Testing Soap Service in Orasi Java Toolkit

## **This tutorial is designed to help a tester better understand how soapUI and Java/Selenium programming can be used in combination to create robust tests with web services.**

## Necessary Tools:

Before getting started there a few things you will need to access. In order to start this tutorial, you will need the free version of soapUI, an IDE (Eclipse is used in this demo), and you will want to access the API Demo, an Orasi repository. You will also be using the services provided at <http://www.webservicex.net>.

# Starting with core classes in API Demo

## SoapService:

The SoapService class is the primary class in which all interactions with Soap service are handled. Everything needed to form the Soap Request and validate the Soap Response is handled here. This class is not intended to be directly used by the test classes; instead, it is used as a Parent class or an extension to other classes which we will get into later.

## SoapServiceCommands:

The SoapServiceCommands class allows easy access to internal runtime commands that are used when you build the Soap Request by XPath. A sample of these commands are GetDate which will get the current date or RemoveNode which will remove the current XML node and all of its children from the Request.

# Creating classes for Service and Operations

For this example, we will be using three public webservice’s from the WebService website given at the top; specifically, the USZip service, the Australian Postcode service, and the UK location service which are all located in the standards and lookup data section. We want to use these three similar services to aid in the construction of proper hierarchal file management which will mimic how an actual professional project is set up.

**The three WSDL’s used:**

USZIP - <http://www.webservicex.net/uszip.asmx?wsdl>

This is a service that uses that Area Code, City Name, State Abbreviation or a Zip code to return relevant data from the input.

Australian Postcode - <http://www.webservicex.net/AustralianPostCode.asmx?WSDL>

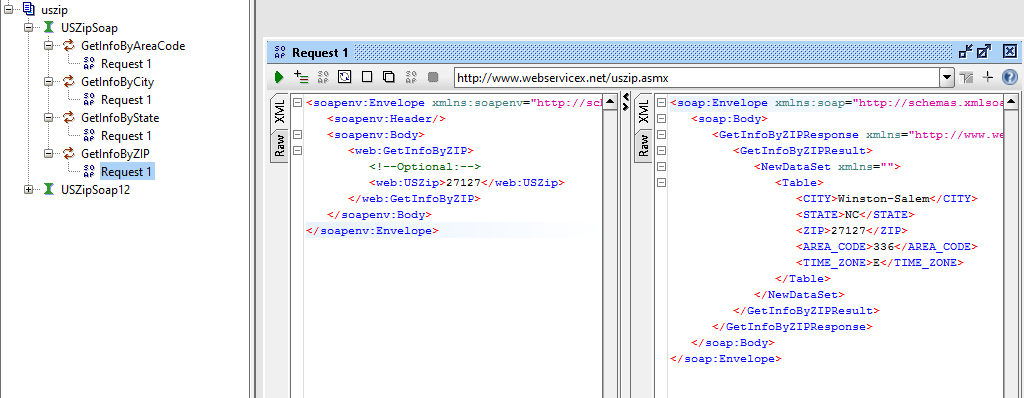
This is a service that uses a Post Code and an Australian location to return relevant data from the input.

UK Location - <http://www.webservicex.net/uklocation.asmx?WSDL>

This is a service that uses County, Post Code, Town or a UK Address to return relevant data from the input.

## SoapUI:

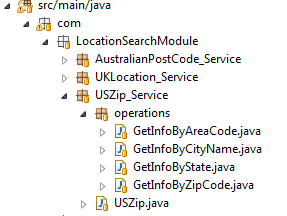
From SoapUI, this is what our sample looks like when we create a new Soap Project with the USZip wsdl. The other services provided will look similar. The project is built up with the main project folder at top followed by a list of soap requests below it. Using the GetInfoByZip operation you can input the Zip Code into the request xml, and upon pressing play the response xml will return city information for that Zip Code.



Our goal is to take this same process, and convert it into a Java format.

## Module package and Service Class’s:

In this example we will be dealing with three services in order to simulate enterprise level services where there can be multiple services that all fall under a single category. In this case our three services all deal with gathering location information. If you are working directly in the Orasi Toolkit, first create a new module package in src/main/java/com called “LocationSearchModule”. This is where all of our service java classes will go. Then you want to create a new package inside of the module for each of the three services. These will be your “Service Packages”. They will contain two objects, a class called USZip.java, UKLocation.java, or AustralianPostCode.java and a package called operations which will house the classes that represent each of the soap requests you would see in SoapUI.



Since we are dealing with a single environment, the USZip class is very simple. In an enterprise field, you may need to have this class handle multiple environments, but for the purpose of this demonstration, we will keep it simple. All you need to do is extend this class into SoapService, and set the Service Name and Service URL to be used. The Service Name in this case is a tracking device used for reporting purposes, while the Service URL is going to be the endpoint triggered and where requests are sent to.

**import** com.orasi.api.soapServices.core.SoapService;

**public** **class** USZip **extends** SoapService{

**public** USZip() {

setServiceName("UsZipService");

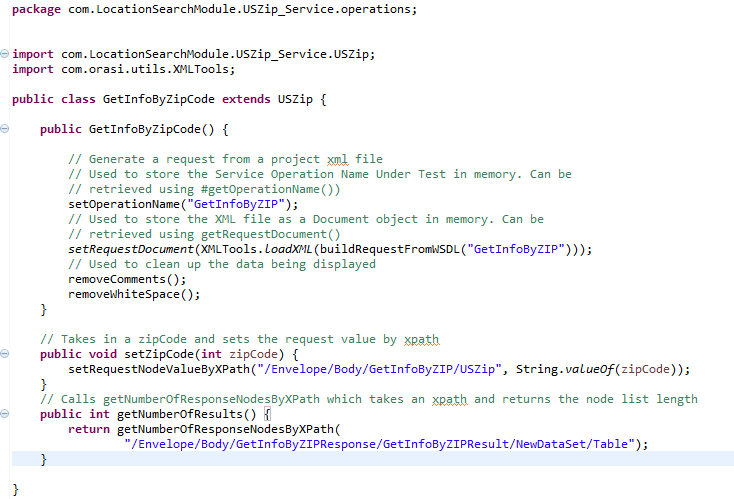
setServiceURL("http://www.webservicex.net/uszip.asmx?wsdl");

}

}

## Operations Class:

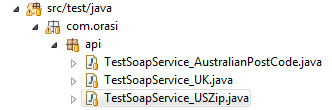
You will now want to start creating your classes under the operations package. These again are designed to mimic the soap requests that you can run in the SoapUI. For each class you will first want to extend off of the WSDL building class which in this case is the USZip class. This will allow you to use the available methods in the SoapServices class in each request class. A GetInfoByZipCode constructor will set an operation name for recording and it will also build a section of the wsdl; specifically, in this case the section that requests info by Zip code. There are two clean up methods that allot the information to be presented in a more readable format. You will also want to create a setZipCode method that finds the zip code request node and inserts a zip into the zip field as well as a GetNumberOfResuts method that uses the getNumberOfResponseNodesByXPath method to return the number of response nodes. This is used as an assertion of sorts when running a test to verify there is response data. Each request class will have the same basic structure as this example.



## TestNG Class:

The final step to tie all of this together will be in the creation of a testNG java class. This class will act like the TestSuite you can create in SoapUI that allows the user to run multiple requests with assertions all at once. We can call it TestSoapService\_USZip.

This is where you will store the testNG class:



Inside each @Test one individual request is created. The first in the example below is an area code test. Inside the method areaCodeTest an GetInfoByAreaCode object called getInfo is instantiated. We use this to access the setAreaCode( ) method as well as the sendRequest( ) method. The TestReporter.LogAPI function checks if any response nodes exist and also displays the getInfo object which is the response node list as long as the request value is correct and finds response information.

