Automate web service testing, Part 1: Create a web service using IBM Rational Software Architect

Mohan Jadhav
Mansoor Ahmed

February 20, 2007

This series of tutorials demonstrates how to automate the testing of a typical web service using an array of technologies such as JUnit, Apache Commons HttpClient and Apache XMLUnit. In this first part, you'll create a simple web service using IBM® Rational® Software Architect.

Before you start

About this series

Web services are increasingly becoming key components in business applications, and with the emergence of paradigms such as Service-Oriented Architecture (SOA), the focus on web services continues to grow. In these service-oriented and service-critical applications, testing of web services assumes equal significance. Rapid creation and deployment of complex web services offers a challenge to the QA team. Automating web service testing can help the team manage the effort efficiently.

Web service testing typically includes the following tasks:

1. Generate the client or skeleton code for the web service
2. Define the test inputs
3. Invoke the web service using the client or skeleton code
4. Verify the actual response is similar to that of the expected response

Of those activities, client or skeleton code generation and response verification require the most effort.

Various tools are available to generate the client or skeleton code for a web service based on its WSDL file, however, the code generated might be proprietary and might need to be regenerated every time the WSDL changes. A solution to this problem is to eliminate the generation of the client or skeleton code and invoke the web service directly through a generic HTTP client. Later in the series, we will see one such toolkit provided by Apache Commons HttpClient API.
Similarly, response verification usually involves some kind of manual intervention, depending on the complexity of the response content. If the response contains simple elements, verification can be simple since it requires you to just check the value in the simple element. For responses which contain complex elements and a huge list of such complex elements, manual verification can require much more effort.

This series introduces you to a technique to automate the testing of a typical web service using an array of technologies such as JUnit, Apache Commons HttpClient (hereafter referred to as HttpClient) and Apache XMLUnit (hereafter referred to as XMLUnit). The technique is demonstrated on the development platform offered by IBM® Rational® Software Architect.

This series has two parts:

- **Part 1** demonstrates how to create a simple web service with IBM Rational Software Architect.
- **Part 2** introduces the XMLUnit, which offers the API to compare two files in XML form.

**About this tutorial**

This tutorial is Part 1 of the series on how to automate the testing of a typical web service, using an array of technologies such as JUnit, Apache Commons HttpClient and Apache XMLUnit. We'll describe the steps involved in creating a simple web service using IBM Rational Software Architect, which offers a sophisticated set of features in an integrated development platform.

**Objectives**

In this tutorial, you learn how to create and test a simple web service using IBM Rational Software Architect.

**Prerequisites**

It is presumed that you have a basic understanding of the web service, its development in Java™, and have some exposure to unit testing tools such as JUnit.

**System requirements**

Your system should have the following:

2. IBM Rational Software Architect, version 6.x

**Create a web service**

**Sample scenario**

We will create a sample web service with a single method that returns a list of users. Each user will have a name and an address. For the sake of simplicity, we will make the web service return a
static list of users. In the real scenario, this list might come from a data source such as a database or a file.

The XML schema for the user is given in Listing 1.

**Listing 1**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="user">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="name"/>
        <xsd:element ref="address"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="street"/>
  <xsd:element name="address">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="street"/>
        <xsd:element ref="city"/>
        <xsd:element ref="zip"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="zip"/>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="city">
    <xsd:complexType/>
  </xsd:element>
</xsd:schema>
```

To test the sample web service:

- Create a web project for the web service.
- Create a Java project that will contain the JUnit test code for testing the web service.
- Use the HttpClient to invoke the web service.
- Get the response from the web service and store it in a file.
- Use the XMLUnit API to compare the actual response with the expected response.

The following sections illustrate these tasks.

**Create the web service**

The first step is to create the web service.

- Open IBM Rational Software Architect. Go to the menu **Window > Open Perspective > web.**
- Go to **File > New > Project** and select **Dynamic Web Project** in the wizard as shown in Figure 1.
Automate web service testing, Part 1: Create a web service using IBM Rational Software Architect

**Figure 1. Create dynamic web project**

- Click on **Next**.
- Enter a project name, such as SampleProject, for the project name, as shown in Figure 2.

**Figure 2. Dynamic web project name**

- Click on **Finish**.
- Upon creating the project, the project explorer view should look like Figure 3.
Figure 3. View of the web project

- Right-click on JavSource under the SampleProject\Java Resources folder in the project explorer view and select New > Package.
- Enter the name of the package, such as com.ibm.ws.sample, in the Java Package page, as shown in Figure 4.

Figure 4. Java package

- Click on Finish.
- Right-click on the package com.ibm.ws.sample and select New > Class to create Java classes named Address.java, User.java and Users.java. See listings 2, 3, 4 below for the source code for these classes.
Listing 2. Address.java

```java
package com.ibm.ws.sample;

public class Address {
    private String street;
    private String city;
    private String zip;

    // Getters
    public String getStreet() { return street; }
    public String getCity() { return city; }
    public String getZip() { return zip; }

    // Setters
    public void setStreet(String street) { this.street = street; }
    public void setCity(String city) { this.city = city; }
    public void setZip(String zip) { this.zip = zip; }
}
```

Listing 3. User.java

```java
package com.ibm.ws.sample;

public class User {
    private String name;
    private Address address;

    // Getters
    public String getName() { return name; }
    public Address getAddress() { return address; }

    // Setters
    public void setName(String name) { this.name = name; }
    public void setAddress(Address address) { this.address = address; }
}
```

Listing 4. Users.java

```java
package com.ibm.ws.sample;

public class Users {
    /**
     * Returns all the users.
     * The list of users might come from a database
     * in the real scenario.
     *
     * @return User[]
     */
    public User[] getUsers() {
        User[] users = new User[3];
        users[0] = getUser("John", "Park Street", "Washington", "012345");
        users[1] = getUser("Mohan", "Avenue Street", "Bangalore", "456789");
        return users;
    }

    /**
     * Returns a User object
     * 
     * @param name
     * @param street
     * @param city
     * @param zip
     * @return User
     */
    private User getUser(String name, String street, String city, String zip) {
        Address address = new Address();
        address.setStreet(street);
        address.setCity(city);
        address.setZip(zip);
        return new User(name, address);
    }
}
```
/* @param name 
 * @param street 
 * @param city 
 * @param zip 
 * @return User 
 */
private User getUser(String name, String street, String city, String zip) 
{ 
    Address address = new Address();
    address.setStreet(street);
    address.setCity(city);
    address.setZip(zip);
    User user = new User();
    user.setName(name);
    user.setAddress(address);
    return user;
}

• After creating the Java classes, the project explorer should look like Figure 5.

Figure 5. Explorer View

• Now, we will make the web service out of the Users.java. Right-click on the Users.java and select **Web Services > Create Web service**. If you don't see this menu option, you should enable the Web Services capability through menu **Window > Preferences > Workbench > Capabilities > Web Service Developer**.

• In the **Web Services** page, uncheck the option for **Start Web service in Web project** and click on **Next** as shown in Figure 6.
Figure 6. Web services

- In the **Object Selection** page, ensure that the **Bean** field contains the value `com.ibm.ws.sample.Users` as shown in Figure 7.

Figure 7. Object selection page

- Click on **Next** in the **Service Deployment Configuration** page without changing the default values, as shown in Figure 8.

Figure 8. Object selection page
• Click on **Next** in the **Service Endpoint Interface Selection** page without changing the default values, as shown in Figure 9.

**Figure 9. Service Endpoint Interface Selection**

![Image of Service Endpoint Interface Selection]

• Click on **Next** in the **Web Service Java Bean Identity** page, ensure that the method getUsers() is selected, as shown in Figure 10.

**Figure 10. Web Service Java Bean Identity**

![Image of Web Service Java Bean Identity]

• In the **Web Service Publication** page, click on **Finish** without changing the default values, as shown in Figure 11.
Figure 11. Web Service Publication

- The Project Explorer view should now look like Figure 12.

Figure 12. Project Explorer

Deploy the web service

Now, that we created the web service, we will deploy the application in the server.

- Right-click on the project name, SampleProject and select Run > Run on Server..., as shown in Figure 13.
Figure 13. Run on Server

- In the Define a New Server page, click on Next without changing the default values, as shown in Figure 14.

Figure 14. Define a new server

- In the Add and Remove Projects page, ensure that SampleProjectEAR is added to the Configured projects section, as shown in Figure 15.
Figure 15. Add and remove projects

- Click on **Finish**.

Now the application will be deployed in the server. You can verify whether the web service is running or not by accessing the URL http://localhost:9080/SampleProject/services/Users in the browser.

**Conclusion**

In this tutorial, you have created a simple web service on the development platform offered by the Rational Software Architect toolset. In Part 2 of the series, you will learn a technique to automate the testing of the web service output using HttpClient and XMLUnit APIs.
Related topic

- Learn more about IBM Rational Software Architect.

© Copyright IBM Corporation 2007
Trademark
(www.ibm.com/developerworks/ibm/trademarks/)