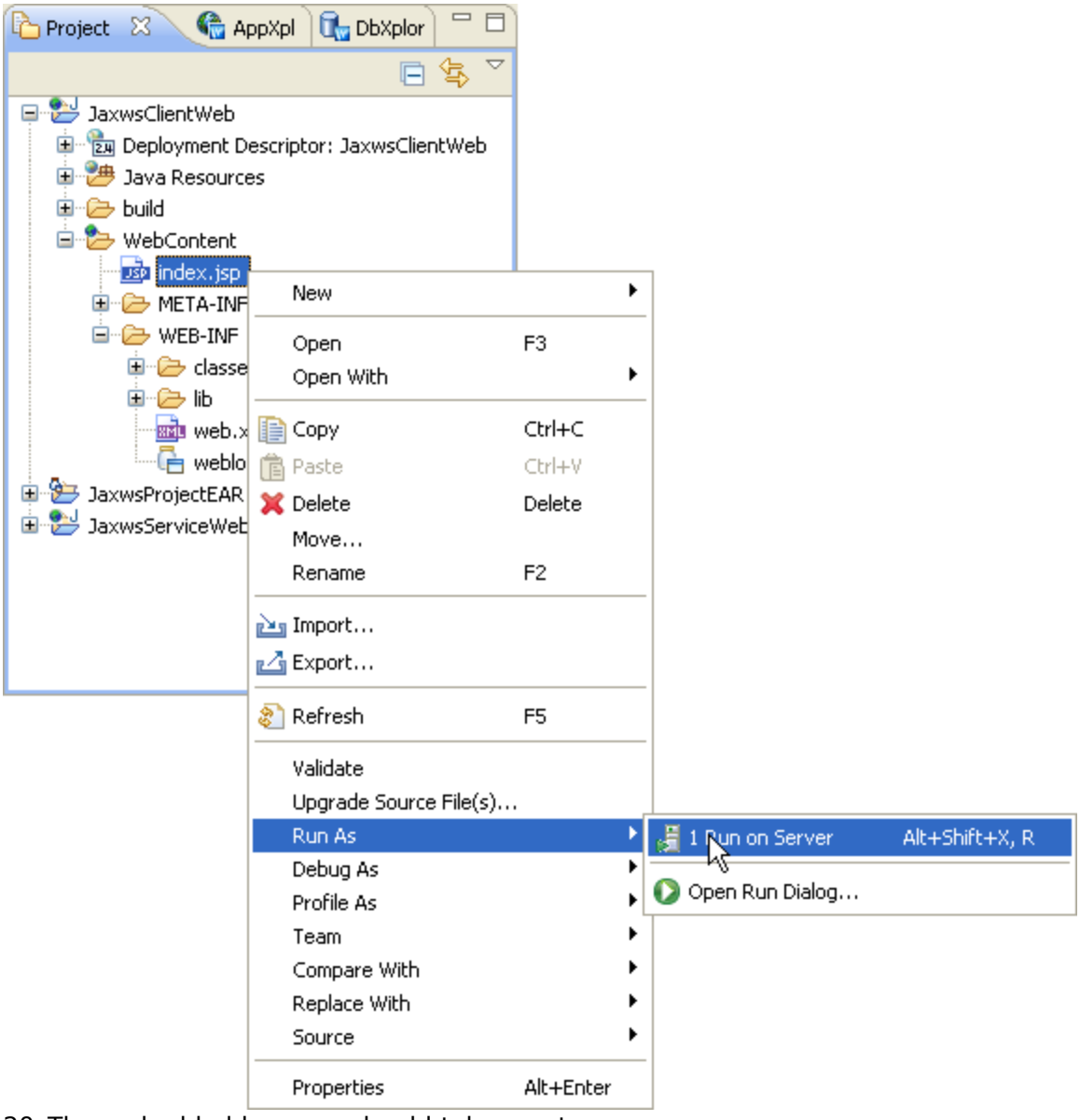
TROUBLESHOOTING TIP:
Make sure that the server is started and running, with the 'JaxwsServiceWeb' project deployed, and that you created the projects with the JAX-WS configuration, not the JAX-RPC configuration, in the previous steps.

13. Click 'Next'
14. Click "Next'
15. Select 'Copy WSDL into Client jar'



16. Click 'Finish'
17. You'll find the HelloJaxWsService.jar file in JaxwsClientWeb/WebContent/WEB-INF/lib
18. Copy index.jsp from the lab resources to the JaxwsClientWeb/WebContent folder

19. Right click on the index.jsp file and choose "Run As.. Run on Server to and deploy it to the server and open it in the embedded Eclipse browser. Click ok on the resulting dialog.



20. The embedded browser should take you to http://localhost:7001/JaxwsClientWeb/index.jsp

The next lab will make use of a client JAR in the user interface code.

LAB3: Custom Bindings

Objective: Creating web service that leverages JAX-B custom bindings, generating a client JAR, plugging client code into JSP and running it.

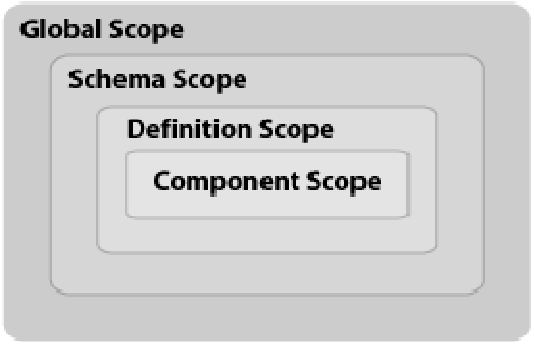
Introduction to customizing XML Schema-to-Java Mapping Using Binding Declarations

Due to the distributed nature of a WSDL, you cannot always control or change its contents to meet the requirements of your application. For example, the WSDL may not be owned by you or it may already be in use by your partners, making changes impractical or impossible. If directly editing the WSDL is not an option, you can customize how the WSDL components are mapped to Java objects by specifying custom *binding declarations*. You can use binding declarations to control specific features, as well, such as asynchrony, wrapper style, and so on, and to control the JAXB data binding artifacts that are produced by customizing the XML Schema.
You can define binding declarations in one of the following ways:

- Create an external binding declarations file that contains all binding declarations for a specific WSDL or XML Schema document.
- **Note:** If customizations are required, Oracle recommends this method to maintain flexibility by keeping the customizations separate from the WSDL or XML Schema document.
- Embed binding declarations within the WSDL or XML Schema document.

The binding declarations are semantically equivalent regardless of which method you choose. Custom binding declarations are associated with a scope, as shown in the following figure.

Scopes for Custom Binding Declarations



The following table describes the meaning of each scope.

Scopes for Custom Binding Declarations	
Scope	**Definition**
Global scope	Describes customization values with global scope. Specifically: ▪ For JAX-WS binding declarations, describes customization values that are defined as part of the root element.

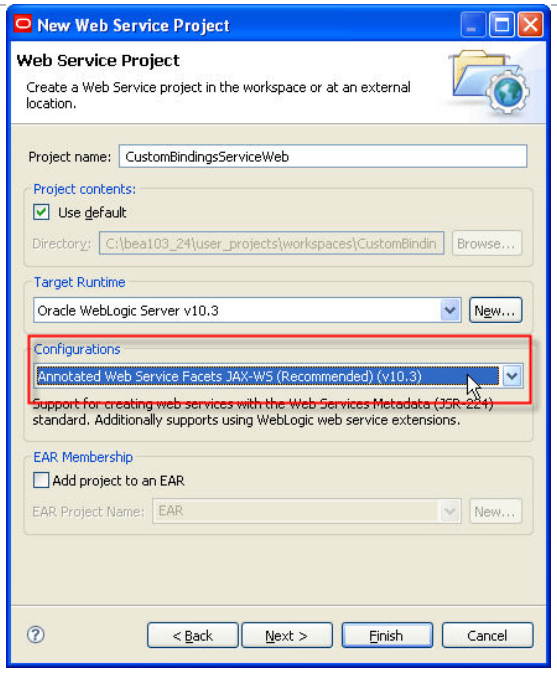
	▪ For JAXB annotations, describes customization values that are contained within the <globalBindings> binding declaration. Global scope values apply to all of the schema elements in the source schema as well as any schemas that are included or imported.
Schema scope	Describes JAXB customization values that are contained within the <schemaBindings> binding declaration. Schema scope values apply to the elements in the target namespace of a schema. **Note:** This scope applies for JAXB binding declarations only
Definition scope	Describes JAXB customization values that are defined in binding declarations of a type definition or global declaration. Definition scope values apply to elements that reference the type definition or global declaration. **Note:** This scope applies for JAXB binding declarations only
Component scope	Describes customization values that apply to the WSDL or schema element that was annotated.

Scopes for custom binding declarations adhere to the following inheritance and overriding rules:

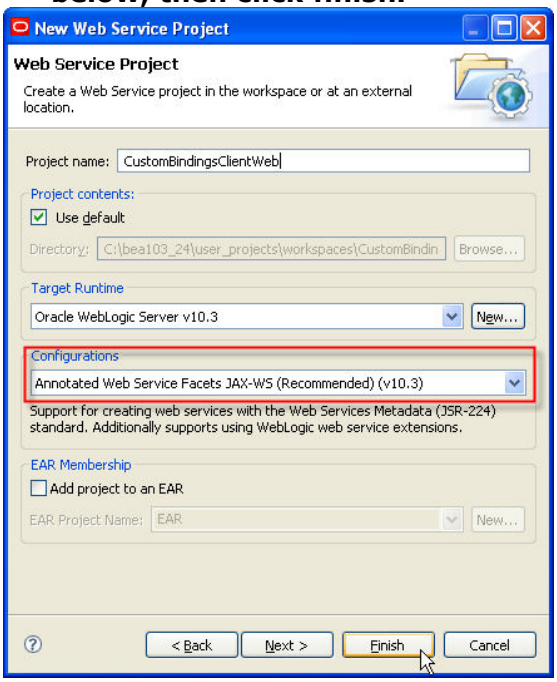
- Inheritance—Customization values are inherited from the top down. For example, a WSDL element (JAX-WS) in a component scope inherits a customization value defined in global scope. A schema element (JAXB) in a component scope inherits a customization value defined in global, schema, and definition scopes.
- Overriding—Customization values are overridden from the bottom up. For example, a WSDL element (JAX-WS) in a component scope overrides a customization value defined in global scope. A schema element (JAXB) in a component scope overrides a customization value defined in definition, schema, and global scopes.

Steps:

1. Create a new workspace called 'CustomBindingsSpace'
2. Select File -> New -> Project -> Web Services
3. Select 'Web Service Project'
4. Click 'Next'
5. Name the project 'CustomBindingsServiceWeb'
6. **Be sure to select 'Annotated Web Service Facets JAX-WS (Recommended)(v10.3)' in the 'Configurations' drop down, as shown below, then click finish.**

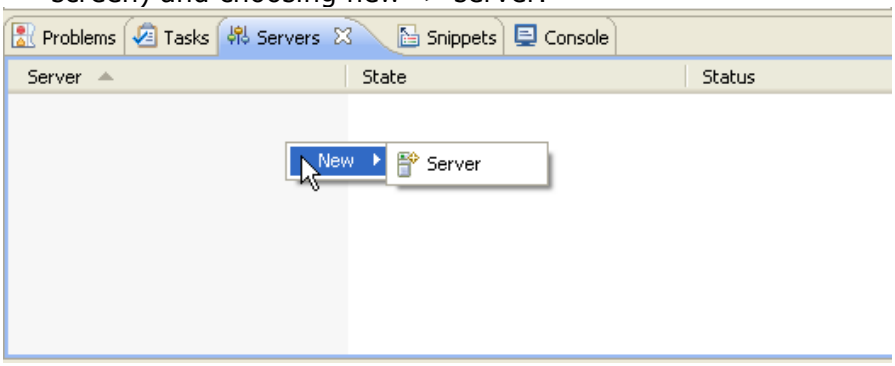


7. Select File -> New -> Project -> Web Services
8. Select 'Web Service Project'
9. Click 'Next'
10. Name the project 'CustomBindingsClientWeb'
11. **Be sure to select 'Annotated Web Service Facets JAX-WS (Recommended)(v10.3)' in the 'Configurations' drop down, as shown below, then click finish.**

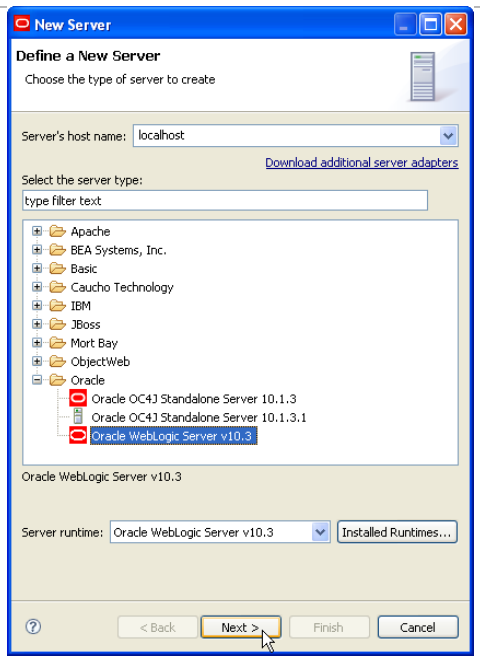


12. Do this next part carefully, making sure the paths are correct
13. In the CustomBindingsServiceWeb project create a folder called 'wsdl' under 'WebContent' and copy the file 'TemperatureService.wsdl' in 'CustomBindingsServiceWeb'.

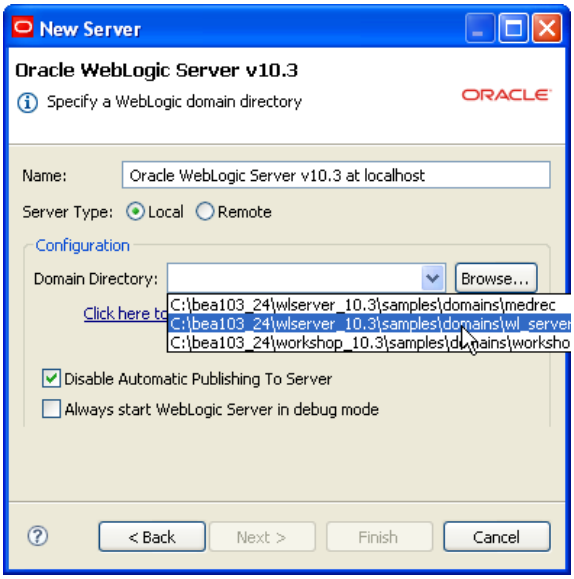
14. Copy the file 'myBindings.xml' to CustomBindingsServiceWeb/WebContent/wsdl.
15. In the CustomBindingsClientWeb project create a folder called 'bindings' under 'WebContent'. NOTE THIS IS NOT THE SAME PROJECT.
16. Copy 'myBindingsClient.xml' to the 'bindings' folder in CustomBindingsClientWeb
17. Go back to CustomBindingsServiceWeb/WebContent/wsdl, right click on the wsdl file and select Web Services -> Generate Web Service using WSDLC.
18. In the first screen delete the value from the 'Java package' field.
19. Click 'Next'
20. In the 'JAXB Bindings for WSDLC' select 'Add', navigate to 'WebContent/wsdl' and choose 'myBindings.xml'
21. Click 'Finish'
22. This will create the web service files in 'CustomBindingsServiceWeb /WebContent/WEB-INF/lib/TemperatureService_wsdl.jar'. If you want, you can open the file with winzip and you'll see the custom packaging reflected in the folder naming.
23. Create a server by right clicking in the servers tab (towards the bottom of the screen) and choosing new -> server.



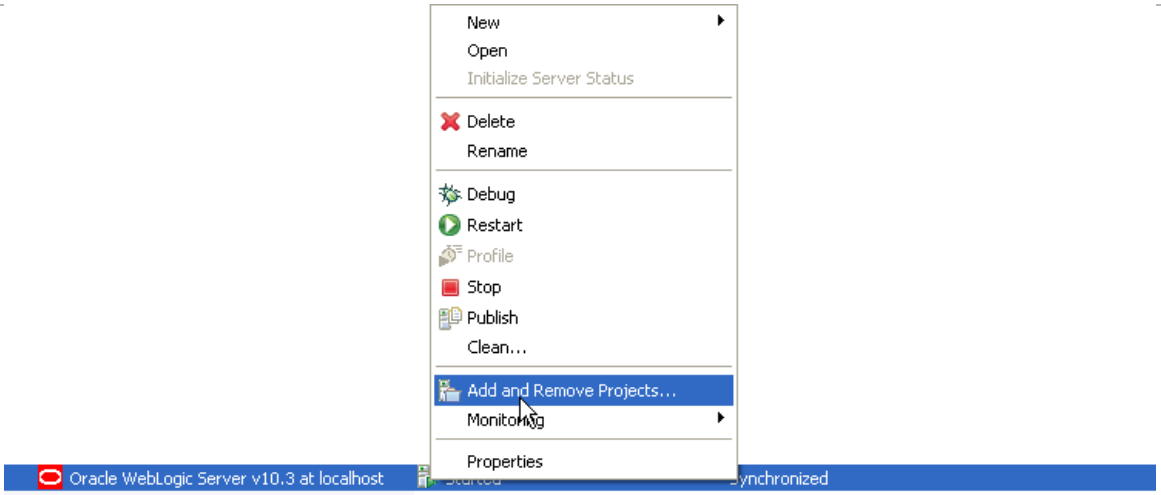
24. Choose the Oracle WebLogic Server 10.3 (default)



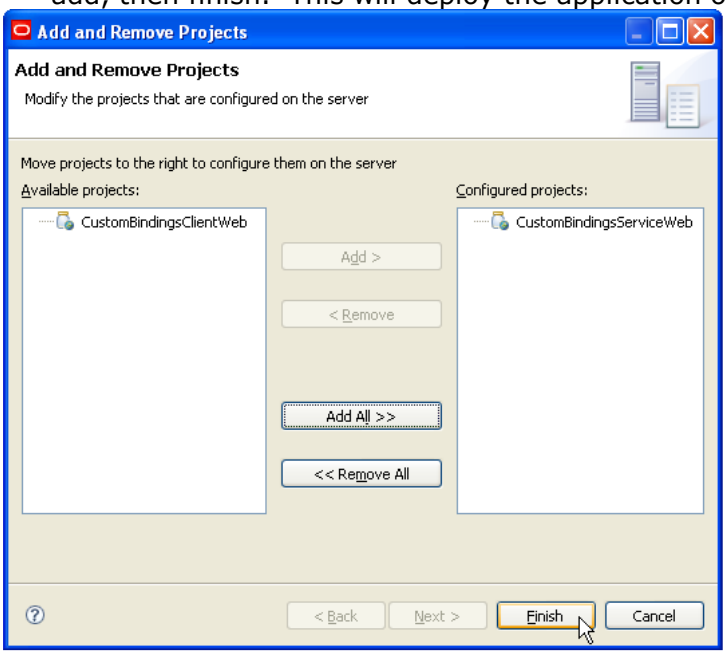
25. Click next.
26. When creating the server, you can use the sample domain that ships with the product by pointing to the directory <bea_home>\wlserver_10.3\samples\domains\wl_server. Choose the domain and then click finish.



27. Now deploy the CustomBindingsServiceWeb project to server that's already running. In the servers tab, right click "Oracle WebLogic Server v10.3 …." and choose "Add and Remove Projects" as shown below.
28. Click next on the resulting dialog.



29. Add the project by clicking on the CustomBindingsServiceWeb and clicking add, then finish. This will deploy the application on the server.



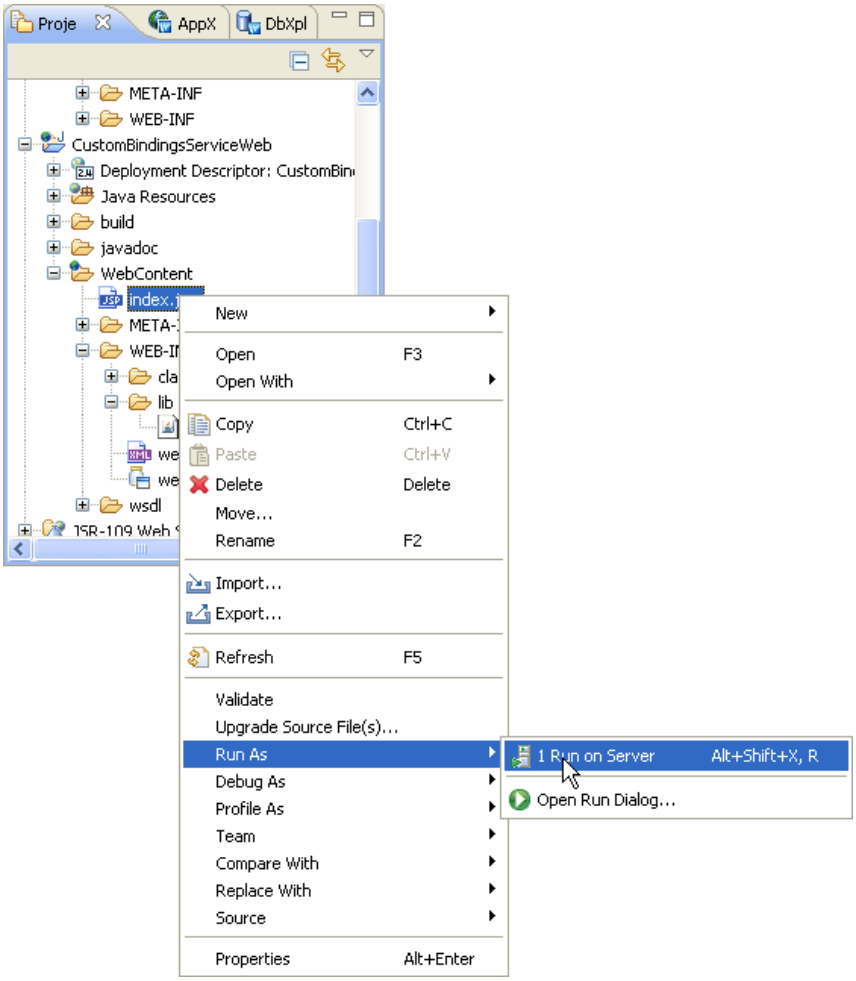
30. Start the server by clicking the green play button, in the servers tab.

31. Go to the 'CustomBindingsClientWeb' project
32. Right click the 'WebContent' folder and select New -> Other
33. Select Web Services -> ClientGen Web Service Client
34. Select the 'Remote' radio button and enter the wsdl location: http://localhost:7001/CustomBindingsServiceWeb/TemperatureService?WSDL

35. Click 'Validate WSDL'

Troubleshooting tip: make sure the CustomBindingsServiceWeb has successfully deployed to a running server, and that you created the projects with the JAX-WS configuration, not the JAX-RPC configuration, in the previous steps.

36. Click 'Next'
37. Click 'Next'
38. In the 'Bindings' section select 'Add'. Navigate to 'WebContent\bindings\myBindingsClient.xml' and add it.
39. Click 'Finish'
40. You now have 'TemperatureService.jar' in WEB-INF/lib. Open it and view the package structure.
41. Copy 'index.jsp' from the lab resources to CustomBindingsServiceWeb /WebContent
42. Right click on the JSP page (as shown below), choose run as.. run on server, and the embedded Eclipse browser should take you to http://localhost:7001/CustomBindingsClientWeb/index.jsp



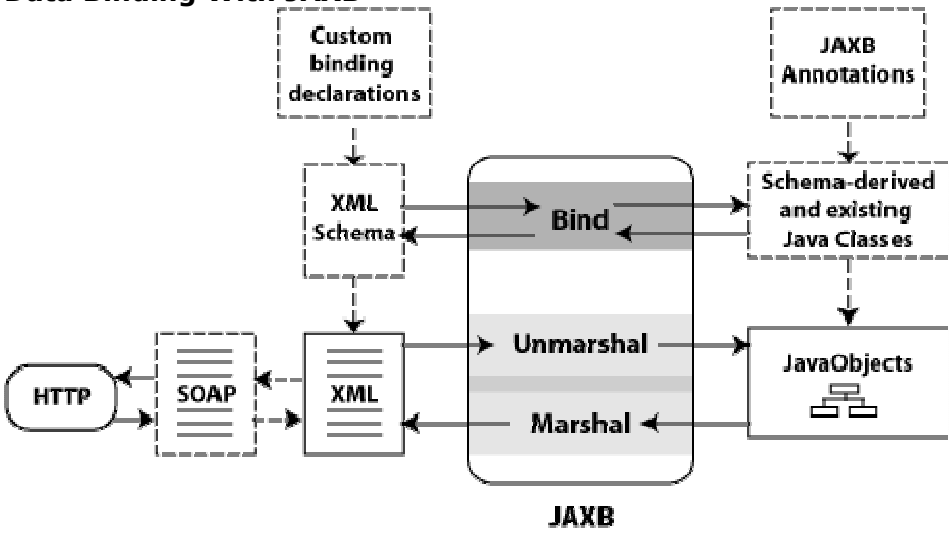
LAB4: JAXB

Objective: Demonstrate the JAXB functionality in Workshop by creating JAXB classes from an xsd and using them in a simple application.

Overview of Data Binding Using JAXB

With the emergence of XML as the standard for exchanging data across disparate systems, Web Service applications need a way to access data that are in XML format directly from the Java application. Specifically, the XML content needs to be converted to a format that is readable by the Java application. *Data binding* describes the conversion of data between its XML and Java representations. JAX-WS uses Java Architecture for XML Binding (JAXB) to manage all of the data binding tasks. Specifically, JAXB binds Java method signatures and WSDL messages and operations and allows you to customize the mapping while automatically handling the runtime conversion. This makes it easy for you to incorporate XML data and processing functions in applications based on Java technology without having to know much about XML. The following figure shows the JAXB data binding process.

Data Binding With JAXB



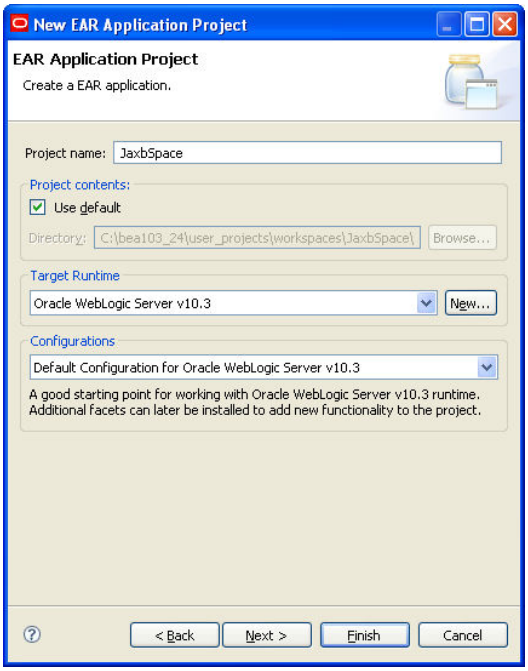
As shown in the previous figure, the JAXB data binding process consists of the following tasks:

- **Bind**—Binds XML Schema to *schema-derived JAXB Java classes*, or value classes. Each class provides access to the content via a set of JavaBean-style access methods (that is, get and set). Binding is managed by the JAXB *schema compiler*.
- **Unmarshal**—Converts the XML document to create a tree of Java program elements, or objects, that represents the content and organization of the document that can be accessed by your Java code. In the content tree, complex types are mapped to value classes. Attribute declarations or elements with simple types are mapped to properties or fields within the value class and you can access the values for them using get and set methods. Unmarshalling is managed by the JAXB *binding framework*.
- **Marshal**—Converts the Java objects back to XML content. In this case, the Java methods that are deployed as WSDL operations determine

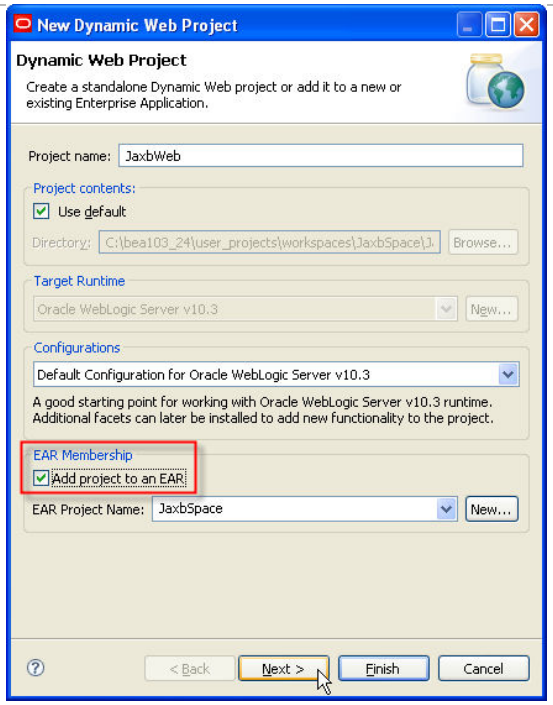
the schema components in the wsdl:types section. Marshalling is managed by the JAXB binding framework. You can use the JAXB binding language to define custom binding declarations or specify JAXB annotations to control the conversion of data between XML and Java.

Steps:

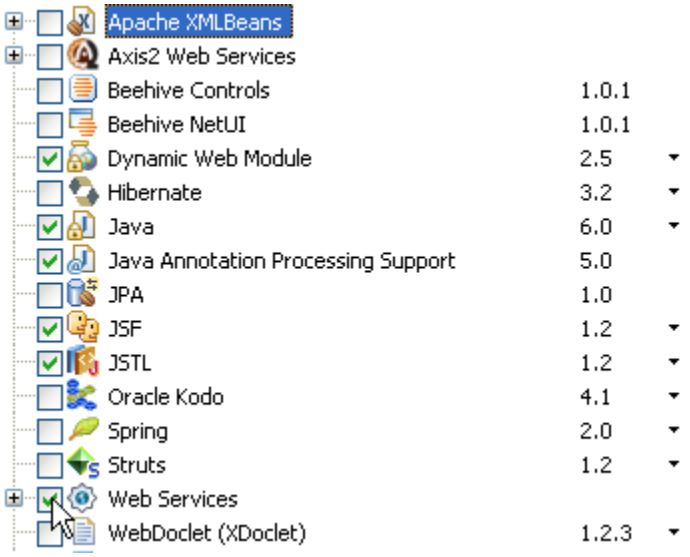
1. Create a new workspace called 'JaxbSpace'
2. Select File -> New -> Project -> J2EE
3. Create a new Enterprise Application called 'JaxbApp'
4. Keep the default configuration
5. Click 'Finish'



6. Select File -> New -> Project -> Web -> Dynamic Web Project
7. Create a Dynamic Web Project called 'JaxbWeb'
8. Add project to the EAR and click 'Next', as shown below

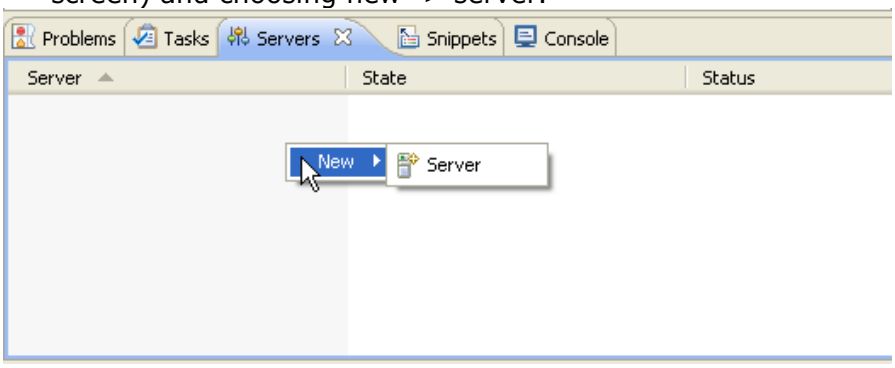


9. Be sure to select the Web Services facet and click 'Finish', as shown below

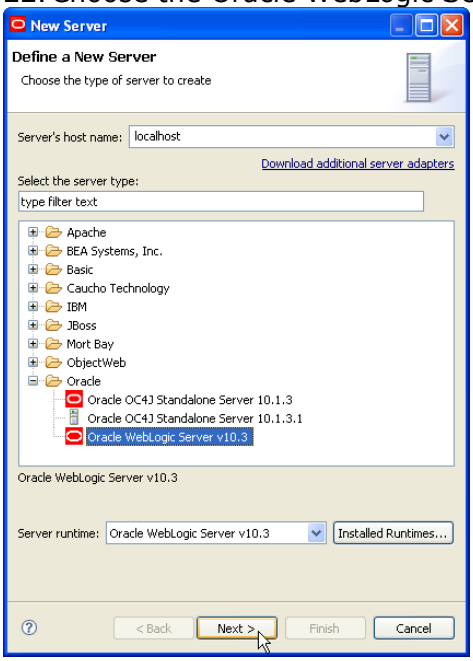


10. Create the folder JaxbWeb/WebContent/xsd
11. Copy the book.xsd file from the lab resources to JaxbWeb/WebContent/xsd
12. Copy the books.xml file to the domain root of the server, located at <bea_home>\wlserver_10.3\samples\domains\wl_server
13. Right click on book.xsd and select Web Services -> Generate a JAXB type JAR
14. Click 'Next'
15. Enter 'book' as the package
16. Click 'Finish'
17. The jar file book.jar is created in JaxbWeb/WebContent/WEB-INF/lib. If you like, you can double click on it to view the contents in WinZip in a file explorer window.
18. Create a package 'book' under JaxbWeb/Java Resources/src

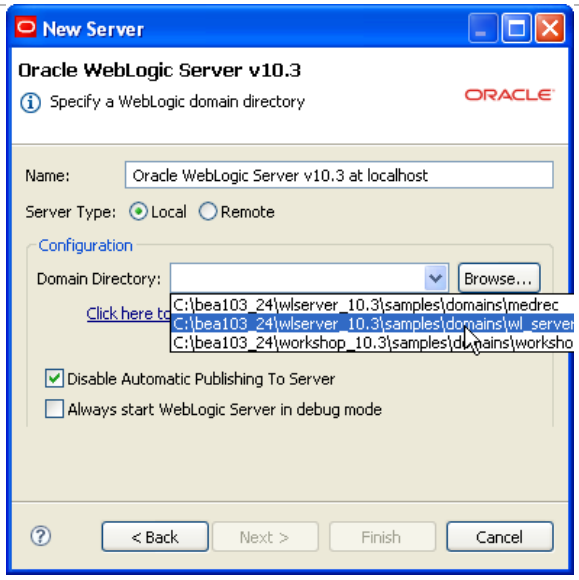
19. Copy in the JaxbTest source file from the lab resources into the 'book' package
20. Copy in the index.jsp from the lab resources to WebContent/JaxbWeb /WebContent, say yes to overwrite and replace the default generated one.
21. Create a server by right clicking in the servers tab (towards the bottom of the screen) and choosing new -> server.



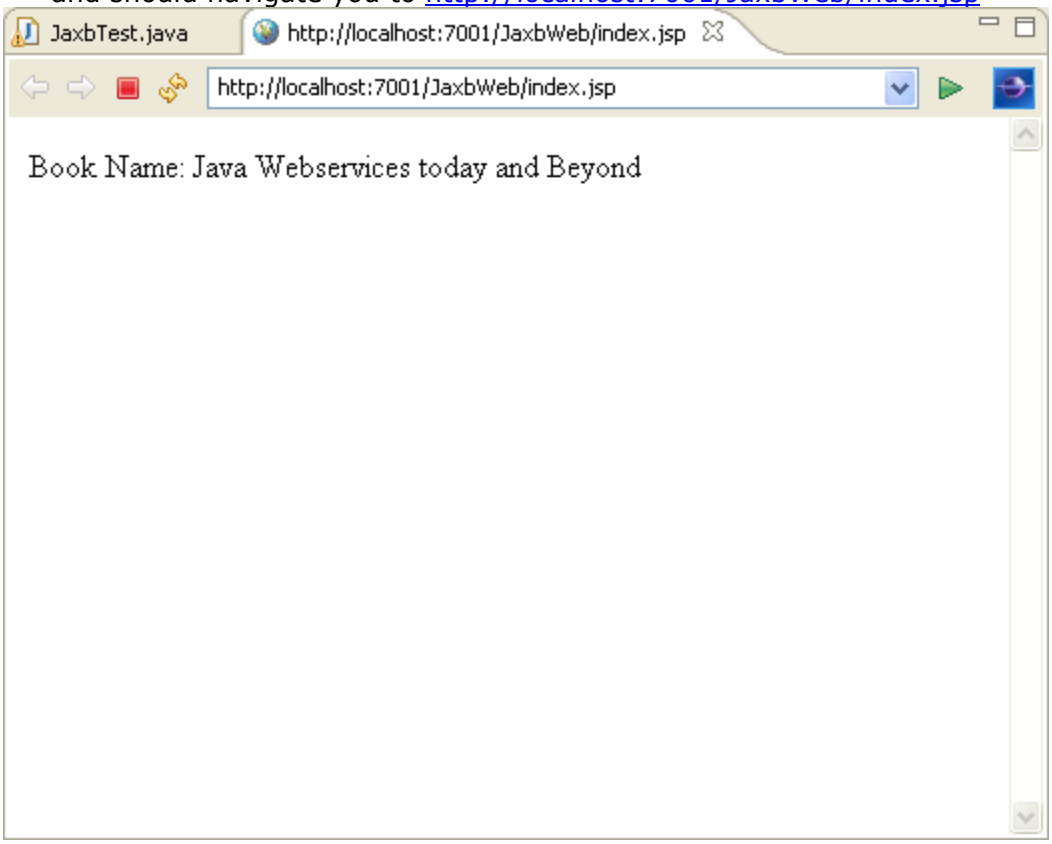
22. Choose the Oracle WebLogic Server 10.3 (default)

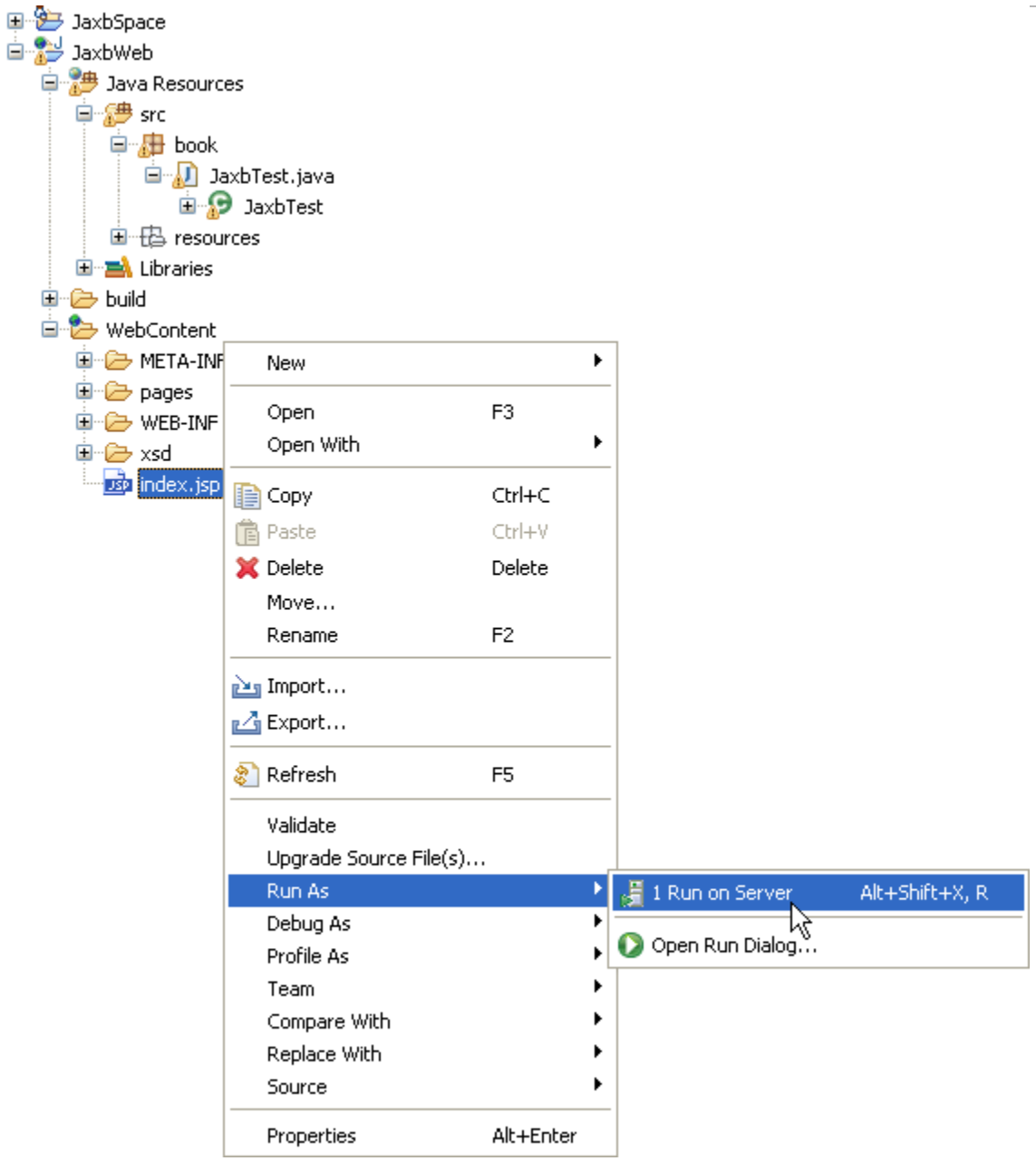


23. Click next.
24. When creating the server, you can use the sample domain that ships with the product by pointing to the directory <bea_home>\wlserver_10.3\samples\domains\wl_server. Choose the domain and then click finish, as shown below.



25. Right click on the JaxbWeb/WebContent/index.jsp (as shown below) and choose run as.. run on server. This will open the embedded eclipse browser and should navigate you to http://localhost:7001/JaxbWeb/index.jsp





26. The resulting webpage calls the JaxbTest class for some output (see below). Examine the JaxbTest.java for details.

LAB5: Class Redefinition (FastSwap)

Objective: Demonstrate the WLS FastSwap capability from within a Workshop project using a stateful session bean.

Java SE 5 introduces the ability to redefine a class at runtime without dropping its ClassLoader or abandoning existing instances. This will allow containers to reload altered classes without disturbing running applications, vastly speeding up iterative development cycles and improving the overall development and testing experiences. The usefulness of Java SE 5's dynamic class redefinition is severely curtailed,

however, by the restriction that the shape of the class – its declared fields and methods – cannot change. The purpose of FastSwap is to remove this restriction in WebLogic Server, allowing the dynamic redefinition of classes with new shapes to facilitate iterative development.

With FastSwap, Java classes are redefined in-place without reloading the ClassLoader thus having the huge advantage of fast turnaround times. This means that developers do not have to sit and wait for an application to redeploy and then navigate back to wherever they were in the Web page flow. They can make their changes, auto compile, and then see the effects immediately.

Supported Application Configurations

• FastSwap is only supported when the server is running in development mode. It is automatically disabled in production mode.
• Only changes to class files in exploded directories are supported. Modifications to class-files in archived applications as well as archived jars appearing in the application's classpath are not supported. Examples are as follows:
    • When a web application is deployed as an archived war within an ear, modifications to any of the classes are not picked up by the FastSwap agent.
    • Within an exploded web application, modifications to Java classes are only supported in the WEB-INF/classes directory; the FastSwap agent does not pick up changes to archived jars residing in WEB-INF/lib.

Application Types and Changes Supported with FastSwap

FastSwap is supported with POJOs (JARs), Web applications (WARs) and enterprise applications (EARs) deployed in an exploded format. FastSwap is not supported with resource adapters (RARs).
The following types of changes are supported with FastSwap:

- Addition of static methods.
- Removal of static methods.
- Addition of instance methods.
- Removal of instance methods.
- Changes to static method bodies.
- Changes to instance method bodies.
- Addition of static fields.
- Removal of static fields.
- Addition of instance fields.
- Removal of instance fields.

The online documentation has a detailed table listing the change types supported with FastSwap.

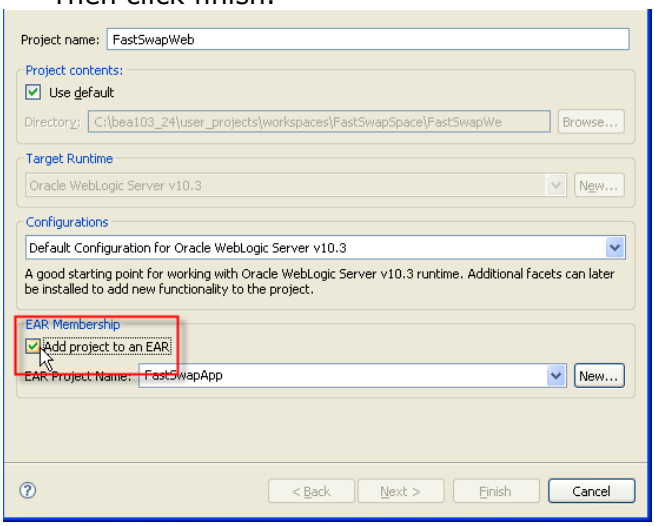
Limitations of FastSwap

- Java Reflection results do not include newly added fields and methods and include removed fields and methods. As a result of this, use of the reflection API on the modified classes can result in undesired behavior.
- Changing the hierarchy of an already existing class is not supported by FastSwap. Example: a) Changing the list of implemented interfaces of a class. b) Changing the superclass of a class is not supported.

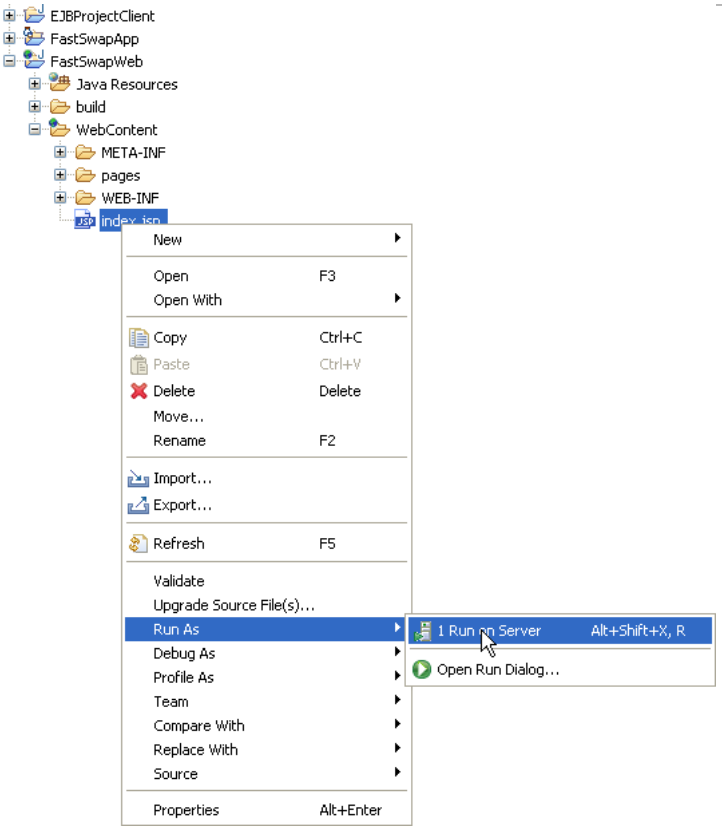
- Addition or Removal of Java Annotations is not supported by FastSwap, since this is tied to reflection changes mentioned above.
- Addition or Removal of methods on EJB Interfaces is not supported by FastSwap since an EJB Compilation step is required to reflect the changes at runtime.
- Addition or Removal of constants from Enums not supported in this release.
- Addition or Removal of the finalize method is not supported.

Steps:

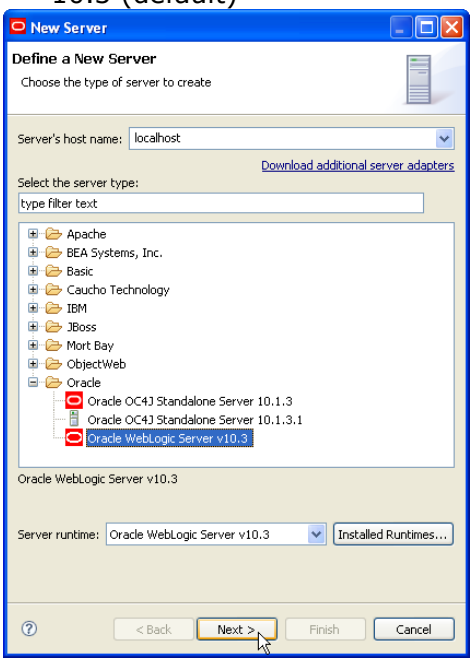
1. Create a new workspace called 'FastSwapSpace'
2. Create a new Enterprise Application called 'FastSwapApp'
3. Select File -> New -> Project -> Web -> Dynamic Web Project called 'FastSwapWeb' and check the box for "add project to an EAR, as shown below. Then click finish.



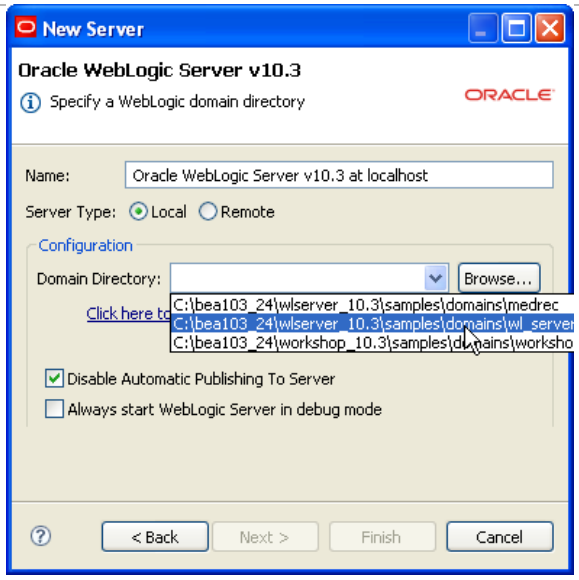
4. Create a new EJB project called 'EJBProject'. Select File -> New -> Project -> EJB -> EJB Project... then check the "add project to an EAR" similar to the preceding step, then click finish.
5. Go to Window -> Preferences -> Validation and disable all validation
6. Create a package under 'ejbModule' called 'sessionbean'
7. Copy 'Account.java', 'AccountBean.java' and 'AuditInterceptor.java' into the sessionbean package from the lab resources
8. Copy 'index.jsp' from the lab resources to the FastSwapWeb/WebContent folder
9. In the project explorer, right click on FastSwapWeb/WebContent/index.jsp and choose run as… run on server..



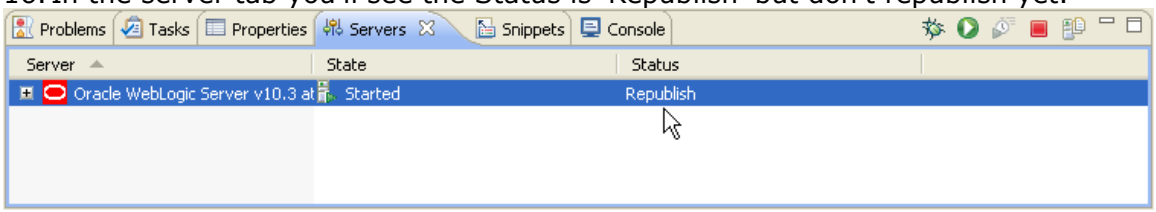
10. This will prompt you to create a server, choose the Oracle WebLogic Server 10.3 (default)



11. Click next.
12. When creating the server, you can use the sample domain that ships with the product by pointing to the directory <bea_home>\wlserver_10.3\samples\domains\wl_server. Choose the domain and then click finish.



13. The embedded browser will launch and navigate you to http://localhost:7001/FastSwapWeb/index.jsp. (If a separate IE window pops up, you can ignore or close it).
14. In the web page, you'll see the current balance is 100. Hit refresh a few times to see the balance increase by 100 each time.
15. In EJBProject open the AccountBean.java file and in the 'getBalance' method change the return to 'balance + 1'. Save the change.
16. In the server tab you'll see the Status is 'Republish' but don't republish yet.



17. Refresh the browser a few more times and notice that it still increments to an even number.
18. Now republish to the server
19. Go back to the browser and refresh and you'll see the balance is xx1 each time
20. Remove the '+1' from the getBalance method and save the change
21. Open the FastSwapApp\EarContent\META-INF\weblogic-application.xml file
22. Add '<fast-swap/>' just before the closing tag </wls:weblogic-application>
23. Save and republish. If this fails with a validation error, go to step 6.
24. Refresh the browser and the balance should now again be 100
25. Add the '+1' back to the getBalance method and save the change
26. Again the server status will be 'Republish' but don't publish
27. Refresh the browser and you'll see the xx1. This happens because the runtime class was updated in the classloader using 'FastSwap'.