# Developing a JAX-WS 2.2

Web Service

Java API for XML-based Web Services, known as JAX-WS ([http://jcp.org/](http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index2.html)

[aboutJava/communityprocess/mrel/jsr224/index2.html](http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index2.html)), is a W3C standards-

based technology for communicating between services and clients using XML over

the HTTP protocol. Some of the W3C standards that JAX-WS Web Services support

are HTTP (<http://www.w3.org/Protocols/>), SOAP ([http://www.w3.org/TR/](http://www.w3.org/TR/soap/)

<soap/>), and **Web Service Description Language** (**WSDL**) ([http://www.w3.org/](http://www.w3.org/TR/wsdl)

[TR/wsdl](http://www.w3.org/TR/wsdl)). JAX-WS is a platform-independent standard; JAX-WS Web Services may

communicate with non-Java clients, for example a .NET client, and a JAX-WS client

may communicate with non-Java Web Services, a .NET Web Service for example.

JAX-WS makes use of XML in the following artifacts:

* The WSDL is an XML document that describes network services as a set
* of endpoints operating on messages; an endpoint being a URL location
* representing a web service
* A client and a web service communicate using SOAP messages, which are in

the XML format

A JAX-WS web service consists of the following components:

* A non-final, non-abstract Java class that is annotated with javax.jws.
* WebService annotation. The web service endpoints, the Java class consists of
* business methods. A web service client creates a proxy of the web service to
* invoke its methods (operations). The web service optionally returns a response.
* The web service request and response are SOAP messages over HTTP.
* An optional Java interface that defines the methods implemented in the web

service implementation class.

**[ 209 ]**

*Developing a JAX-WS 2.2 Web Service*

This chapter has the following sections:

* Setting up the environment
* Creating a Java EE web project
* Creating a web descriptor
* Creating a JAX-WS Web Service
* Creating a web service client
* Deploying the JAX-WS application with Maven
* Running the JAX-WS application

## Setting up the environment

##### We need to download and install the following software:

* **WildFly 8.1.0.Final**: Download wildfly-8.1.0.Final.zip from

<http://wildfly.org/downloads/>.

* **MySQL 5.6 Database-Community Edition**: Download this edition from <http://dev.mysql.com/downloads/mysql/>. When installing MySQL, install **Connector/J** too.
* **Eclipse IDE for Java EE Developers**: Download Eclipse Luna from

<https://www.eclipse.org/downloads/packages/release/Luna/SR1>.

* **JBoss Tools (Luna) 4.2.0.Final (or the latest version)**: Install this as a plugin to Eclipse from the Eclipse Marketplace ([http://tools.jboss.org/](http://tools.jboss.org/downloads/installation.html) [downloads/installation.html](http://tools.jboss.org/downloads/installation.html)).
* **Apache Maven**: Download version 3.05 or higher from [http://maven.](http://maven.apache.org/download.cgi) [apache.org/download.cgi](http://maven.apache.org/download.cgi).
* **Java 7**: Download Java 7 from [http://www.oracle.com/technetwork/](http://www.oracle.com/technetwork/java/javase/downloads/index.html?ssSourceSiteId=ocomcn) [java/javase/downloads/index.html?ssSourceSiteId=ocomcn](http://www.oracle.com/technetwork/java/javase/downloads/index.html?ssSourceSiteId=ocomcn).

Set the environment variables, JAVA\_HOME, JBOSS\_HOME, and MAVEN\_HOME. Add

%JAVA\_HOME%/bin, %MAVEN\_HOME%/bin, and %JBOSS\_HOME%/bin to the PATH

##### environment variable.

Create a WildFly 8.1.0 runtime as discussed in *Chapter 1*, *Getting Started with EJB 3.x*.

**[ 210 ]**

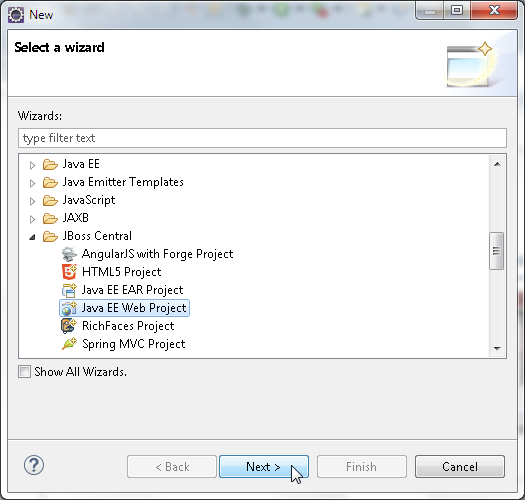
## Creating a Java EE web project

*Chapter 6*

In this section, we will create a Java EE Web Project for a JAX-WS Web Service. Select

**File** | **New**. In **New**, select **JBoss Central** | **Java EE Web Project**, as shown in the

following screenshot. Now click on **Next**.



**[ 211 ]**

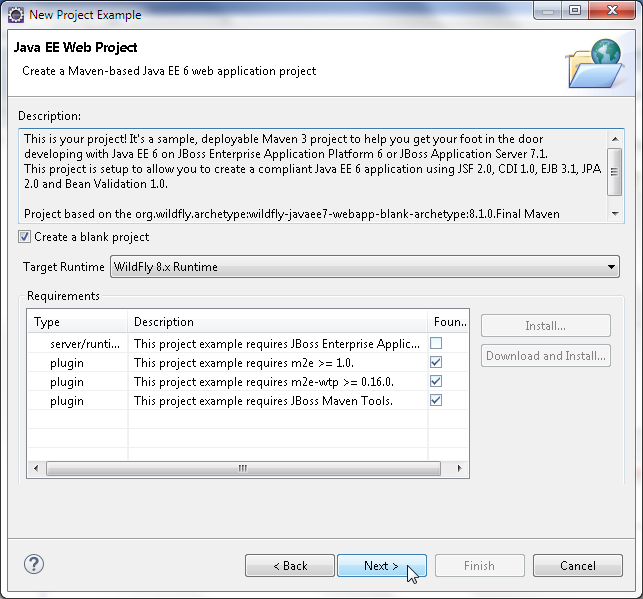
*Developing a JAX-WS 2.2 Web Service*

A test is run for the requirements, which include **m2e** and **m2eclipse-wtp** plugins,

and the **JBoss Maven Tools** plugin. Select the checkbox and create a blank project;

select **WildFly 8.x Runtime** for **Target Runtime**, as shown in the following

screenshot. Now click on **Next**.

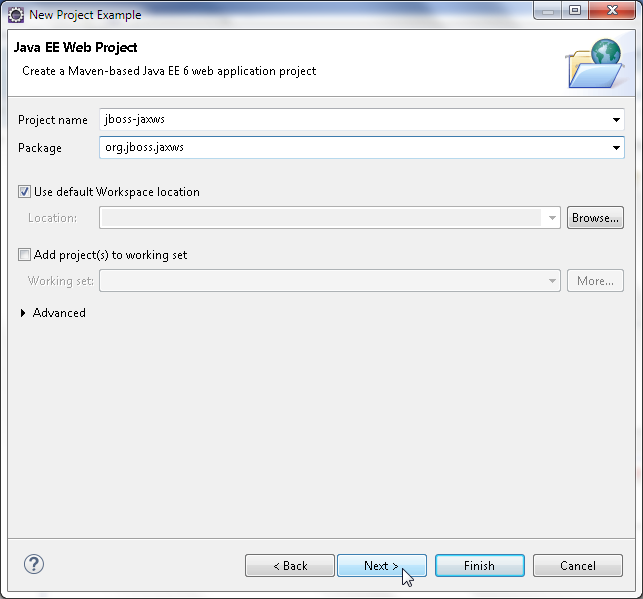


**[ 212 ]**

*Chapter 6*

Specify **Project name** (jboss-jaxws) and **Package** (org.jboss.jaxws), as shown in

##### the following screenshot. Now click on **Next**.



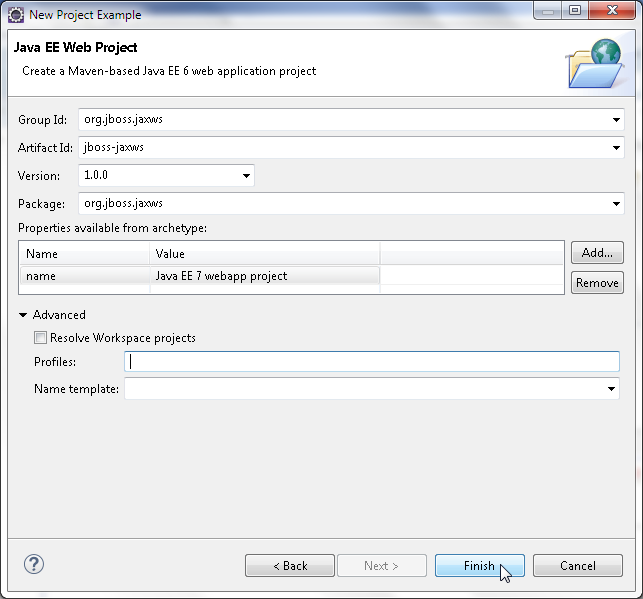
**[ 213 ]**

*Developing a JAX-WS 2.2 Web Service*

Specify **Group Id** as org.jboss.jaxws, **Artifact Id** (jboss-jaxws), **Version** (1.0.0),

and **Package** (org.jboss.jaxws), as shown in the following screenshot. Then click

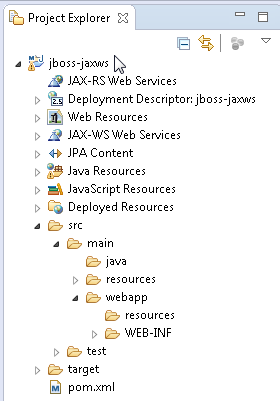
on **Finish**.



A Maven-based Java EE Web Project, jboss-jaxws, is generated, as shown in

**Project Explorer** in the following screenshot. Delete the jboss-jaxws/src/main/

resources/META-INF/persistence.xml configuration file:



**[ 214 ]**

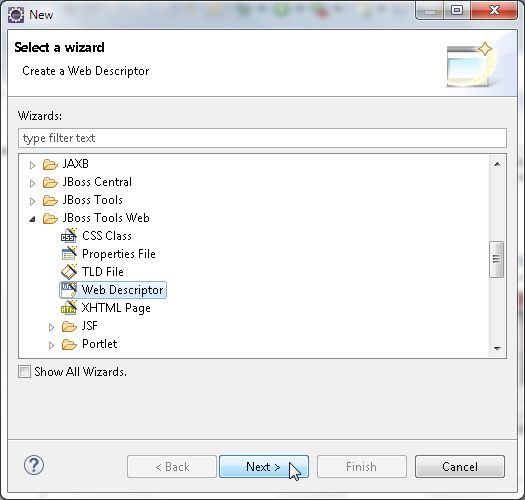
## Creating a web descriptor

*Chapter 6*

A JAX-WS Web Service requires a Web Descriptor. In this section, we will create a

web descriptor. Select **File** | **New** | **Other**. In **New**, select **JBoss Tools Web** | **Web**

**Descriptor** and click on **Next**, as shown in the following screenshot:



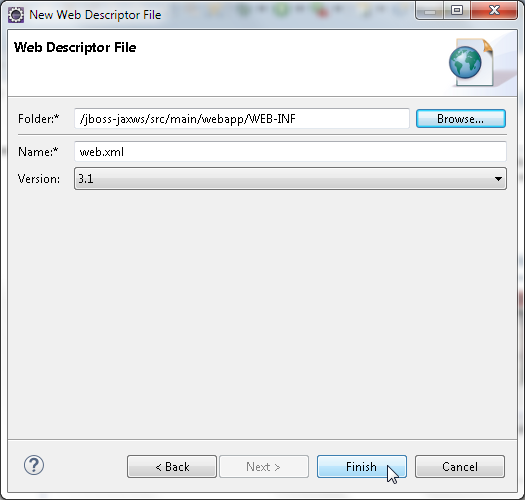
**[ 215 ]**

*Developing a JAX-WS 2.2 Web Service*

Click on **Browse** for the **Folder** field to select the webapp/WEB-INF folder. Specify

**Name** as web.xml, select **Version** as **3.1**, and click on **Finish**, as shown in the

following screenshot:



**[ 216 ]**

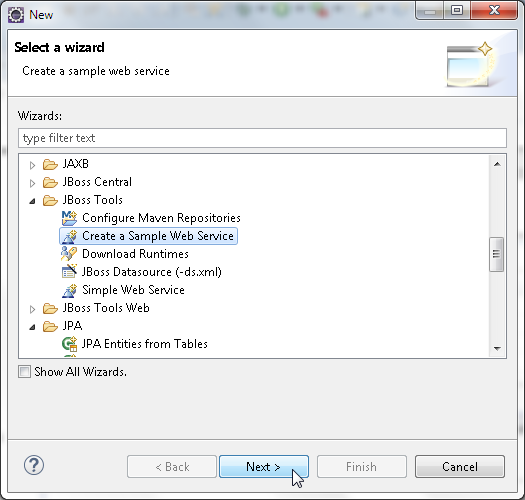
## Creating a JAX-WS web service

*Chapter 6*

In this section, we will create a JAX-WS Web Service in the Java EE Web Project.

Select **File** | **New** | **Other**. In **New**, select **JBoss Tools** | **Create a Sample Web**

**Service**, as shown in the following screenshot. Now click on **Next**.



**[ 217 ]**

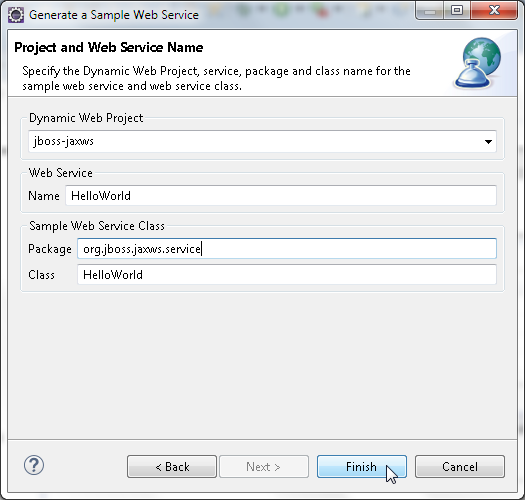
*Developing a JAX-WS 2.2 Web Service*

In **Generate a Sample Web Service**, specify **Project and Web Service Name**. Select the

jboss-jaxws web project and specify **Name** of **Web Service** as HelloWorld. For

**Sample Web Service Class**, specify **Package** as org.jboss.jaxws.service and

**Class** as HelloWorld, as shown in the following screenshot. Now click on **Finish**.



##### The HelloWorld class gets created. Annotate the class with the @WebService

##### annotation, which indicates that the class implements a web service. Add the

##### following elements in the @WebService annotation:

|  |  |  |
| --- | --- | --- |
| **Element** | **Description** | **Value** |
| portName | This is the port name of the web service. This is also  wsdl:portname in the web service WSDL. | HelloWorldPort |

**[ 218 ]**

*Chapter 6*

|  |  |  |
| --- | --- | --- |
| **Element** | **Description** | **Value** |
| serviceName | This is the service name of the web service.  This is also the  ws:servicename  in the WSDL. | HelloWorldService |
| targetNamespace | This is the target namespace for the wsdl:service in the WSDL. | <http://org.jboss.jaxws.service/> |
| endpointInterface | This is the name of the service endpoint interface defining the web service methods. | org.jboss.jaxws.service.HelloWS |

##### The HelloWorld implementation class implements the HelloWS interface. Add a

##### business method sayHello(String) that takes a String name as argument and

##### returns a String message. The business method is annotated with the @WebMethod

##### annotation. A business method must not be static or final. The business methods

##### are exposed to a client as web service operations. The web service endpoint Java

##### class is listed here:

package org.jboss.jaxws.service;

import javax.jws.WebMethod; import javax.jws.WebService;

@WebService(portName = "HelloWorldPort", serviceName = "HelloWorldService", targetNamespace = ["http://org.jboss.jaxws.service/",](http://org.jboss.jaxws.service/) endpointInterface = "org.jboss.jaxws.service.HelloWS")

public class HelloWorld implements HelloWS{

@WebMethod ()

public String sayHello(String name) {

return "Hello "+name +" Welcome to Web Services!";

}

}

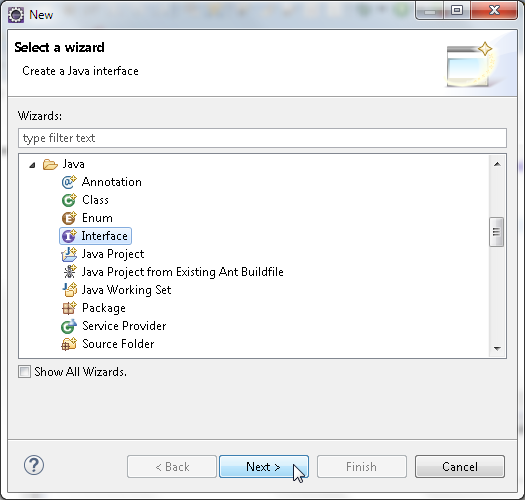
**[ 219 ]**

*Developing a JAX-WS 2.2 Web Service*

##### As the endpoint implements an endpoint interface, create a Java interface. Select

##### **File** | **New** | **Other**. In **New**, select **Interface** and click on **Next**, as shown in the

##### following screenshot:



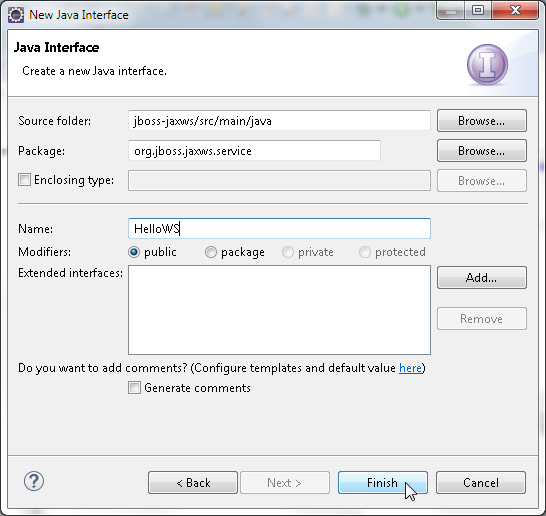
In **New Java Interface**, select **Source folder** as src/main/java, specify **Package** as

org.jboss.jaxws.service, and specify **Name** as HelloWS, as shown here. Then

click on **Finish**.

**[ 220 ]**

*Chapter 6*



The org.jboss.jaxws.HelloWS interface gets added to the jobss-jaxws

project. Annotate the HelloWS interface with @WebService, which indicates

##### that the Java interface is a web service interface. The name element of @WebService

specifies the web service name, which is used as the name of wsdl:portType.

The targetNamespace specifies the target namespace for the web service. The

endpoint interface defines a sayHello method annotated with @WebMethod, with

its operationName element set to hello. The endpoint interface is listed here:

package org.jboss.jaxws.service;

import javax.jws.WebMethod; import javax.jws.WebService;

@WebService(name = "HelloWS", targetNamespace = ["http://org.jboss.jaxws.service/")](http://org.jboss.jaxws.service/)

**[ 221 ]**

*Developing a JAX-WS 2.2 Web Service*

public interface HelloWS { @WebMethod(operationName = "hello") public String sayHello(String name);

}

##### A JAX-WS Web Service can be published using the JSR-109 programming model for

##### implementing web services in Java. WildFly 8.1 supports the JSR-109 deployment

##### model in which a web service can be configured as a servlet class in web.xml. We

##### will use the JSR-109 deployment model; we need to configure the web service

as a servlet in web.xml. Specify the endpoint class org.jboss.jaxws.service.

HelloWorld as a servlet in web.xml with the corresponding servlet mapping URL as

/HelloWorld. The web.xml file is listed here:

<?xml version="1.0" encoding="UTF-8"?>

<web-app [xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance) [xmlns="http://xmlns.jcp.org/xml/ns/javaee"](http://xmlns.jcp.org/xml/ns/javaee) [xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee](http://xmlns.jcp.org/xml/ns/javaee) [http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"](http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd) version="3.1">

<servlet>

<display-name>HelloWorld</display-name>

<servlet-name>HelloWorld</servlet-name>

<servlet-class>org.jboss.jaxws.service.HelloWorld</servlet- class>

</servlet>

<servlet-mapping>

<servlet-name>HelloWorld</servlet-name>

<url-pattern>/HelloWorld</url-pattern>

</servlet-mapping>

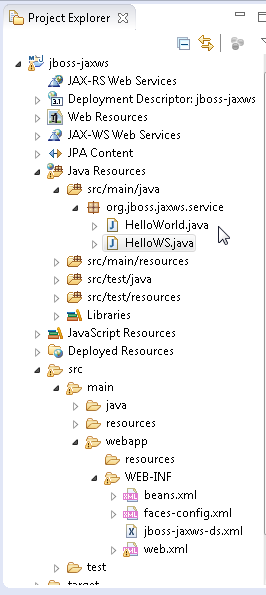
</web-app>

**[ 222 ]**

*Chapter 6*

##### The HelloWorld web service class and the service endpoint interface HelloWS are

##### shown in Eclipse in the following screenshot:



**[ 223 ]**

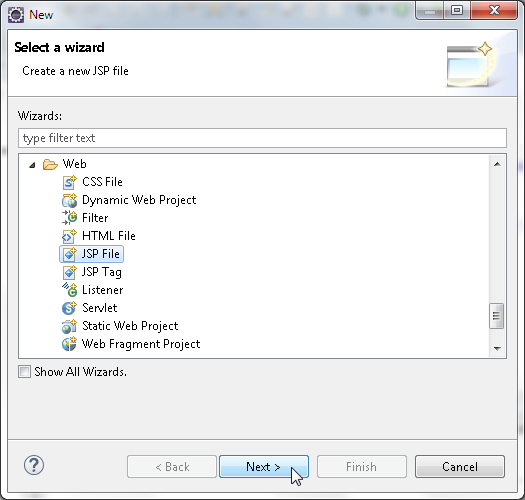
*Developing a JAX-WS 2.2 Web Service*

## Creating a web service client

In this section, we will create a JSP Web Service client for the HelloWorld web

service. To create a JSP, select **File** | **New** | **Other**. In **New**, select **Web** | **JSP File**

and click on **Next**, as shown in the following screenshot:

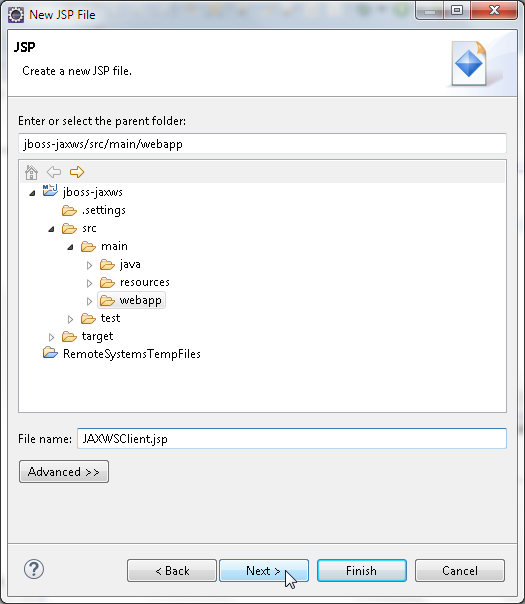


In the **New JSP File** wizard, select the webapp folder, specify **File name** as

JAXWSClient.jsp, and click on **Next**, as shown in the following screenshot:

**[ 224 ]**

*Chapter 6*



##### Select the **New JSP file (html)** template and click on **Finish**. The JAXWSClient.jsp

##### file gets added to the webapp folder. In the client JSP, we will invoke the web service

##### with a name and display the web service response in the browser. First, create a URL

##### object for the WSDL. The URL for a web service is constructed from the context root

+ endpoint:

URL wsdlLocation = new URL("http://localhost:8080/jboss-jaxws/ HelloWorld?WSDL");

**[ 225 ]**

*Developing a JAX-WS 2.2 Web Service*

The jboss-jaxws in the URL is the context root and the /HelloWorld is the servlet

mapping URL for the web service endpoint as specified in the web.xml. Next, create

a QName object for the service name. A QName represents a qualified name. Specify

arguments to the QName constructor as the target na[mespace http://org.jboss.](http://org.jboss.jaxws.service/)

[jaxws.](http://org.jboss.jaxws.service/)service/ and the service name HelloWorldService:

QName serviceName = new [QName("http://org.jboss.jaxws.service/","HelloWorldService");](http://org.jboss.jaxws.service/)

The client view of a web service is provided by a javax.xml.ws.Service object.

Create javax.xml.ws.Service from the WSDL location URL and QName for the

service name:

Service service = Service.create(wsdlLocation, serviceName);

Get a proxy to the web service using the getPort(Class endpointInterface)

method. Specify the endpoint interface class as org.jboss.jaxws.service.

HelloWS.class:

HelloWS port = service.getPort(org.jboss.jaxws.service.HelloWS.class);

##### Invoke the sayHello method of the web service proxy with a name as an argument.

##### The sayHello method returns a String. Output the response from the web service:

String result = port.sayHello("John Smith"); out.println(result);

The JAXWSClient.jsp is listed here:

<%@ page language="java" contentType="text/html; charset=ISO-8859- 1" pageEncoding="ISO-8859-1"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" ["http://www.w3.org/TR/html4/loose.dtd">](http://www.w3.org/TR/html4/loose.dtd)

<%@ page import="org.jboss.jaxws.service.\*,javax.xml.ws.WebServiceRef, java.net.URL,javax.xml.namespace.QName,javax.xml.ws.Service"%>

<html>

<head>

<meta http-equiv="Content-Type" content="text/xml; charset=windows-1252" />

<title>JAXWS Client</title>

</head>

<body>

<%URL wsdlLocation = new URL ( "http://localhost:8080/jboss-jaxws/HelloWorld?WSDL");

**[ 226 ]**

*Chapter 6*

QName serviceName = new

[QName("http://org.jboss.jaxws.service/",](http://org.jboss.jaxws.service/) "HelloWorldService"); Service service = Service.create(wsdlLocation, serviceName);

HelloWS port = service.getPort(org.jboss.jaxws.service.HelloWS.class);

String result = port.sayHello("John Smith"); out.println(result);

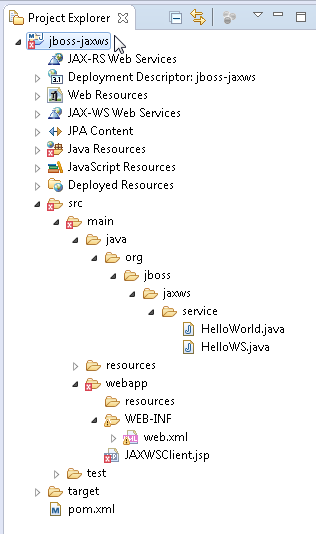
%>

</body>

</html>

##### The directory structure of the jboss-jaxws application is shown in Java EE web

project in the following screenshot:



**[ 227 ]**

*Developing a JAX-WS 2.2 Web Service*

## Deploying the JAX-WS application with Maven

##### In this section, we will compile, package, and deploy the jboss-jaxws application

##### to WildFly 8.1. Specify the JAX-WS-related dependencies discussed in the following

##### table in the pom.xml file:

|  |  |
| --- | --- |
| **Dependency** | **Description** |
| **Group Id**: com.sun.xml.ws  **Artifact Id**: jaxws-rt | Open source reference implementation of the  JSR 224: Java API for XML Web Services |
| **Group Id**: javax.xml.ws  **Artifact Id**: jaxws-api | JAX-WS (JSR 224) API |
| **Group Id**: com.sun.xml.bind  **Artifact Id**: jaxb-impl | JAXB (JSR 222) Reference implementation |
| **Group Id**: com.sun.xml.bind  **Artifact Id**: jaxb-xjc | JAXB (JSR 222) Reference implementation-  schema compiler |

We will use the JAX-WS Maven Plugin ([http://jax-ws-commons.java.net/](http://jax-ws-commons.java.net/jaxws-maven-plugin/)

[jaxws-maven-plugin/](http://jax-ws-commons.java.net/jaxws-maven-plugin/)), which is the Maven adapter for the JAX-WS's toolset.

The plugin provides wsgen and wsimport goals to generate the required portable

artifacts for a web service. First, run the wsgen goal to generate the JAX-WS web

service portable artifacts, including the WSDL, from an endpoint implementation

class. Subsequently, run the wsimport goal to generate the web service portable

artifacts, used by web service clients, from a WSDL. For the wsgen goal, specify the

service endpoint interface in the <sei/> element as org.jboss.jaxws.service.

HelloWorld, and specify the service name in the <serviceName/> element as

HelloWorldService. Set <genwsdl/> for the wsgen goal to true. Specify the Maven

Compiler Plugin and the Maven WAR Plugin, with the output directory set to the

deployments directory. The pom.xml code is listed here:

<?xml version="1.0" encoding="UTF-8"?>

<project [xmlns="http://maven.apache.org/POM/4.0.0"](http://maven.apache.org/POM/4.0.0) [xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance) [xsi:schemaLocation="http://maven.apache.org/POM/4.0.0](http://maven.apache.org/POM/4.0.0) [http://maven.apache.org/maven-v4\_0\_0.xsd">](http://maven.apache.org/maven-v4_0_0.xsd)

<modelVersion>4.0.0</modelVersion>

<groupId>org.jboss.jaxws</groupId>

<artifactId>jboss-jaxws</artifactId>

<version>1.0.0</version>

<packaging>war</packaging>

<name>Java EE 7 webapp project</name>

**[ 228 ]**

*Chapter 6*

<description>A starter Java EE 7 webapp project for use on JBoss WildFly / WildFly, generated from the jboss-javaee6-webapp archetype</description>

[<url>http://wildfly.org</url>](http://wildfly.org/)

<properties>

<project.build.sourceEncoding>UTF- 8</project.build.sourceEncoding>

<!-- JBoss dependency versions -->

<version.wildfly.maven.plugin>1.0.2.Final

</version.wildfly.maven.plugin>

<!-- Define the version of the JBoss BOMs we want to import to specify

tested stacks. -->

<version.jboss.bom>8.1.0.Final</version.jboss.bom>

<!-- other plugin versions -->

<version.compiler.plugin>3.1</version.compiler.plugin>

<version.war.plugin>2.1.1</version.war.plugin>

<!-- maven-compiler-plugin -->

<maven.compiler.target>1.7</maven.compiler.target>

<maven.compiler.source>1.7</maven.compiler.source>

</properties>

<repositories>

<repository>

<id>JBoss Repository</id>

<url>https://repository.jboss.org/nexus/content

/groups/public/</url>

</repository>

</repositories>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.wildfly.bom</groupId>

<artifactId>jboss-javaee-7.0-with-tools</artifactId>

<version>${version.jboss.bom}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

<dependency>

<groupId>org.wildfly.bom</groupId>

<artifactId>jboss-javaee-7.0-with-hibernate</artifactId>

<version>${version.jboss.bom}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

**[ 229 ]**

*Developing a JAX-WS 2.2 Web Service*

</dependencyManagement>

<dependencies>

<!-- First declare the APIs we depend on and need for compilation. All of them are provided by JBoss WildFly -->

<!-- Import the CDI API, we use provided scope as the API is included in

JBoss WildFly -->

<dependency>

<groupId>javax.enterprise</groupId>

<artifactId>cdi-api</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>com.sun.xml.ws</groupId>

<artifactId>jaxws-rt</artifactId>

<version>2.2.8</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.xml.ws</groupId>

<artifactId>jaxws-api</artifactId>

<version>2.2.8</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>com.sun.xml.bind</groupId>

<artifactId>jaxb-impl</artifactId>

<version>2.2.7</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

</dependency>

<!-- Import the Common Annotations API (JSR-250), we use provided scope

as the API is included in JBoss WildFly -->

<dependency>

<groupId>org.jboss.spec.javax.annotation</groupId>

<artifactId>jboss-annotations-api\_1.2\_spec</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

**[ 230 ]**

*Chapter 6*

<groupId>javax.xml.bind</groupId>

<artifactId>jaxb-api</artifactId>

<version>2.2.7</version>

<scope>provided</scope>

</dependency>

</dependencies>

<build>

<!-- Maven will append the version to the finalName (which is the name

given to the generated war, and hence the context root) -->

<finalName>${project.artifactId}</finalName>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.eclipse.m2e</groupId>

<artifactId>lifecycle-mapping</artifactId>

<version>1.0.0</version>

<configuration>

<lifecycleMappingMetadata>

<pluginExecutions>

<pluginExecution>

<pluginExecutionFilter>

<groupId>org.jvnet.jax-ws-commons</groupId>

<artifactId>jaxws-maven-plugin</artifactId>

<versionRange>[2.2,)</versionRange>

<goals>

<goal>wsimport</goal>

</goals>

</pluginExecutionFilter>

<action>

<ignore />

</action>

</pluginExecution>

</pluginExecutions>

</lifecycleMappingMetadata>

</configuration>

</plugin>

<!-- Compiler plugin enforces Java 1.6 compatibility and activates annotation processors -->

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>${version.compiler.plugin}</version>

<configuration>

<source>${maven.compiler.source}</source>

**[ 231 ]**

*Developing a JAX-WS 2.2 Web Service*

<target>${maven.compiler.target}</target>

</configuration>

</plugin>

<plugin>

<artifactId>maven-war-plugin</artifactId>

<version>${version.war.plugin}</version>

<configuration>

<outputDirectory>C:\wildfly-8.1.0.Final\standalone\deployments</outputDirectory>

<!-- Java EE 7 doesn't require web.xml, Maven needs to catch up! -->

<failOnMissingWebXml>false</failOnMissingWebXml>

</configuration>

</plugin>

<!-- The WildFly plugin deploys your war to a local WildFly container -->

<!-- To use, run: mvn package wildfly:deploy -->

<plugin>

<groupId>org.wildfly.plugins</groupId>

<artifactId>wildfly-maven-plugin</artifactId>

<version>${version.wildfly.maven.plugin}</version>

</plugin>

<plugin>

<groupId>org.jvnet.jax-ws-commons</groupId>

<artifactId>jaxws-maven-plugin</artifactId>

<version>2.2</version>

<executions>

<execution>

<id>HelloWorldService</id>

<phase>compile</phase>

<goals>

<goal>wsgen</goal>

</goals>

<configuration>

<sei>org.jboss.jaxws.service.HelloWorld</sei>

<genwsdl>true</genwsdl>

<servicename>HelloWorldService</servicename>

<keep>true</keep>

</configuration>

</execution>

</executions>

</plugin>

<plugin>

<groupId>org.jvnet.jax-ws-commons</groupId>

**[ 232 ]**

*Chapter 6*

<artifactId>jaxws-maven-plugin</artifactId>

<version>2.3</version>

<executions>

<execution>

<id>HelloWorldService</id>

<goals>

<goal>wsimport</goal>

</goals>

</execution>

</executions>

<configuration>

<packagename>org.jboss.jaxws.service</packagename>

<target>2.0</target>

<keep>true</keep>

</configuration>

<dependencies>

<dependency>

<groupId>com.sun.xml.ws</groupId>

<artifactId>jaxws-tools</artifactId>

<version>2.2.8</version>

<exclusions>

<exclusion>

<groupId>org.jvnet.staxex</groupId>

<artifactId>stax-ex</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.jvnet.staxex</groupId>

<artifactId>stax-ex</artifactId>

<version>1.7.6</version>

<exclusions>

<exclusion>

<groupId>javax.xml.stream</groupId>

<artifactId>stax-api</artifactId>

</exclusion>

</exclusions>

</dependency>

</dependencies>

</plugin>

</plugins>

</pluginManagement>

</build>

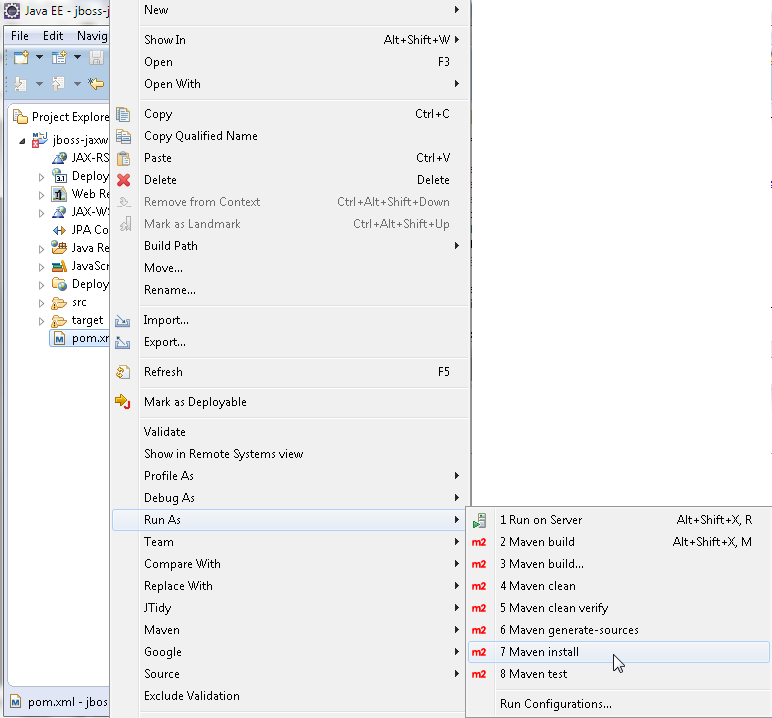
</project>

**[ 233 ]**

*Developing a JAX-WS 2.2 Web Service*

Right-click on pom.xml in the **Project Explorer** and select **Run As** | **Maven install**, as

shown in the following screenshot:



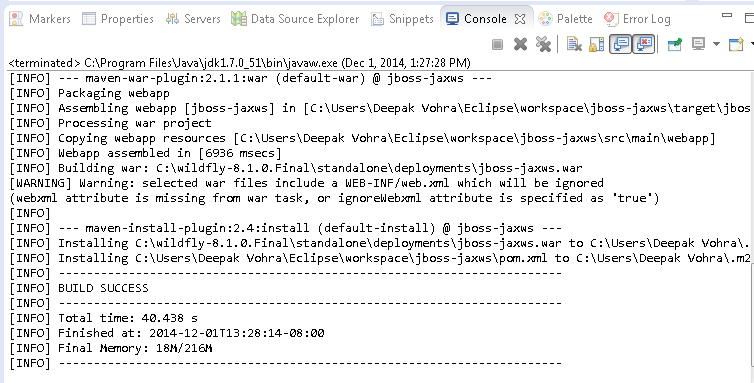
The jboss-jaxws project gets compiled and packaged into a jboss-jaxws.war

archive. A BUILD SUCCESS message indicates that the Maven build completed

without any error, as shown here:

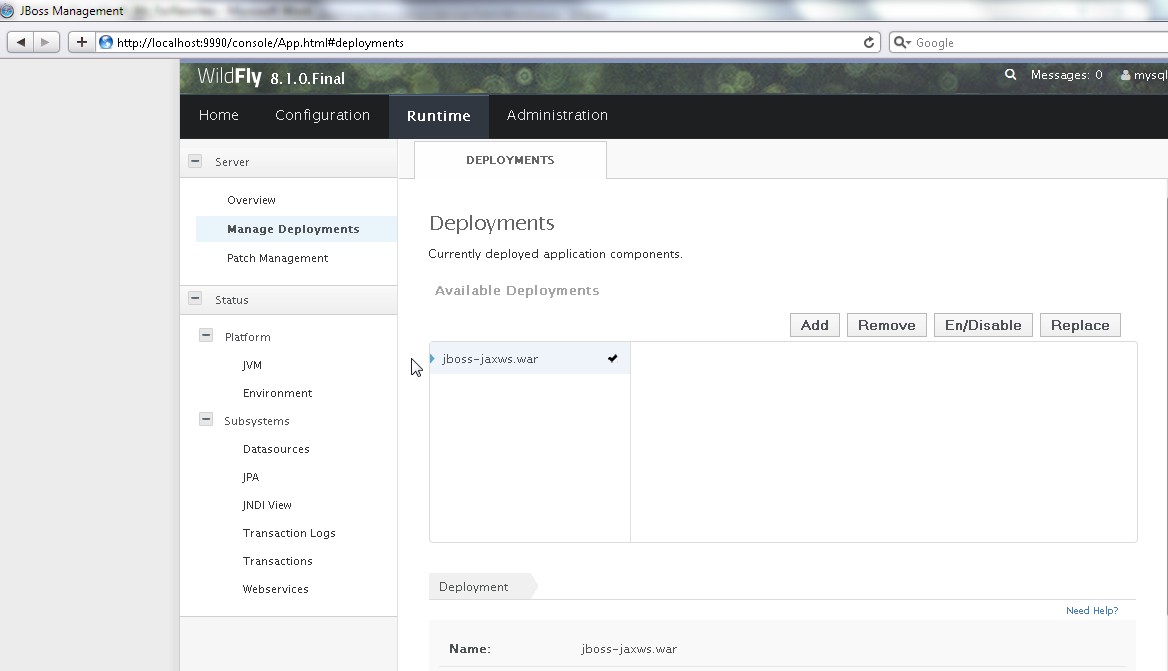
**[ 234 ]**

*Chapter 6*



##### Start WildFly 8.1 if it is not already started. The jboss-jaxws.war archive gets

deployed to WildFly 8.1, as shown in the **Administration** console:



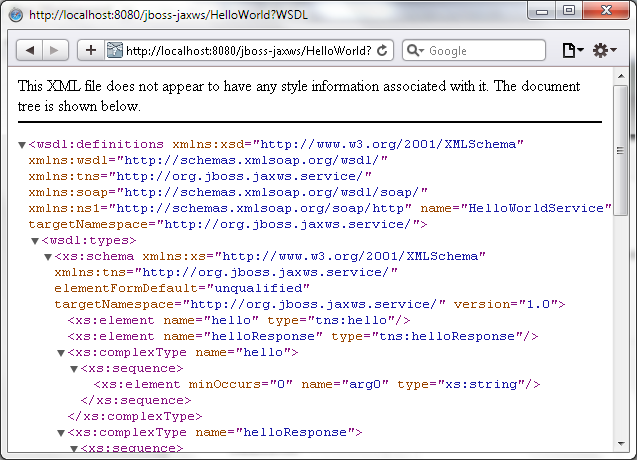
**[ 235 ]**

*Developing a JAX-WS 2.2 Web Service*

The WSDL for the web service can be invoked in a browser with the URL

http://localhost:8080/jboss-jaxws/HelloWorld?WSDL, as shown in the

following screenshot:



##### The WSDL file's content is shown as follows:

<wsdl:definitions [xmlns:xsd="http://www.w3.org/2001/XMLSchema"xmln](http://www.w3.org/2001/XMLSchema) [s:wsdl="http://schemas.xmlsoap.org/wsdl/"xmlns:tns="http://org.jbo](http://org.jbo/) [ss.jaxws.service/"xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap](http://schemas.xmlsoap.org/wsdl/soap)

[/"xmlns:ns1="http://schemas.xmlsoap.org/soap/http"](http://schemas.xmlsoap.org/soap/http) name="HelloWorl [dService"targetNamespace="http://org.jboss.jaxws.service/">](http://org.jboss.jaxws.service/)

<wsdl:types>

<xs:schema [xmlns:xs="http://www.w3.org/2001/XMLSchema"xmlns:tns="h](http://www.w3.org/2001/XMLSchema) ttp://org.jboss.jaxws.service/"elementFormDefault="unqualified"tar [getNamespace="http://org.jboss.jaxws.service/"](http://org.jboss.jaxws.service/) version="1.0">

<xs:element name="hello" type="tns:hello"/>

<xs:element name="helloResponse" type="tns:helloResponse"/>

<xs:complexType name="hello">

<xs:sequence>

<xs:element minOccurs="0" name="arg0" type="xs:string"/>

</xs:sequence>

</xs:complexType>

<xs:complexType name="helloResponse">

**[ 236 ]**

*Chapter 6*

<xs:sequence>

<xs:element minOccurs="0" name="return" type="xs:string"/>

</xs:sequence>

</xs:complexType>

</xs:schema>

</wsdl:types>

<wsdl:message name="hello">

<wsdl:part element="tns:hello" name="parameters"></wsdl:part>

</wsdl:message>

<wsdl:message name="helloResponse">

<wsdl:part element="tns:helloResponse" name="parameters">

</wsdl:part>

</wsdl:message>

<wsdl:portType name="HelloWS">

<wsdl:operation name="hello">

<wsdl:input message="tns:hello" name="hello"></wsdl:input>

<wsdl:output message="tns:helloResponse"name="helloResponse">

</wsdl:output>

</wsdl:operation>

</wsdl:portType>

<wsdl:binding name="HelloWorldServiceSoapBinding" type="tns:HelloW S">

<soap:binding [style="document"transport="http://schemas.xmlsoap.or](http://schemas.xmlsoap.or/) g/soap/http"/>

<wsdl:operation name="hello">

<soap:operation soapAction="" style="document"/>

<wsdl:input name="hello">

<soap:body use="literal"/>

</wsdl:input>

<wsdl:output name="helloResponse">

<soap:body use="literal"/>

</wsdl:output>

</wsdl:operation>

</wsdl:binding>

<wsdl:service name="HelloWorldService">

<wsdl:port binding="tns:HelloWorldServiceSoapBinding" name="HelloWorldPort">

<soap:address location="http://localhost:8080/jboss- jaxws/HelloWorld"/>

</wsdl:port>

</wsdl:service>

</wsdl:definitions>

**[ 237 ]**

*Developing a JAX-WS 2.2 Web Service*

## Running the JAX-WS application

In this section, we will invoke the web service using a JSP client. Invoke the

JAXWSClient.jsp file in a browser with the URL

http://localhost:8080/jboss- jaxws/JAXWSClient.jsp. The web service

response gets displayed in the browser, as shown in the following screenshot:



## Summary

##### In this chapter, we developed a JAX-WS Web Service in Eclipse IDE with JBoss Tools

##### sample web service. We created an endpoint implementation class and an endpoint

##### interface. We developed a JSP client for the web service. The web service is deployed

##### using the JSR-109 deployment model, which involves configuring the endpoint class

##### as a servlet in web.xml. We compiled and packaged the web service project with

##### Maven build tools and deployed the web application to WildFly 8.1. Subsequently,

##### we ran the web service client JSP in a browser.

In the next chapter, we will discuss RESTful web services with WildFly 8.1.

**[ 238 ]**

# Developing a JAX-RS 1.1

Web Service

In this chapter, we will discuss **Representational State Transfer** (**REST**) web services, specifically those based on Java API for RESTful web services (JAX-