spring

WEB SERVICES

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Spring Web Services (Spring-WS) is one of the project developed by the Spring Community. Its prime focus is to create document-driven Web Services. The Spring Web Services project facilitates contract-first SOAP service development, provides multiple ways to create flexible web services, which can manipulate XML payloads in multiple ways. Being Spring based, Spring Web Services uses Spring Concepts like Dependency Injection and Configurations seamlessly. Spring-WS requires Spring 3.0 version.

Spring Framework was initially written by Rod Johnson and was first released under the Apache 2.0 license in June 2003. This tutorial has been written based on the Spring Framework Version 4.1.6 released in March 2015.

Audience

This tutorial is designed for Java Programmers with a need to understand the Spring Web Services Framework in detail along with its architecture and actual usage. This tutorial will bring the readers to the intermediate level of expertise and from there they can take themselves to a higher level of proficiency.

Prerequisites

Before proceeding with this tutorial, you should have a good understanding of Java Programming Language. Additionally, understanding of the Eclipse IDE (Integrated Development Environment) is also required because all the examples have been compiled using the Eclipse IDE.

Copyright and Disclaimer

Table of Contents

1. Spring WS – Overview

Spring Web Services (Spring-WS) is one of the projects developed by the Spring Community. Its prime focus is to create document-driven Web Services. The Spring Web Services project facilitates contract-first **SOAP Service Development**, provides multiple ways to create flexible web services, which can manipulate XML payloads in multiple ways.

The Spring web services uses Spring concepts like dependency injection and configurations seamlessly. The Spring-WS requires Spring 3.0 Version. With contract-first development, we start with **WSDL Contract** and then will use JAVA to implement the required contract.

As opposed to the contract-last approach where JAVA interfaces generate WSDL/XSD contract. The WSDL based contract remains independent of JAVA implementation in the contract-first approach. In case we require changing the JAVA interfaces, then there is no need to communicate the changes made in the existing WSDL contract to the web services users. Spring-WS aims to provide loose coupling between the WSDL contract and its JAVA based implementation.

Features

Following are the features of Spring Web Services:

- **XML Mapping to Objects** – XML based requests can be mapped to any object using the information stored in the Message Payload, SOAP Action Header or by using an XPath Expression.

- **Multiple API Support to parse XML** – Apart from the standard JAXP APIs (DOM, SAX, StAX) to parse the incoming XML requests, other libraries like JDOM, dom4j, XOM are also supported.

- **Multiple API Support to marshal XML** – Spring Web Services supports JAXB 1 and 2, Castor, XMLBeans, JiBX, and XStream libraries using its Object/XML Mapping module. The Object/XML Mapping module can also be used in non-web services code as well.

- **Spring based configurations** – Spring Web Services uses the Spring Application Contexts for its configurations having a similar architecture as that of the Spring Web MVC.

- **Integrated WS-Security module** – Using the WS-Security module, you can Sign, Encrypt, Decrypt SOAP Messages or Authenticate them.

- **Support for Acegi Security –** Using the WS-Security implementation of Spring Web Services, Acegi configuration can be used for your SOAP services.

Architecture

The Spring-WS project consists of five major modules, which are explained below.

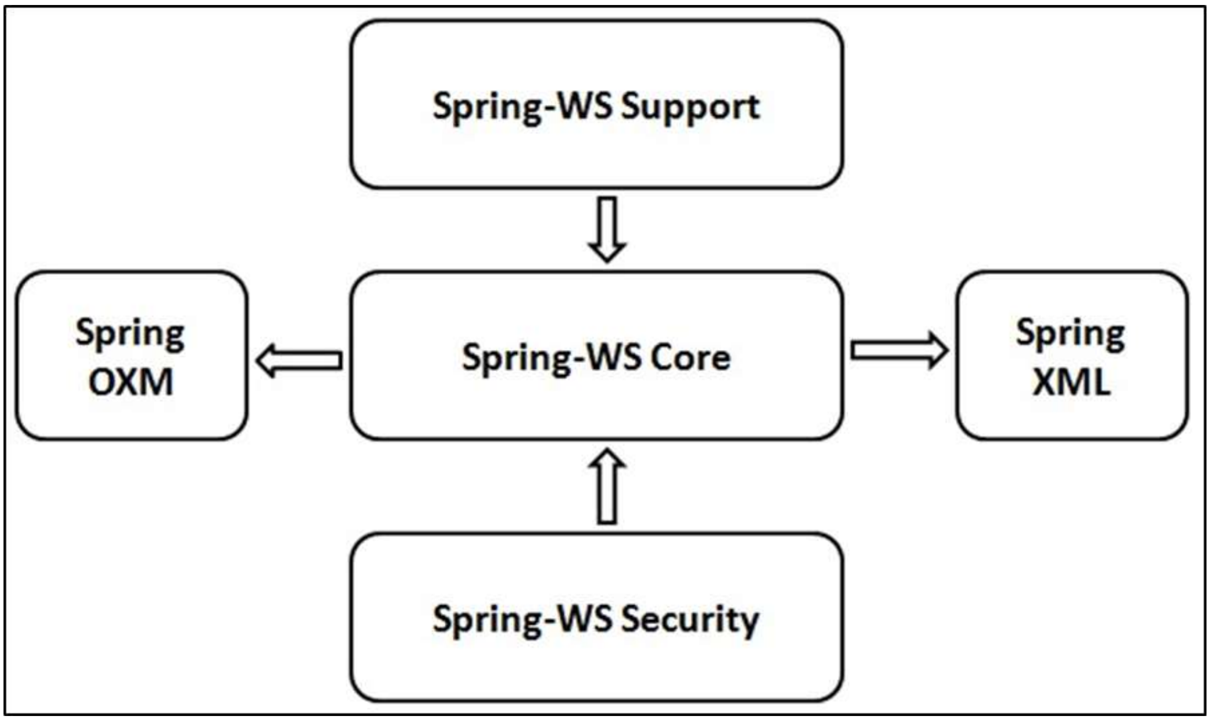
- **Spring-WS Core** – It is the primary module and provides the Central Interfaces like **WebServiceMessage** and **SoapMessage**, the server-side framework, powerful message dispatching capability and support classes to implement Web service endpoints. It also provides Web Service consumer client as **WebServiceTemplate**.

- **Spring-WS Support** – This module provides supports for JMS, emails, etc.

- **Spring-WS Security** – This module is responsible to provide WS-Security implementation integrated with core Web Service Module. Using this module, we can add principal tokens, sign, encrypt and decrypt SOAP messages. This module allows using the existing Spring Security Implementation for authentication and authorization.

- **Spring XML** – This module provides XML support classes for Spring Web Services. This module is internally used by Spring-WS framework.

- **Spring OXM** – This module provides support classes for XML vs Object Mapping.



2. Spring WS – Environment Setup

In this Chapter, we will understand the process of setting up Spring-WS on Windows and Linux based systems. The Spring-WS can be easily installed and integrated with your current **Java environment** and **MAVEN** by following a few simple steps without any complex setup procedures. User administration is required while installation.

System Requirements

The following table lists out the system requirements, while the subsequent steps will guide us through the environment setup procedure.

JDK	Java SE 2 JDK 1.5 or above
Memory	1 GB RAM (recommended)
Disk Space	No minimum requirement
Operating System Version	Windows XP or above, Linux

Let us now proceed with the steps to install Spring-WS.

Step1 – Verify the Java Installation

To begin with, you need to have Java Software Development Kit (SDK) installed on your system. To verify this, execute any of the following two commands depending on the platform you are working on.

If the Java installation has been done properly, then it will display the current version and specification of your Java installation. A sample output is given in the following table.

Platform	Command	Sample Output
Windows	Open command console and type: \>java -version	Java version "1.7.0_60" Java (TM) SE Run Time Environment (build 1.7.0_60-b19) Java Hotspot (TM) 64-bit Server VM (build 24.60-b09,mixed mode)
Linux	Open command terminal and type: $java -version	java version "1.7.0_25" Open JDK Runtime Environment (rhel-2.3.10.4.el6_4-x86_64) Open JDK 64-Bit Server VM (build 23.7-b01, mixed mode)

- We assume the readers of this tutorial have Java SDK version 1.7.0_60 installed on their system.
- In case you do not have Java SDK, download its current version from – http://www.oracle.com/technetwork/java/javase/downloads/index.html and have it installed.

Step 2: Set your Java Environment

Set the environment variable **JAVA_HOME** to point to the base directory location where Java is installed on your machine.

For example:

Platform	Description
Windows	Set JAVA_HOME to C:\ProgramFiles\java\jdk1.7.0_60
Linux	Export JAVA_HOME=/usr/local/java-current

Append the full path of Java compiler location to the System Path.

Platform	Description
Windows	Append the String "C:\Program Files\Java\jdk1.7.0_60\bin" to the end of the system variable PATH.
Linux	Export PATH=$PATH:$JAVA_HOME/bin/

Execute the command **java -version** from the command prompt as explained above.

Step 3: Download Maven archive

Download Maven 3.3.3 from – http://maven.apache.org/download.cgi

OS	Archive name
Windows	apache-maven-3.3.3-bin.zip
Linux	apache-maven-3.3.3-bin.tar.gz
Mac	apache-maven-3.3.3-bin.tar.gz

Step 4: Extract the Maven archive

Extract the archive, to the directory you wish to install Maven 3.3.3. The subdirectory apache-maven-3.3.3 will be created from the archive.

OS	Location (can be different based on your installation)
Windows	C:\Program Files\Apache Software Foundation\apache-maven-3.3.3
Linux	/usr/local/apache-maven
Mac	/usr/local/apache-maven

Step 5: Set Maven environment variables

Add M2_HOME, M2 and MAVEN_OPTS to the environment variables.

OS	Output

Windows	Set the environment variables using system properties.
	M2_HOME=C:\Program Files\Apache Software Foundation\apache-maven-3.3.3
	M2=%M2_HOME%\bin
	MAVEN_OPTS=-Xms256m -Xmx512m

Linux	Open command terminal and set environment variables. export M2_HOME=/usr/local/apache-maven/apache-maven-3.3.3 export M2=$M2_HOME/bin export MAVEN_OPTS=-Xms256m -Xmx512m
Mac	Open command terminal and set environment variables. export M2_HOME=/usr/local/apache-maven/apache-maven-3.3.3 export M2=$M2_HOME/bin export MAVEN_OPTS=-Xms256m -Xmx512m

Step 6: Add Maven bin directory location to the system path

Now append M2 variable to the System Path.

OS	Output
Windows	Append the string ;%M2% to the end of the system variable, Path.
Linux	export PATH=$M2:$PATH
Mac	export PATH=$M2:$PATH

Step 7: Verify Maven installation

Now open the console, execute the following **mvn** command.

OS	Task	Command
Windows	Open Command Console	c:\> mvn --version
Linux	Open Command Terminal	$ mvn --version
Mac	Open Terminal	machine:< joseph$ mvn --version

Finally, verify the output of the above commands, which should be something as shown below:

OS	Output
Windows	Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06; 2015-04-22T17:27:37+05:30) Maven home: C:\Program Files\Apache Software Foundation\apache-maven-3.3.3 Java version: 1.7.0_75, vendor: Oracle Corporation Java home: C:\Program Files\Java\jdk1.7.0_75\jre Default locale: en_US, platform encoding: Cp1252
Linux	Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06; 2015-04-22T17:27:37+05:30)

	Maven home: /usr/local/apache-maven/apache-maven-3.3.3 Java version: 1.7.0_75, vendor: Oracle Corporation Java home: /usr/local/java-current/jdk1.7.0_75/jre
Mac	Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06; 2015-04-22T17:27:37+05:30) Maven home: /usr/local/apache-maven/apache-maven-3.3.3 Java version: 1.7.0_75, vendor: Oracle Corporation Java home: /Library/Java/Home/jdk1.7.0_75/jre

Step 8 - Setup Eclipse IDE

All the examples in this tutorial have been written using the Eclipse IDE. It is recommended that the readers should have the latest version of Eclipse installed on their machine. To install the Eclipse IDE, download the latest Eclipse binaries from the following link – http://www.eclipse.org/downloads/. Once the installation is downloaded, unpack the binary distribution into a convenient location.

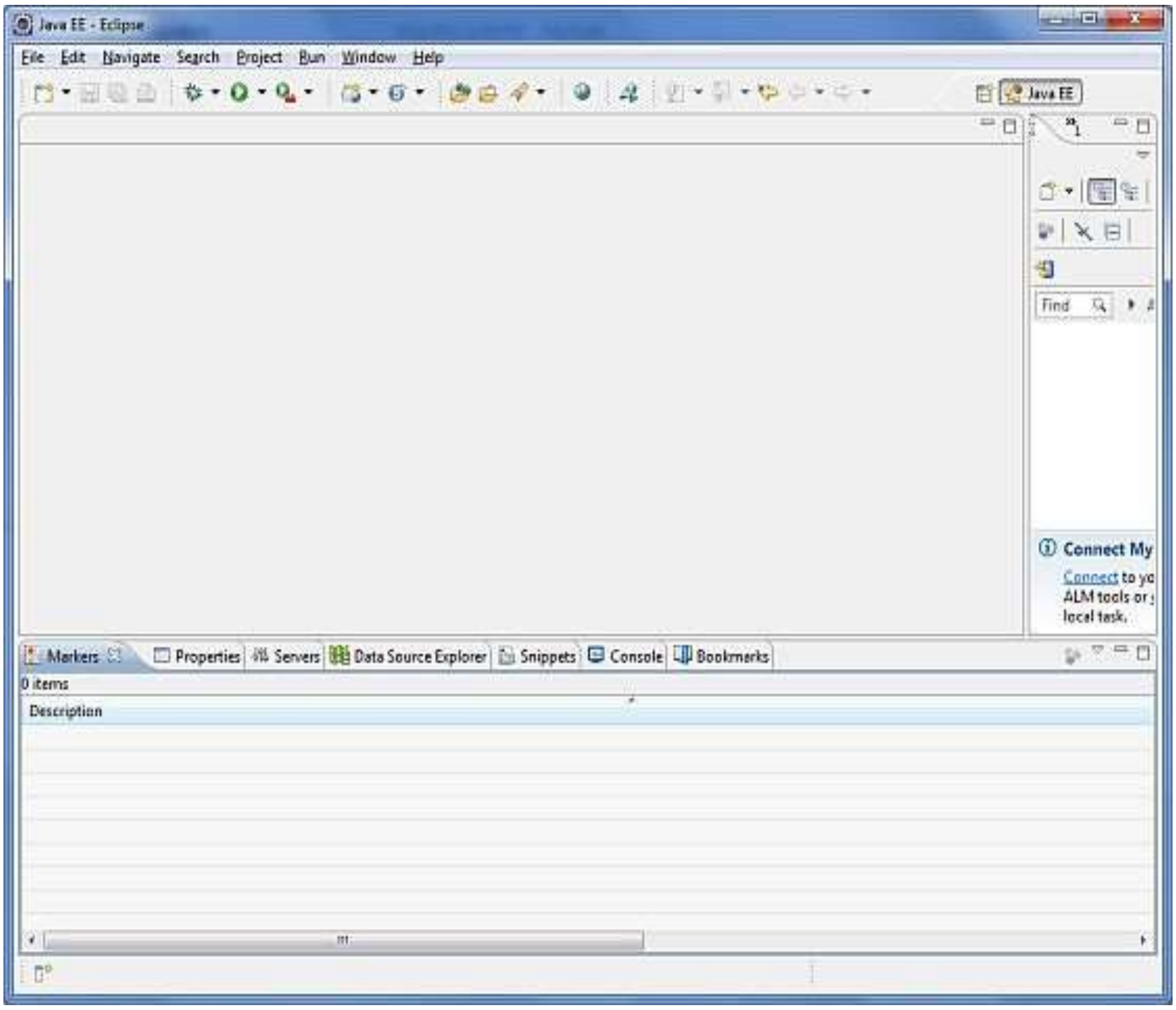
For example in **C:\eclipse** on windows, or **/usr/local/eclipse** on Linux/Unix and finally set the PATH variable appropriately. Eclipse can be started by executing the following commands on the windows machine, or you can simply double click on eclipse.exe.

```
%C:\eclipse\eclipse.exe
```

Eclipse can be started by executing the following commands on the UNIX (Solaris, Linux, etc.) machine:

```
$/usr/local/eclipse/eclipse
```

After a successful startup, if everything is fine then it should display the following screen:



Step 9: Setup Apache Tomcat

We can download the latest version of Tomcat from – http://tomcat.apache.org/. Once the installation is downloaded, unpack the binary distribution into a convenient location. For example in the **C:\apache-tomcat-7.0.59** on a windows machine, or in the **/usr/local/apache-tomcat-7.0.59** on a Linux/Unix machine and then set the **CATALINA_HOME** environment variable pointing to the installation locations.

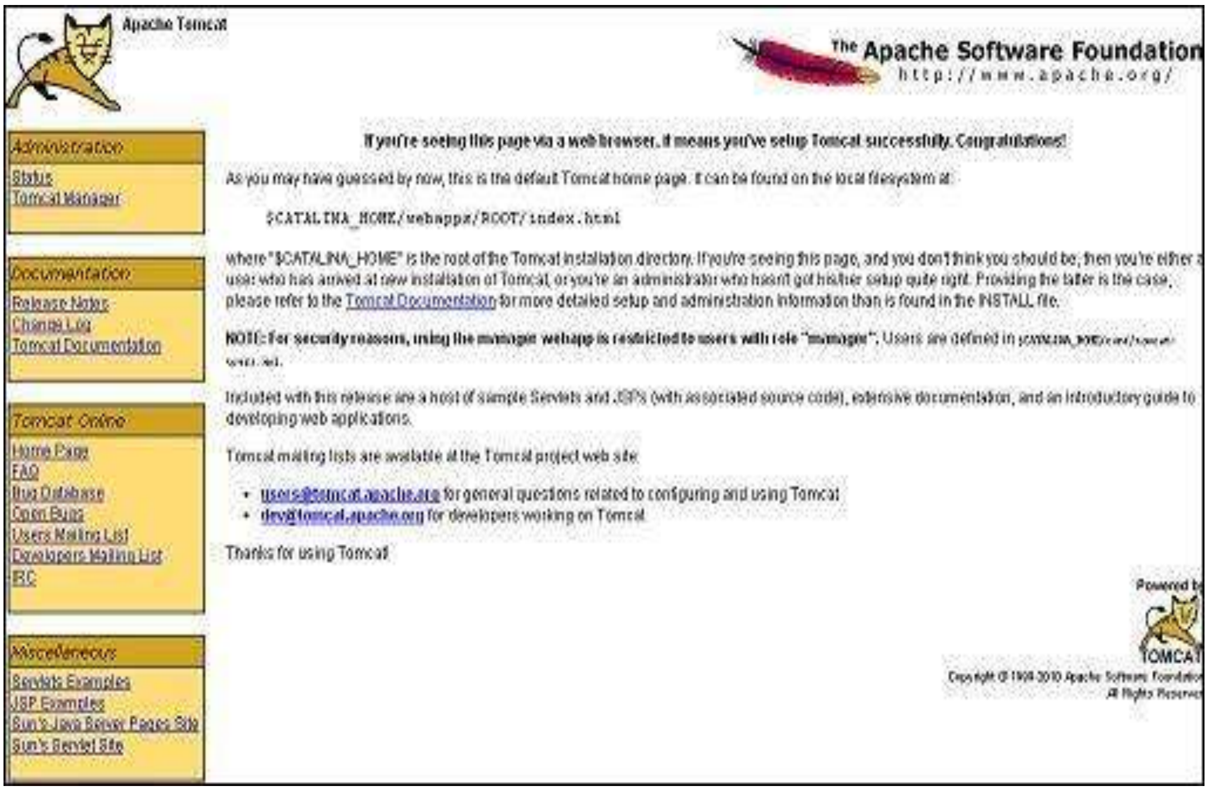
Tomcat can be started by executing the following commands on a windows machine, or you can simply double click on startup.bat

```
%CATALINA_HOME%\bin\startup.bat


or


C:\apache-tomcat-7.0.59\bin\startup.bat
```

Tomcat can be started by executing the following commands on UNIX (Solaris, Linux, etc.) machine:

```
$CATALINA_HOME/bin/startup.sh

or

/usr/local/apache-tomcat-7.0.59/bin/startup.sh
```

After a successful startup, the default web applications included with Tomcat will be available by visiting – **http://localhost:8080/**. If everything is ok, then it should display the following screen:



If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

```
$CATALINA_HOME/webapps/ROOT/index.html
```

where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the Tomcat Documentation for more detailed setup and administration information than is found in the INSTALL file.

NOTE: For security reasons, using the manager webapp is restricted to users with role "manager". Users are defined in $CATALINA_HOME/conf/tomcat-users.xml.

Included with this release are a host of sample Servlets and JSPs (with associated source code), extensive documentation, and an introductory guide to developing web applications.

Tomcat mailing lists are available at the Tomcat project web site:

- users@tomcat.apache.org for general questions related to configuring and using Tomcat
- dev@tomcat.apache.org for developers working on Tomcat

Thanks for using Tomcat!

Copyright © 1999-2010 Apache Software Foundation
All Rights Reserved

Further information about configuring and running Tomcat can be found in the documentation included here, as well as on the Tomcat website – http://tomcat.apache.org.

Tomcat can be stopped by executing the following commands on a windows machine:

```
%CATALINA_HOME%\bin\shutdown

or

C:\apache-tomcat-7.0.59\bin\shutdown
```

Tomcat can be stopped by executing the following commands on the UNIX (Solaris, Linux, etc.) machine:

```
$CATALINA_HOME/bin/shutdown.sh


or


/usr/local/apache-tomcat-7.0.59/bin/shutdown.sh
```

Once we are done with this last step, we are ready to proceed for the first Web Services Example, which we will discuss in the next chapter.

3. Spring WS – First Application

Let us start writing an actual SOAP based web service with Spring-WS Framework. Before we start writing our first example using the Spring-WS framework, we have to ensure that the Spring-WS environment is setup properly as explained in the previous Environment Setup Chapter. We are assuming that the readers have some basic working knowledge with the Eclipse IDE.

Therefore, let us proceed to write a simple Spring WS Application which will expose a web service method to book a leave in an HR Portal.

Contract-first Approach

Spring-WS uses the Contract-first approach, which means we should have our **XML Structures** ready before writing any JAVA based implementation code. We are defining a LeaveRequest Object, which has sub-objects – Leave and Employee.

Following are the required XML constructs:

Leave.xml

```
<Leave xmlns="http://tutorialspoint.com/hr/schemas">
   <StartDate>2016-07-03</StartDate>
   <EndDate>2016-07-07</EndDate>
</Leave>
```

Employee.xml

```
<Employee xmlns="http://tutorialspoint.com/hr/schemas">
   <Number>404</Number>
   <FirstName>Mahesh</FirstName>
   <LastName>Parashar</LastName>
</Employee>
```

LeaveRequest.xml

```
<LeaveRequest xmlns="http://tutorialspoint.com/hr/schemas">
   <Leave>
      <StartDate>2016-07-03</StartDate>
      <EndDate>2016-07-07</EndDate>
   </Leave>
   <Employee>
      <Number>404</Number>
      <FirstName>Mahesh</FirstName>
      <LastName>Parashar</LastName>
   </Employee>
</LeaveRequest>
```

hr.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:hr="http://tutorialspoint.com/hr/schemas"
   elementFormDefault="qualified"
   targetNamespace="http://tutorialspoint.com/hr/schemas">
   <xs:element name="LeaveRequest">
      <xs:complexType>
         <xs:all>
            <xs:element name="Leave" type="hr:LeaveType"/>
            <xs:element name="Employee" type="hr:EmployeeType"/>
         </xs:all>
      </xs:complexType>
   </xs:element>
   <xs:complexType name="LeaveType">
      <xs:sequence>
         <xs:element name="StartDate" type="xs:date"/>
         <xs:element name="EndDate" type="xs:date"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="EmployeeType">
      <xs:sequence>
         <xs:element name="Number" type="xs:integer"/>
         <xs:element name="FirstName" type="xs:string"/>
         <xs:element name="LastName" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

Create the Project

Let us now open a command console, go the C:\MVN directory and execute the following **mvn** command.

```
C:\MVN>mvn archetype:generate -DarchetypeGroupId=org.springframework.ws -
DarchetypeArtifactId=spring-ws-archetype -DgroupId=com.tutorialspoint.hr -
DartifactId=leaveService
```

Maven will start processing and will create the complete Java Application Project Structure.

```
[INFO] Scanning for projects...
[INFO]
[INFO] -----------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----------------------------------------------------------------------
[INFO]
[INFO] Using property: groupId = com.tutorialspoint.hr
[INFO] Using property: artifactId = leaveService
Define value for property 'version':  1.0-SNAPSHOT: :
[INFO] Using property: package = com.tutorialspoint.hr
Confirm properties configuration:
groupId: com.tutorialspoint.hr
artifactId: leaveService
version: 1.0-SNAPSHOT
package: com.tutorialspoint.hr
 Y: :
[INFO] ------------------------------------------------------------------------
---
[INFO] Using following parameters for creating project from Old (1.x) Archetype:
 spring-ws-archetype:2.0.0-M1
[INFO] ------------------------------------------------------------------------
---
[INFO] Parameter: groupId, Value: com.tutorialspoint.hr
[INFO] Parameter: packageName, Value: com.tutorialspoint.hr
[INFO] Parameter: package, Value: com.tutorialspoint.hr
[INFO] Parameter: artifactId, Value: leaveService
[INFO] Parameter: basedir, Value: C:\mvn
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\mvn\leaveService
[INFO] -----------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] -----------------------------------------------------------------------
[INFO] Total time: 35.989 s
[INFO] Finished at: 2017-01-21T11:18:31+05:30
[INFO] Final Memory: 17M/178M
[INFO] -----------------------------------------------------------------------
```

Now go to **C:/MVN** directory. We will see a java application project created named **leaveService** (as specified in artifactId). Update the pom.xml and add

HumanResourceService.java and HumanResourceServiceImpl.java in the following folder – C:\MVN\leaveService\src\main\java\com\tutorialspoint\hr\service folder. Once that is done, then add LeaveEndpoint.java in the following folder – C:\MVN\leaveService\src\main\java\com\tutorialspoint\hr\ws folder.

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
   <project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/maven-v4_0_0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.tutorialspoint.hr</groupId>
   <artifactId>leaveService</artifactId>
   <packaging>war</packaging>
   <version>1.0-SNAPSHOT</version>
   <name>leaveService Spring-WS Application</name>
   <url>http://www.springframework.org/spring-ws</url>
   <build>
      <finalName>leaveService</finalName>
   </build>
   <dependencies>
      <dependency>
         <groupId>org.springframework.ws</groupId>
         <artifactId>spring-ws-core</artifactId>
         <version>2.4.0.RELEASE</version>
      </dependency>
      <dependency>
         <groupId>jdom</groupId>
         <artifactId>jdom</artifactId>
         <version>1.0</version>
      </dependency>
      <dependency>
         <groupId>jaxen</groupId>
         <artifactId>jaxen</artifactId>
         <version>1.1</version>
      </dependency>
      <dependency>
         <groupId>wsdl4j</groupId>
         <artifactId>wsdl4j</artifactId>
         <version>1.6.2</version>
      </dependency>
   </dependencies>
</project>
```

HumanResourceService.java

```
package com.tutorialspoint.hr.service;

import java.util.Date;

public interface HumanResourceService {
   void bookLeave(Date startDate, Date endDate, String name);
}
```

HumanResourceServiceImpl.java

```
package com.tutorialspoint.hr.service;

import java.util.Date;

import org.springframework.stereotype.Service;

@Service
public class HumanResourceServiceImpl implements HumanResourceService {
   public void bookLeave(Date startDate, Date endDate, String name) {
      System.out.println("Booking holiday for [" + startDate + "-" + endDate + "] for
[" + name + "] ");
   }
}
```

LeaveEndpoint.java

```
package com.tutorialspoint.hr.ws;

import java.text.SimpleDateFormat;
import java.util.Date;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.ws.server.endpoint.annotation.Endpoint;
import org.springframework.ws.server.endpoint.annotation.PayloadRoot;
import org.springframework.ws.server.endpoint.annotation.RequestPayload;

import com.tutorialspoint.hr.service.HumanResourceService;
import org.jdom.Element;
import org.jdom.JDOMException;
import org.jdom.Namespace;
import org.jdom.xpath.XPath;

@Endpoint
public class LeaveEndpoint {

private static final String NAMESPACE_URI = "http://tutorialspoint.com/hr/schemas";

private XPath startDateExpression;

private XPath endDateExpression;

private XPath nameExpression;

private HumanResourceService humanResourceService;

   @Autowired
   public LeaveEndpoint(HumanResourceService humanResourceService)

      throws JDOMException {
      this.humanResourceService = humanResourceService;

      Namespace namespace = Namespace.getNamespace("hr", NAMESPACE_URI);

      startDateExpression = XPath.newInstance("//hr:StartDate");
      startDateExpression.addNamespace(namespace);

      endDateExpression = XPath.newInstance("//hr:EndDate");
      endDateExpression.addNamespace(namespace);

      nameExpression = XPath.newInstance("concat(//hr:FirstName,' ',//hr:LastName)");
      nameExpression.addNamespace(namespace);
   }
```

```
    @PayloadRoot(namespace = NAMESPACE_URI, localPart = "LeaveRequest")
    public void handleLeaveRequest(@RequestPayload Element leaveRequest)
       throws Exception {
       SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
       Date startDate = dateFormat.parse(startDateExpression.valueOf(leaveRequest));
       Date endDate = dateFormat.parse(endDateExpression.valueOf(leaveRequest));
       String name = nameExpression.valueOf(leaveRequest);

       humanResourceService.bookLeave(startDate, endDate, name);
    }
}
```

/WEB-INF/spring-ws-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/context"
   xmlns:sws="http://www.springframework.org/schema/web-services"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
   http://www.springframework.org/schema/web-services
   http://www.springframework.org/schema/web-services/web-services-2.0.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:component-scan base-package="com.tutorialspoint.hr"/>
   <bean id="humanResourceService"
class="com.tutorialspoint.hr.service.HumanResourceServiceImpl" />
   <sws:annotation-driven/>

   <sws:dynamic-wsdl id="leave"
      portTypeName="HumanResource"
      locationUri="/leaveService/"
      targetNamespace="http://tutorialspoint.com/hr/definitions">
      <sws:xsd location="/WEB-INF/hr.xsd"/>
   </sws:dynamic-wsdl>
</beans>
```

/WEB-INF/web.xml

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
   version="2.4">

   <display-name>TutorialsPoint HR Leave Service</display-name>

   <servlet>
      <servlet-name>spring-ws</servlet-name>
      <servlet-
class>org.springframework.ws.transport.http.MessageDispatcherServlet</servlet-class>
      <init-param>
         <param-name>transformWsdlLocations</param-name>
         <param-value>true</param-value>
      </init-param>
   </servlet>
   <servlet-mapping>
      <servlet-name>spring-ws</servlet-name>
      <url-pattern>/*</url-pattern>
   </servlet-mapping>
</web-app>
```

/WEB-INF/hr.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:hr="http://tutorialspoint.com/hr/schemas"
   elementFormDefault="qualified"
   targetNamespace="http://tutorialspoint.com/hr/schemas">
   <xs:element name="LeaveRequest">
      <xs:complexType>
         <xs:all>
            <xs:element name="Leave" type="hr:LeaveType"/>
            <xs:element name="Employee" type="hr:EmployeeType"/>
         </xs:all>
      </xs:complexType>
   </xs:element>
   <xs:complexType name="LeaveType">
      <xs:sequence>
         <xs:element name="StartDate" type="xs:date"/>
         <xs:element name="EndDate" type="xs:date"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="EmployeeType">
      <xs:sequence>
         <xs:element name="Number" type="xs:integer"/>
         <xs:element name="FirstName" type="xs:string"/>
         <xs:element name="LastName" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

Build the Project

Let us now open the command console, go the C:\MVN\leaveService directory and execute the following **mvn** command.

```
C:\MVN\leaveService>mvn clean package
```

Maven will start building the project.

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building leaveService Spring-WS Application 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ leaveService ---
[INFO] Deleting C:\mvn\leaveService\target
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ leaveService ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e.
build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ leaveService ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. build is
platform dependent!
[INFO] Compiling 3 source files to C:\mvn\leaveService\target\classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @
leaveService ---
```

```
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,
i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory C:\mvn\leaveService\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ leaveService -
--
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ leaveService ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-war-plugin:2.2:war (default-war) @ leaveService ---
[INFO] Packaging webapp
[INFO] Assembling webapp [leaveService] in [C:\mvn\leaveService\target\leaveService]
[INFO] Processing war project
[INFO] Copying webapp resources [C:\mvn\leaveService\src\main\webapp]
[INFO] Webapp assembled in [7159 msecs]
[INFO] Building war: C:\mvn\leaveService\target\leaveService.war
[INFO] WEB-INF\web.xml already added, skipping
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 19.667 s
[INFO] Finished at: 2017-01-21T11:56:43+05:30
[INFO] Final Memory: 18M/173M
[INFO] ------------------------------------------------------------------------
```

Import Project in Eclipse

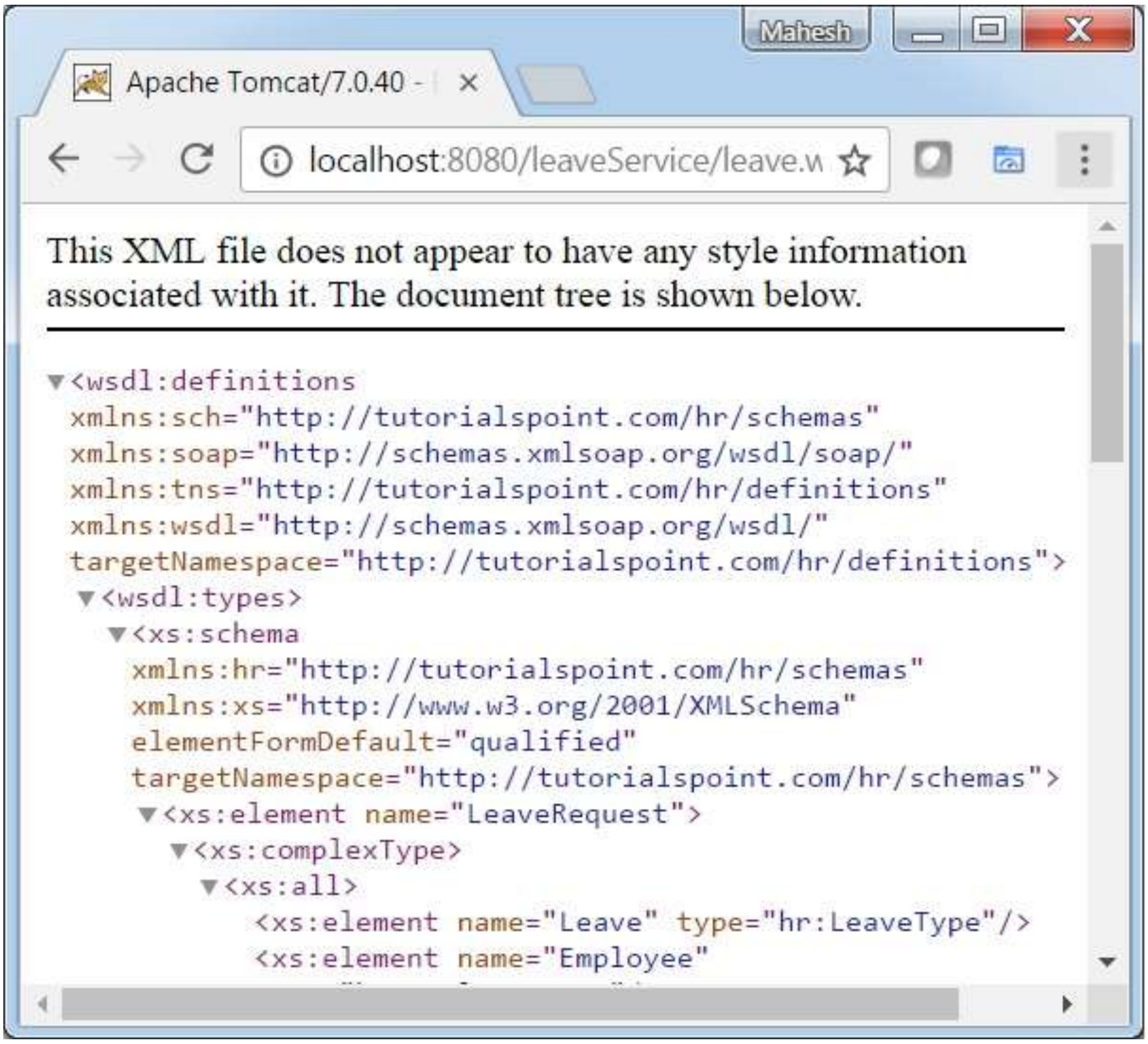
Follow the steps given below to import the project in Eclipse.

- Open Eclipse.
- Select **File → Import →** option.
- Select Maven Projects Option. Click on the Next Button.
- Select Project location, where the **leaveService project** was created using Maven.
- Click on the Finish Button.

Run the Project

Once we are done with creating source and configuration files, export the application. Right click on the application, use Export → WAR File option and save the leaveService.war file in Tomcat's webapps folder.

Start the Tomcat server and ensure we are able to access other webpages from the webapps folder using a standard browser. Try to access the URL – http://localhost:8080/leaveService/leave.wsdl, if everything is ok with the Spring Web Application, we should see the following screen.



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<wsdl:definitions
  xmlns:sch="http://tutorialspoint.com/hr/schemas"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://tutorialspoint.com/hr/definitions"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://tutorialspoint.com/hr/definitions">
  ▼<wsdl:types>
    ▼<xs:schema
      xmlns:hr="http://tutorialspoint.com/hr/schemas"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified"
      targetNamespace="http://tutorialspoint.com/hr/schemas">
      ▼<xs:element name="LeaveRequest">
        ▼<xs:complexType>
          ▼<xs:all>
              <xs:element name="Leave" type="hr:LeaveType"/>
              <xs:element name="Employee"
```

4. Spring WS – Static WSDL

In the previous chapter, we have generated WSDL automatically using the Spring WS Configuration. In this case, we will display how to expose the existing WSDL using the Spring WS.

Step	Description
1	Create a project with a name leaveService under a package com.tutorialspoint as explained in the Spring WS - First Application chapter.
2	Create a WSDL leave.wsdl under the /WEB-INF/wsdl sub-folder.
3	Update spring-ws-servlet.xml under the /WEB-INF sub-folder. We are using the static-wsdl tag here instead of the dynamic-wsdl.
4	The final step is to create content of all source and configuration files and export the application as explained below.

/WEB-INF/spring-ws-servlet.xml

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:schema="http://tutorialspoint.com/hr/schemas"
   xmlns:tns="http://tutorialspoint.com/hr/definitions"
   targetNamespace="http://tutorialspoint.com/hr/definitions">
   <wsdl:types>
      <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
         <xsd:import namespace="http://tutorialspoint.com/hr/schemas"
            schemaLocation="hr.xsd"/>
      </xsd:schema>
   </wsdl:types>
   <wsdl:message name="LeaveRequest">
      <wsdl:part element="schema:LeaveRequest" name="LeaveRequest"/>
   </wsdl:message>
   <wsdl:portType name="HumanResource">
      <wsdl:operation name="Leave">
         <wsdl:input message="tns:LeaveRequest" name="LeaveRequest"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="HumanResourceBinding" type="tns:HumanResource">
      <soap:binding style="document"
         transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="Leave">
         <soap:operation soapAction="http://mycompany.com/RequestLeave"/>
         <wsdl:input name="LeaveRequest">
            <soap:body use="literal"/>
         </wsdl:input>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="HumanResourceService">
      <wsdl:port binding="tns:HumanResourceBinding" name="HumanResourcePort">
         <soap:address location="http://localhost:8080/leaveService/"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

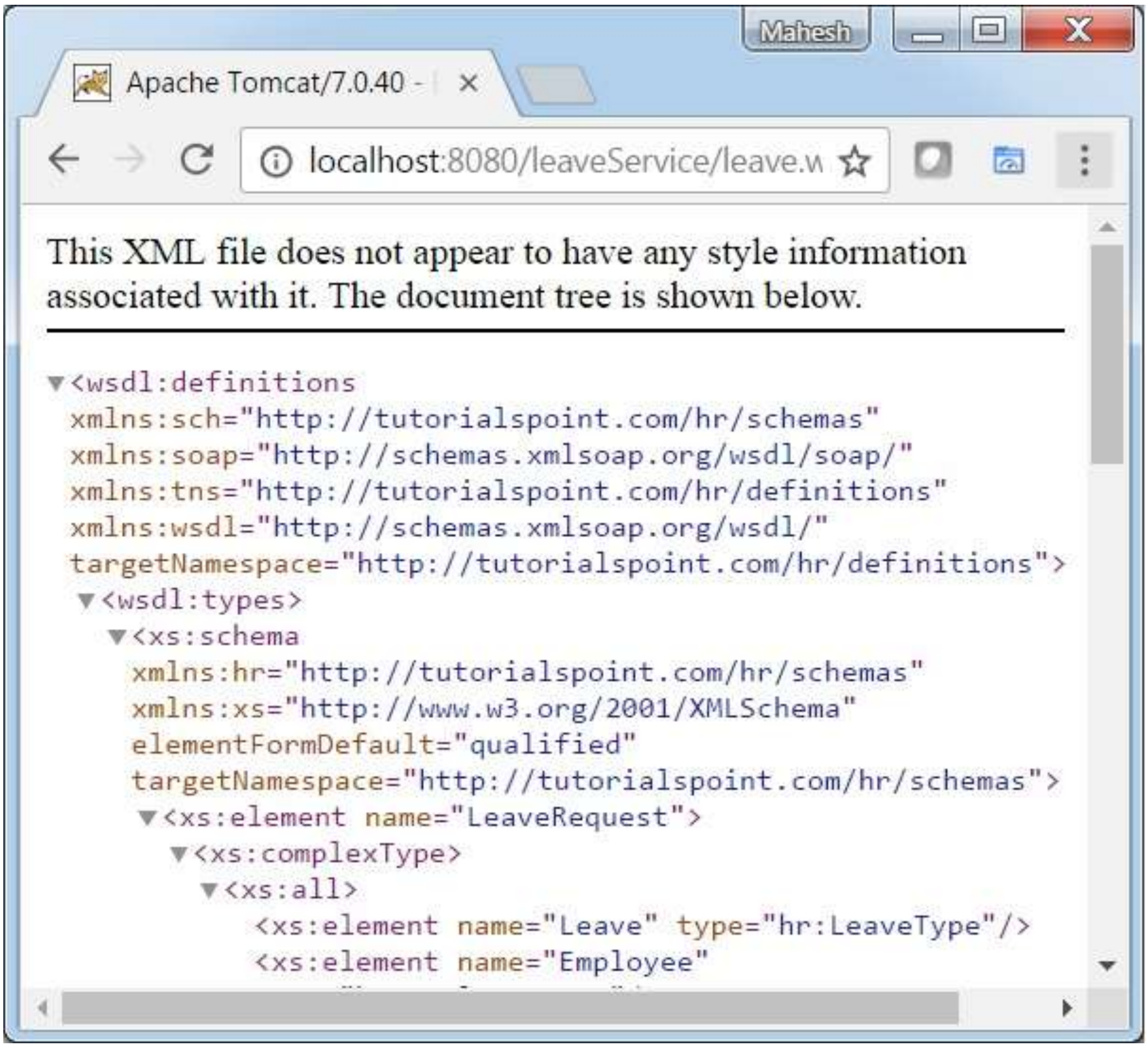
/WEB-INF/spring-ws-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/context"
   xmlns:sws="http://www.springframework.org/schema/web-services"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
   http://www.springframework.org/schema/web-services
   http://www.springframework.org/schema/web-services/web-services-2.0.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:component-scan base-package="com.tutorialspoint.hr"/>

   <sws:annotation-driven/>
   <sws:static-wsdl id="leave" location="/WEB-INF/wsdl/leave.wsdl"/>
   </beans>
```

Run the Project

Once we are done with creating source and configuration files, we should export the application. Right click on the application, use Export → WAR File option and save your leaveService.war file in Tomcat's webapps folder.

Now, start the Tomcat server and ensure that we can access other webpages from the webapps folder using a standard browser. Try to access the URL – http://localhost:8080/leaveService/leave.wsdl, if everything is ok with the Spring Web Application, we will see the following screen.



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<wsdl:definitions
  xmlns:sch="http://tutorialspoint.com/hr/schemas"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://tutorialspoint.com/hr/definitions"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://tutorialspoint.com/hr/definitions">
  ▼<wsdl:types>
    ▼<xs:schema
      xmlns:hr="http://tutorialspoint.com/hr/schemas"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified"
      targetNamespace="http://tutorialspoint.com/hr/schemas">
      ▼<xs:element name="LeaveRequest">
        ▼<xs:complexType>
          ▼<xs:all>
              <xs:element name="Leave" type="hr:LeaveType"/>
              <xs:element name="Employee"
```

5. Spring WS – Writing Server

In this chapter, we will understand how to create a web application server using Spring WS.

Step	Description
1	Create a project with a name countryService under a package com.tutorialspoint as explained in the Spring WS - First Application chapter.
2	Create countries.xsd, domain classes, CountryRepository and CountryEndPoint as explained in the following steps.
3	Update spring-ws-servlet.xml under the /WEB-INF sub-folder.
4	The final step is to create content for all the source and configuration files and export the application as explained below.

countries.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:tns="http://tutorialspoint/schemas"
   targetNamespace="http://tutorialspoint/schemas"
   elementFormDefault="qualified">

   <xs:element name="getCountryRequest">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="name" type="xs:string"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>

   <xs:element name="getCountryResponse">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="country" type="tns:country"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:complexType name="country">
      <xs:sequence>
         <xs:element name="name" type="xs:string"/>
         <xs:element name="population" type="xs:int"/>
         <xs:element name="capital" type="xs:string"/>
         <xs:element name="currency" type="tns:currency"/>
      </xs:sequence>
   </xs:complexType>

   <xs:simpleType name="currency">
      <xs:restriction base="xs:string">
         <xs:enumeration value="GBP"/>
         <xs:enumeration value="USD"/>
         <xs:enumeration value="INR"/>
      </xs:restriction>
   </xs:simpleType>
</xs:schema>
```

Create the Project

Let us open the command console, go the C:\MVN directory and execute the following **mvn** command.

```
C:\MVN>mvn archetype:generate -DarchetypeGroupId=org.springframework.ws -
DarchetypeArtifactId=spring-ws-archetype -DgroupId=com.tutorialspoint -
DartifactId=countryService
```

Maven will start processing and will create the complete Java Application Project Structure.

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] Using property: groupId = com.tutorialspoint
[INFO] Using property: artifactId = countryService
Define value for property 'version':  1.0-SNAPSHOT: :
[INFO] Using property: package = com.tutorialspoint
Confirm properties configuration:
```

```
groupId: com.tutorialspoint
artifactId: countryService
version: 1.0-SNAPSHOT
package: com.tutorialspoint
 Y: :
[INFO] -----------------------------------------------------------------------
---
[INFO] Using following parameters for creating project from Old (1.x) Archetype:
 spring-ws-archetype:2.0.0-M1
[INFO] -----------------------------------------------------------------------
---
[INFO] Parameter: groupId, Value: com.tutorialspoint
[INFO] Parameter: packageName, Value: com.tutorialspoint
[INFO] Parameter: package, Value: com.tutorialspoint
[INFO] Parameter: artifactId, Value: countryService
[INFO] Parameter: basedir, Value: C:\mvn
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\mvn\countryService
[INFO] -----------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] -----------------------------------------------------------------------
[INFO] Total time: 35.989 s
[INFO] Finished at: 2017-01-21T11:18:31+05:30
[INFO] Final Memory: 17M/178M
[INFO] -----------------------------------------------------------------------
```

Now go to C:/MVN directory. We will see a java application project created named countryService (as specified in artifactId). Update the pom.xml.

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
   <project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/maven-v4_0_0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.tutorialspoint.hr</groupId>
   <artifactId>countryService</artifactId>
   <packaging>war</packaging>
   <version>1.0-SNAPSHOT</version>
   <name>countryService Spring-WS Application</name>
   <url>http://www.springframework.org/spring-ws</url>
   <build>
      <finalName>countryService</finalName>
   </build>
   <dependencies>
      <dependency>
         <groupId>org.springframework.ws</groupId>
         <artifactId>spring-ws-core</artifactId>
         <version>2.4.0.RELEASE</version>
      </dependency>
      <dependency>
         <groupId>jdom</groupId>
         <artifactId>jdom</artifactId>
         <version>1.0</version>
      </dependency>
      <dependency>
         <groupId>jaxen</groupId>
         <artifactId>jaxen</artifactId>
         <version>1.1</version>
      </dependency>
      <dependency>
         <groupId>wsdl4j</groupId>
         <artifactId>wsdl4j</artifactId>
         <version>1.6.2</version>
      </dependency>
   </dependencies>
</project>
```

Create Domain Classes

Copy the countries.xsd in C:\mvn\countryService\src\main\resources folder. Let us open the command console, go the C:\mvn\countryService\src\main\resources directory and execute the following **xjc** command to generate domain classes using the countries.xsd.

```
C:\MVN\countryService\src\main\resources>xjc -p com.tutorialspoint countries.xsd
```

Maven will start processing and will create the domain classes in com.tutorialspoint package.

```
parsing a schema...
compiling a schema...
```

```
com\tutorialspoint\Country.java
com\tutorialspoint\Currency.java
com\tutorialspoint\GetCountryRequest.java
com\tutorialspoint\GetCountryResponse.java
com\tutorialspoint\ObjectFactory.java
com\tutorialspoint\package-info.java
```

Create folder java in C:\mvn\countryService\src\main folder. Copy all the classes in the C:\mvn\countryService\src\main\java folder. Create CountryRepository and CountryEndPoint to represent the country database and country server respectively.

CountryRepository.java

```
package com.tutorialspoint;

import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.propertyeditors.CurrencyEditor;
import org.springframework.stereotype.Component;
import org.springframework.util.Assert;

@Component
public class CountryRepository {
   private static final List<Country> countries = new ArrayList<Country>();
   public CountryRepository(){
      initData();
   }

   public void initData() {
      Country us = new Country();
      us.setName("United States");
      us.setCapital("Washington");
      us.setCurrency(Currency.USD);
      us.setPopulation(46704314);

      countries.add(us);

      Country india = new Country();
      india.setName("India");
      india.setCapital("New Delhi");
      india.setCurrency(Currency.INR);
      india.setPopulation(138186860);

      countries.add(india);

      Country uk = new Country();
      uk.setName("United Kingdom");
      uk.setCapital("London");
      uk.setCurrency(Currency.GBP);
      uk.setPopulation(63705000);

      countries.add(uk);
   }
   public Country findCountry(String name) {
      Assert.notNull(name);
      Country result = null;
      for (Country country : countries) {
         if (name.trim().equals(country.getName())) {
            result = country;
         }
      }
      return result;
   }
}
```

CountryEndPoint.java

```
package com.tutorialspoint.ws;

import org.jdom.JDOMException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.ws.server.endpoint.annotation.Endpoint;
import org.springframework.ws.server.endpoint.annotation.PayloadRoot;
import org.springframework.ws.server.endpoint.annotation.RequestPayload;
import org.springframework.ws.server.endpoint.annotation.ResponsePayload;

import com.tutorialspoint.Country;
import com.tutorialspoint.CountryRepository;
import com.tutorialspoint.GetCountryRequest;
import com.tutorialspoint.GetCountryResponse;
```

```
@Endpoint
public class CountryEndPoint {

   private static final String NAMESPACE_URI = "http://tutorialspoint/schemas";

   private CountryRepository countryRepository;

   @Autowired
   public CountryEndPoint(CountryRepository countryRepository) throws JDOMException {
      this.countryRepository = countryRepository;
   }

   @PayloadRoot(namespace = NAMESPACE_URI, localPart = "getCountryRequest")
   @ResponsePayload
   public GetCountryResponse getCountry(@RequestPayload GetCountryRequest request)
throws JDOMException {
      Country country = countryRepository.findCountry(request.getName());
      GetCountryResponse response = new GetCountryResponse();
      response.setCountry(country);
      return response;
   }
}
```

/WEB-INF/spring-ws-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/context"
   xmlns:sws="http://www.springframework.org/schema/web-services"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
   http://www.springframework.org/schema/web-services
   http://www.springframework.org/schema/web-services/web-services-2.0.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:component-scan base-package="com.tutorialspoint"/>
   <sws:annotation-driven/>

   <sws:dynamic-wsdl id="countries"
```

```
      portTypeName="CountriesPort"
      locationUri="/countryService/"
      targetNamespace="http://tutorialspoint.com/definitions">
      <sws:xsd location="/WEB-INF/countries.xsd"/>
   </sws:dynamic-wsdl>
</beans>
```

/WEB-INF/web.xml

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
   version="2.4">

   <display-name>TutorialsPoint Country Service</display-name>

   <servlet>
      <servlet-name>spring-ws</servlet-name>
      <servlet-
class>org.springframework.ws.transport.http.MessageDispatcherServlet</servlet-class>
      <init-param>
         <param-name>transformWsdlLocations</param-name>
         <param-value>true</param-value>
      </init-param>
   </servlet>
   <servlet-mapping>
      <servlet-name>spring-ws</servlet-name>
      <url-pattern>/*</url-pattern>
   </servlet-mapping>
</web-app>
```

Build the Project

Let us open the command console. Go the C:\MVN\countryService directory and execute the following mvn command.

```
C:\MVN\countryService>mvn clean package
```

Maven will start building the project.

```
INFO] Scanning for projects...
```

```
[INFO]

[INFO] ------------------------------------------------------------------------

[INFO] Building countryService Spring-WS Application 1.0-SNAPSHOT

[INFO] ------------------------------------------------------------------------

[INFO]

[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ countryService ---

[INFO] Deleting C:\mvn\countryService\target

[INFO]

[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ countryService --
-

[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e.
build is platform dependent!

[INFO] Copying 0 resource

[INFO]

[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ countryService

---

[INFO] Changes detected - recompiling the module!

[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. build is
platform dependent!

[INFO] Compiling 4 source files to C:\mvn\countryService\target\classes

[INFO]

[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @
countryService ---

[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e.
build is platform dependent!

[INFO] skip non existing resourceDirectory C:\mvn\countryService\src\test\resources

[INFO]

[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ countryService
---

[INFO] No sources to compile

[INFO]

[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ countryService ---


[INFO] No tests to run.

[INFO]

[INFO] --- maven-war-plugin:2.2:war (default-war) @ countryService ---

[INFO] Packaging webapp

[INFO] Assembling webapp [countryService] in
[C:\mvn\countryService\target\countryService]

[INFO] Processing war project

[INFO] Copying webapp resources [C:\mvn\countryService\src\main\webapp]

[INFO] Webapp assembled in [5137 msecs]
```

```
[INFO] Building war: C:\mvn\countryService\target\countryService.war
[INFO] WEB-INF\web.xml already added, skipping
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 16.484 s
[INFO] Finished at: 2017-01-28T09:07:59+05:30
[INFO] Final Memory: 19M/170M
[INFO] ------------------------------------------------------------------------
```

Run the Project

Once we have created the source and configuration files, export the countryService.war file in Tomcat's webapps folder.

Now, start the Tomcat server and ensure if we can access other webpages from the webapps folder using a standard browser. Make a POST request to the URL – http://localhost:8080/countryService/ and by using any SOAP client make the following request.

```
<x:Envelope xmlns:x="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:tns="http://tutorialspoint/schemas">
   <x:Header/>
   <x:Body>
      <tns:getCountryRequest>
         <tns:name>United States</tns:name>
      </tns:getCountryRequest>
   </x:Body>
</x:Envelope>
```

You will see the following result.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header/>
   <SOAP-ENV:Body>
      <ns2:getCountryResponse xmlns:ns2="http://tutorialspoint/schemas">
         <ns2:country>
            <ns2:name>United States</ns2:name>
            <ns2:population>46704314</ns2:population>
            <ns2:capital>Washington</ns2:capital>
            <ns2:currency>USD</ns2:currency>
         </ns2:country>
      </ns2:getCountryResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

6. Spring WS – Unit Test Server

In this chapter, we will understand how to unit test a web application service created by using the Spring WS.

Step	Description
1	Update project countryService created in the Spring WS – Write Server chapter. Add src/test/java folder.
2	Create CustomerEndPointTest.java under the – src/test/java/com/tutorialspoint/ws folder and then update the POM.xml as detailed below.
3	Add spring-context.xml under the src/main/resources sub-folder.
4	The final step is to create content for all the source and configuration files and test the application as explained below.

POM.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/maven-v4_0_0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.tutorialspoint</groupId>
   <artifactId>countryService</artifactId>
   <packaging>war</packaging>
   <version>1.0-SNAPSHOT</version>
   <name>countryService Spring-WS Application</name>
   <url>http://www.springframework.org/spring-ws</url>
   <build>
      <finalName>countryService</finalName>
   </build>
   <dependencies>
      <dependency>
         <groupId>org.springframework.ws</groupId>
         <artifactId>spring-ws-core</artifactId>
```

```
            <version>2.4.0.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>2.5</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.ws</groupId>
            <artifactId>spring-ws-test</artifactId>
            <version>2.4.0.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>3.1.2.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>jdom</groupId>
            <artifactId>jdom</artifactId>
            <version>1.0</version>
        </dependency>
        <dependency>
            <groupId>jaxen</groupId>
            <artifactId>jaxen</artifactId>
            <version>1.1</version>
        </dependency>
        <dependency>
            <groupId>wsdl4j</groupId>
            <artifactId>wsdl4j</artifactId>
            <version>1.6.2</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.5</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

```
</project>
```

spring-context.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/context"
   xmlns:sws="http://www.springframework.org/schema/web-services"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
   http://www.springframework.org/schema/web-services
   http://www.springframework.org/schema/web-services/web-services-2.0.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:component-scan base-package="com.tutorialspoint"/>
   <sws:annotation-driven/>

   <bean id="schema" class="org.springframework.core.io.ClassPathResource">
      <constructor-arg index="0" value="countries.xsd" />
   </bean>
</beans>
```

CustomerEndPointTest.java

```
package com.tutorialspoint.ws;

import javax.xml.transform.Source;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.xml.XmlBeanDefinitionReader;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.GenericApplicationContext;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.ws.test.server.MockWebServiceClient;
import org.springframework.xml.transform.StringSource;
import static org.springframework.ws.test.server.RequestCreators.withPayload;
import static org.springframework.ws.test.server.ResponseMatchers.payload;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration( locations = "/spring-context.xml" )
public class CustomerEndPointTest {

   @Autowired
   private ApplicationContext applicationContext;

   private MockWebServiceClient mockClient;

   @Before
   public void createClient() {
      mockClient = MockWebServiceClient.createClient(applicationContext);
      GenericApplicationContext ctx = (GenericApplicationContext) applicationContext;
       final XmlBeanDefinitionReader definitionReader = new
XmlBeanDefinitionReader(ctx);
      definitionReader.setValidationMode(XmlBeanDefinitionReader.VALIDATION_NONE);
      definitionReader.setNamespaceAware(true);
   }

   @Test
   public void testCountryEndpoint() throws Exception {
      Source requestPayload = new StringSource(
         "<getCountryRequest xmlns='http://tutorialspoint/schemas'>"+
         "<name>United States</name>"+
         "</getCountryRequest>");
      Source responsePayload = new StringSource(
         "<getCountryResponse xmlns='http://tutorialspoint/schemas'>" +
         "<country>" +
         "<name>United States</name>"+
         "<population>46704314</population>"+
         "<capital>Washington</capital>"+
         "<currency>USD</currency>"+
         "</country>"+
         "</getCountryResponse>");

mockClient.sendRequest(withPayload(requestPayload)).andExpect(payload(responsePayload))
;
```

```
    }
}
```

Build the Project

Let us open the command console, go to the C:\MVN\countryService directory and execute the following mvn command.

```
C:\MVN\countryService>mvn test
```

Maven will start building and test the project.

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building countryService Spring-WS Application 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ countryService --
-
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e.
build is platform dependent!
[INFO] Copying 2 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ countryService ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @
countryService ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e.
build is platform dependent!
[INFO] skip non existing resourceDirectory C:\MVN\countryService\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ countryService
---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ countryService ---

[INFO] Surefire report directory: C:\MVN\countryService\target\surefire-reports


-------------------------------------------------------
 T E S T S
```

```
-------------------------------------------------------
Running com.tutorialspoint.ws.CustomerEndPointTest
Feb 27, 2017 11:49:30 AM org.springframework.test.context.TestContextManager
retrieveTestExecutionListeners
INFO: @TestExecutionListeners is not present for class [class com.tutorialspoint
.ws.CustomerEndPointTest]: using defaults.
Feb 27, 2017 11:49:30 AM org.springframework.beans.factory.xml.XmlBeanDefinition
Reader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [spring-context.xml]

Feb 27, 2017 11:49:30 AM org.springframework.context.support.GenericApplicationContext
prepareRefresh
INFO: Refreshing org.springframework.context.support.GenericApplicationContext@b
2eddc0: startup date [Mon Feb 27 11:49:30 IST 2017]; root of context hierarchy
Feb 27, 2017 11:49:31 AM org.springframework.ws.soap.addressing.server.Annotatio
nActionEndpointMapping afterPropertiesSet
INFO: Supporting [WS-Addressing August 2004, WS-Addressing 1.0]
Feb 27, 2017 11:49:31 AM org.springframework.ws.soap.saaj.SaajSoapMessageFactory
 afterPropertiesSet
INFO: Creating SAAJ 1.3 MessageFactory with SOAP 1.1 Protocol
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.386 sec
Feb 27, 2017 11:49:31 AM org.springframework.context.support.GenericApplicationC
ontext doClose
INFO: Closing org.springframework.context.support.GenericApplicationContext@b2ed
dc0: startup date [Mon Feb 27 11:49:30 IST 2017]; root of context hierarchy

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3.517 s
[INFO] Finished at: 2017-02-27T11:49:31+05:30
[INFO] Final Memory: 11M/109M
[INFO] ------------------------------------------------------------------------
```

7. Spring Web Services – Writing Client

In this chapter, we will learn how to create a client for the web application server created in the Spring WS - Writing Server using the Spring WS.

Step	Description
1	Update the project countryService under the package com.tutorialspoint as explained in the Spring WS – Writing Server chapter.
2	Create CountryServiceClient.java under the package com.tutorialspoint.client and MainApp.java under the package com.tutorialspoint as explained in the following steps.

CountryServiceClient.java

```
package com.tutorialspoint.client;


import org.springframework.ws.client.core.support.WebServiceGatewaySupport;


import com.tutorialspoint.GetCountryRequest;
import com.tutorialspoint.GetCountryResponse;


public class CountryServiceClient extends WebServiceGatewaySupport {
   public GetCountryResponse getCountryDetails(String country){
      String uri = "http://localhost:8080/countryService/";
      GetCountryRequest request = new GetCountryRequest();
      request.setName(country);


      GetCountryResponse response =(GetCountryResponse)
getWebServiceTemplate().marshalSendAndReceive(uri, request);
      return response;
   }
}
```

MainApp.java

```
package com.tutorialspoint;

import org.springframework.oxm.jaxb.Jaxb2Marshaller;

import com.tutorialspoint.client.CountryServiceClient;

public class MainApp {
   public static void main(String[] args) {
      CountryServiceClient client = new CountryServiceClient();

      Jaxb2Marshaller marshaller = new Jaxb2Marshaller();
      marshaller.setContextPath("com.tutorialspoint");
      client.setMarshaller(marshaller);
      client.setUnmarshaller(marshaller);
      GetCountryResponse response = client.getCountryDetails("United States");

      System.out.println("Country : " + response.getCountry().getName());
      System.out.println("Capital : " + response.getCountry().getCapital());
      System.out.println("Population : " + response.getCountry().getPopulation());
      System.out.println("Currency : " + response.getCountry().getCurrency());
   }
}
```

Start the Web Service

Start the Tomcat server and ensure that we can access other webpages from the webapps folder using a standard browser.

Test Web Service Client

Right click on the MainApp.java in your application under Eclipse and use **run as Java Application** command. If everything is ok with the application, it will print the following message.

```
Country : United States
Capital : Washington
Population : 46704314
Currency : USD
```

Here, we have created a Client – **CountryServiceClient.java** for the SOAP based web service. MainApp uses CountryServiceClient to make a hit to the web service, makes a post request and gets the data.

8. Spring WS – Unit Test Client

In this chapter, we will learn how to unit test a client created in the Spring WS – Writing Client chapter for the web application server created in the Spring WS – Writing Server chapter using the Spring WS.

Step	Description
1	Update the project countryService under the package com.tutorialspoint as explained in the Spring WS - Writing Server chapter.
2	Create CountryServiceClientTest.java under the package com.tutorialspoint under folder SRC → Test → Java as explained in steps given below.

CountryServiceClientTest.java

```
package com.tutorialspoint;

import static org.junit.Assert.*;

import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import org.springframework.oxm.jaxb.Jaxb2Marshaller;

import com.tutorialspoint.client.CountryServiceClient;

public class CountryServiceClientTest {
   CountryServiceClient client;

   @Before
   public void setUp() throws Exception {
      client = new CountryServiceClient();

      Jaxb2Marshaller marshaller = new Jaxb2Marshaller();
      marshaller.setContextPath("com.tutorialspoint");
      client.setMarshaller(marshaller);
      client.setUnmarshaller(marshaller);
   }
   @Test
   public void test() {
      GetCountryResponse response = client.getCountryDetails("United States");
      Country expectedCountry = new Country();
      expectedCountry.setCapital("Washington");
      Country actualCountry = response.getCountry();
      Assert.assertEquals(expectedCountry.getCapital(), actualCountry.getCapital());
   }
}
```

Start the Web Service

Start the Tomcat server and ensure we are able to access other webpages from the webapps folder using a standard browser.

Unit Test Web Service Client

Let us open the command console, go to the C:\MVN\countryService directory and execute the following mvn command.

```
C:\MVN\countryService>mvn test
```

Maven will start building and testing the project.

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building countryService Spring-WS Application 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] ---maven-resources-plugin:2.6:resources (default-resources) @ countryService ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,
i.e. build is platform dependent!
[INFO] Copying 2 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ countryService ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. build is
platform dependent!
[INFO] Compiling 10 source files to C:\MVN\countryService\target\classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @
countryService ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,
i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory C:\MVN\countryService\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ country
Service ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. build is
platform dependent!
[INFO] Compiling 2 source files to C:\MVN\countryService\target\test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ countryService ---

[INFO] Surefire report directory: C:\MVN\countryService\target\surefire-reports


-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.tutorialspoint.CountryServiceClientTest
Feb 27, 2017 8:45:26 PM org.springframework.ws.soap.saaj.SaajSoapMessageFactory
afterPropertiesSet
INFO: Creating SAAJ 1.3 MessageFactory with SOAP 1.1 Protocol
Feb 27, 2017 8:45:26 PM org.springframework.oxm.jaxb.Jaxb2Marshaller
createJaxbContextFromContextPath
INFO: Creating JAXBContext with context path [com.tutorialspoint]
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.457 sec
Running com.tutorialspoint.ws.CustomerEndPointTest
Feb 27, 2017 8:45:27 PM org.springframework.test.context.TestContextManager
retrieveTestExecutionListeners
INFO: @TestExecutionListeners is not present for class [class com.tutorialspoint
.ws.CustomerEndPointTest]: using defaults.
Feb 27, 2017 8:45:27 PM org.springframework.beans.factory.xml.XmlBeanDefinitionR
eader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [spring-context.xml]

Feb 27, 2017 8:45:27 PM org.springframework.context.support.GenericApplicationContext
prepareRefresh
INFO: Refreshing org.springframework.context.support.GenericApplicationContext@5
17c642: startup date [Mon Feb 27 20:45:27 IST 2017]; root of context hierarchy
Feb 27, 2017 8:45:28 PM org.springframework.ws.soap.addressing.server.Annotation
ActionEndpointMapping afterPropertiesSet
INFO: Supporting [WS-Addressing August 2004, WS-Addressing 1.0]
```

```
Feb 27, 2017 8:45:28 PM org.springframework.ws.soap.saaj.SaajSoapMessageFactory
afterPropertiesSet
INFO: Creating SAAJ 1.3 MessageFactory with SOAP 1.1 Protocol
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.243 sec
Feb 27, 2017 8:45:28 PM org.springframework.context.support.GenericApplicationContext
doClose
INFO: Closing org.springframework.context.support.GenericApplicationContext@517c
642: startup date [Mon Feb 27 20:45:27 IST 2017]; root of context hierarchy

Results :

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 5.686 s
[INFO] Finished at: 2017-02-27T20:45:28+05:30
[INFO] Final Memory: 17M/173M
[INFO] ------------------------------------------------------------------------
```