Chapter No. 4
"Working with Your First TestSuite"
In this package, you will find:
A Biography of the author of the book
A preview chapter from the book, Chapter NO.4 "Working with Your First TestSuite"
A synopsis of the book’s content
Information on where to buy this book

About the Author

Charitha Kankanamge is Manager, Quality Assurance and Senior Technical Lead at WSO2 with more than 9 years of experience in Software Quality Assurance. Charitha is specialized in SOA and middleware testing. He lead the WSO2 QA team since 2007. He is also a committer of the Apache Software Foundation contributing to Apache web services project. Charitha is interested in researching new technologies in software-testing space as well as new trends in agile and exploratory testing processes.

Prior to joining WSO2, Charitha has worked at Virtusa inc. for 3 years where he was involved in multiple on-site and off-shore project assignments. In his rare offline moments, he enjoys playing guitar and watching movies.

Charitha has been involved in reviewing two books, *Apache Jmeter, Emilly H. Halili and Quick Start Apache Axis2, Deepal Jayasinghe* both being published by Packt Publishing.

Charitha can be reached through his blog:
http://charithaka.blogspot.com

For More Information:
www.packtpub.com/web-services-testing-with-soapui/book
Web Services Testing with soapUI

This book is all about using soapUI for functional and performance testing of service-oriented solutions. soapUI can be used to test various aspects of a service-oriented solution without merely playing the role of a web service invocation tool. We will follow a simple tutorial-style approach throughout the book in which we will explore all key features provided by soapUI based on a sample web services project. This book is ideally designed to guide readers to get more detailed insight on soapUI by doing a lot of hands-on exercises.

What This Book Covers

Chapter 1, Web Services Testing and soapUI, introduces soapUI by giving an overview of its history, features, and installation of soapUI in your computer. We will begin our journey towards learning soapUI by discussing some key characteristics of SOA, Web services and Web services testing in general.

Chapter 2, The Sample Project, introduces the sample web services project which will be used as the target application for functional and performance testing in the remaining chapters of the book. In this chapter, we will build a simple web services based application using Apache Axis2 open source web services framework. The primary objective of building this sample application is to use it in all demonstrations of soapUI features. As we will not discuss any topics related to soapUI or web services testing in general in this chapter, you may skip the details and download the sample web services project from http://www.PacktPub.com/support.

Chapter 3, First Steps with soapUI and Projects, serves as a guide for getting started with soapUI projects. Based on one of the web services that we built as part of the sample web services project in Chapter 2, The Sample Project, we will discuss the schema and WSDL of the web service in detail. We will use soapUI to invoke the operations of sample web service and discuss the SOAP requests, responses, and faults.

Chapter 4, Working with Your First TestSuite, demonstrates the basic constructs of a soapUI project—TestSuites, TestCases, and TestSteps—which prepares you for the next chapters of the book. We will also look into the validation of responses using assertions and soapUI properties.

Chapter 5, Load and Performance Testing with soapUI, covers the steps that you would have to follow when using soapUI as a load and performance testing tool. We will demonstrate the load test strategies provided by soapUI and the load test specific assertions.

For More Information:
www.packtpub.com/web-services-testing-with-soapui/book
Chapter 6, *Web Services Simulation with soapUI*, briefly describes how web services can be simulated using soapUI. We will demonstrate the usage of soapUI mock services model and static as well as dynamic mock responses.

Chapter 7, *Advanced Functional Testing with soapUI*, introduces the testing aspects of web services extensions such as WS-Security and WS-Addressing. We will use an improved version of the sample web services project which we built in Chapter 2, The Sample Project for the demonstrations in this chapter.

Chapter 8, *Getting Started with REST Testing*, introduces the concepts related to RESTful web services and how soapUI can be utilized in RESTful services testing. We will demonstrate the use of soapUI in RESTful services testing by using a publicly hosted sample web application.

Chapter 9, *Testing Databases with soapUI*, briefly describes the direct database query invocations of soapUI. In this chapter, we will discuss the database testing features provided by soapUI such as JDBC requests and assertions.

Chapter 10, *JMS Testing with soapUI*, demonstrates the use of JMS in soapUI. By exposing one of the sample web services over JMS transport, we will explore the JMS testing capabilities provided by soapUI.

Chapter 11, *Extending soapUI with Scripting*, introduces the scripting facilities given by soapUI in order to extend the default behavior of soapUI tests. We will look into the use of soapUI API methods through Groovy scripts inside our tests.

Chapter 12, *Automated Testing with soapUI*, demonstrates various automated testing approaches with soapUI. In this chapter, we will discuss the integration of soapUI tests with build tools such as Apache Maven.

Chapter 13, *Miscellaneous Topics*, introduces some useful tools integrated with soapUI such as WS-I validation tool and the utilities provided by external web services framework such as Apache Axis2. This chapter also demonstrates the use of soapUI when testing services by sending attachments.

For More Information:
www.packtpub.com/web-services-testing-with-soapui/book
Working with Your First TestSuite

In the previous chapter, we discussed the basics of soapUI projects. When you are testing an individual service or a complete service-oriented solution, it is not just sufficient to create a project with multiple SOAP requests and manually execute and validate the responses of them one by one. Instead, we should follow some mechanism to execute our tests in a well organized manner, so that, we could minimize some unnecessary delays and focus on achieving a greater level of test coverage.

soapUI TestSuites are one of the key building blocks in a project which allow us to structure and execute functional tests. In this chapter, we will look into the following high-level topics of building a comprehensive test suite in order to test our sample hotel room reservation system:

- Creating a TestSuite
- Running the TestSuite
- Adding test assertions
- Adding properties to the tests

A sample test scenario

We have built a sample hotel reservation system in Chapter 2, The Sample Project, and invoked one of its services in Chapter 3, First Steps with soapUI and Projects. However, we did not do a complete end-to-end workflow with our system. Let’s think about the following scenario:

For More Information:
www.packtpub.com/web-services-testing-with-soapui/book
A new guest has arrived to the hotel. An operator of the hotel reservation system registers the new guest in system, looks for a room, and reserves it for the guest.

This is the preliminary use case of our sample system. All three web services which we discussed previously, GuestManagementService, RoomManagementService, and ReservationService take part in this scenario. How are we going to test this particular scenario using soapUI? Will it be enough to create a project and add SOAP requests to execute the relevant operations of each web service individually, as we did in Chapter 3, First Steps with soapUI and Projects?

The answer will be yes, if we need to execute this particular scenario once and for all. However, we do not live in a world where software is built in a single run, tested once and used in production. Instead, there are a large number of iterations per release cycle as well as multiple versions. In such cases, if we do not maintain our tests in a reusable and structured manner, we will end up in a chaos.

We are going to use some of the important constructs of soapUI to build a comprehensive test suite to verify the preceding scenario. Open soapUI and start our journey of exploring more exciting features!

Creating a TestSuite

A soapUI functional test consists of three key elements as follows:

- **TestStep**: A TestStep is the foundation of any functional test. It is used to manage the execution flow of the test and validate the test results. A TestStep is directly associated with a TestCase.

- **TestCase**: In a soapUI project, a TestCase is a collection of TestSteps organized for testing some functionality of the service under test.

- **TestSuite**: A TestSuite is a collection of TestCases which work together as a logical unit to test some specific functionality.

The structure of these elements in a soapUI project can be represented in a diagram as follows:
We will discuss each of these elements in detail while we go through our sample TestSuite. Without spending more time on theoretical aspects, let's start to build our soapUI project.

Our first scenario, as we just described, is a relatively simple one. Open the HotelReservationProject in soapUI which we have created in the previous chapter. We have already added the WSDL of GuestManagementService there.

In this exercise, we will remove the SOAP 1.2 binding from each web service as we do not have to bother about SOAP versions at this moment. Therefore, you may remove GuestManagementServiceSoap12Binding (interface) from the project.

For More Information:
www.packtpub.com/web-services-testing-with-soapui/book
In addition to the WSDL of GuestManagementService, we will need to add the rest of the WSDLs of our sample hotel reservation system by performing the following steps:

1. Right-click on the HotelReservationProject and select Add WSDL. You will be prompted to specify the URL or Browse in the file system for a WSDL. This is shown in the following screenshot:

   ![Add WSDL](image)

2. We are going to add WSDL URL of the RoomManagementService. The WSDL which has been automatically generated by Apache Axis2, can be accessed by navigating your web browser to `http://localhost:8080/axis2/services/RoomManagementService?wsdl`. Once you have made sure that the WSDL is accessible in the browser, specify the WSDL URL of RoomManagementService (http://localhost:8080/axis2/services/RoomManagementService?wsdl) as the WSDL location.

3. Uncheck the Create TestSuite check box so that we can manually add a TestSuite as we preferred.

   If we select the Create TestSuite option, soapUI automatically generates a test suite for the imported WSDL. soapUI scans through the WSDL and extracts all the operations. Then it generates test cases for each of the operations. By default, one TestCase for each operation is created. So, if you have five operations in the WSDL, you will automatically get five TestCases.

4. Leave the Create Requests check box checked and click on OK.

5. Now, you will notice that the RoomManagementService interface is also added to our project. Make sure to remove the RoomManagementServiceSoap12Binding from the project. Similarly, repeat the preceding steps to add ReservationService as well.

For More Information:
www.packtpub.com/web-services-testing-with-soapui/book
6. As of now, our project structure will be similar to the following screenshot:

Under each service interface, you can see the operations corresponding to the operations defined in the respective WSDL. soapUI represents the MEP used by a particular operation using two distinct icons as shown in the following screenshot:

As **addGuest** operation uses request-response MEP, it is denoted by two circular arrow heads. The **deleteGuest** operation is denoted by a single arrow head since it is a one-way operation.

Now, think about our first test scenario again and decompose it into a few steps for clarity.

- Add a few hotel rooms to the system
- Upon the arrival of a new guest, a receptionist at the hotel registers the new guest in the system
- Finally, the receptionist reserves an available room for the guest
This particular scenario involves a few web service calls. New rooms can be added to the system by invoking RoomManagementService. Guest registration can be done through GuestManagementService. Finally the room reservation will be done by calling ReservationService. Though these three web services are logically related to each other, we have implemented our system in a way that they can be invoked independently.

Adding TestCases

We are ready to test our scenario. First, we are going to check whether the RoomManagementService works as expected. We can add a single TestCase to test each of the operations in RoomManagementService or we can add separate TestCases for operations. The choice depends on the functionality of service or operations. If the service has a large number of operations it will be a good idea to add a separate TestCase for each operation. In our case, we have a maximum number of three operations per service. Therefore, we may add one TestCase for a service. For the sake of completeness of the topic, we are going to follow the first approach where we add a separate test case for each operation; this gives us three test cases. Perform the following steps to generate TestCases:

1. Right-click on RoomManagementServiceSoap11Binding in the left-hand side menu of the project and select Generate TestSuite. Then the Generate TestSuite window will be seen as shown in the following screenshot:

   ![Generate TestSuite Window](image)

   - TestSuite: <create>
   - Style: One TestCase for each Operation
   - Request Content: Use existing Requests in Interface
   - Operations: AddRoom, deleteRoom, getRoomDetails

2. Accept the default options and click on OK, so that we will have one test case for each operation.

For More Information:
www.packtpub.com/web-services-testing-with-soapui/book
3. Specify `RoomManagementServiceTestSuite` as the name of the TestSuite.

4. Now, we will have a TestSuite with three TestCases as follows:
   - `addRoom TestCase`
   - `deleteRoom TestCase`
   - `getRoomDetails TestCase`

5. Each of these TestCases consist of one TestStep. We will look into each of the TestCases in the following sections.

### addRoom TestCase

`addRoom` is our preliminary TestCase which is used to add a new room record into the system. Initially, we do not include any additional TestSteps into this TestCase. We accept the default test steps generated by soapUI and proceed.

The `addRoom` TestCase editor can be opened by double-clicking on `addRoom TestCase` in the left-hand side menu or right-clicking on the TestCase and selecting `Show TestCase Editor`. If you open the TestCase Editor in either of these ways, it will be similar to the following screenshot:

For More Information:
You can see that our TestCase includes one TestStep named `addRoom`. If you double click on the `addRoom` TestStep, you will recognize it as a usual SOAP request.

**SOAP request** is one of the different set of TestSteps included in soapUI TestCases. soapUI provides you with various TestSteps which perform different types of test execution tasks. For example, HTTP Request TestStep can be used to send a raw HTTP message to a web page or a service. If you want to hold the execution of the current TestCase for a specified time, you can use Delay TestStep. JDBC Request TestStep can be used to invoke a SQL query in a database while running the TestCase.

As we are exploring soapUI in detail within the context of this book, we will use almost all of these TestSteps given in soapUI. For the moment, let's use the default SOAP request TestStep as follows:

1. Double-click on the `addRoom` TestStep which is under `addRoom TestCase` in the left-hand side menu of the project. This will open the SOAP request which can be forwarded to the `addRoom` operation in `RoomManagementService`.
2. Replace the ? entries with appropriate values. For example:
   
   ```xml
   <typ:roomNumber>101</typ:roomNumber>
   <typ:roomType>Standard</typ:roomType>
   <typ:roomSize>Double</typ:roomSize>
   ```

As we are ready with everything needed to test, we can run the `addRoom TestCase` now. However, we will wait until we have completed the rest of the operations in `RoomManagementService`.

If you look at the structure of the `RoomManagementServiceTestSuite`, the test cases are organized by soapUI under a specific order. `addRoom TestCase` is at the top and `getRoomDetails TestCase` is at the bottom. When running the TestSuite, soapUI executes the TestCases sequentially as they appear in the tree view of the TestSuite. Therefore, when we run `RoomManagementServiceTestSuite`, `addRoom TestCase` will be executed first. After that, the `deleteRoom TestCase` will be executed. Finally, the `getRoomDetails TestCase` will be executed.

Therefore, if were to execute this in the default order, after you add a room, it will be deleted instantly by the execution of `deleteRoom TestCase`. When the `getRoomDetails TestCase` executes at last, you always get a SOAP Fault since the particular room does not exist. Because of this, you should reorganize the order of execution of TestCases by moving the `getRoomDetails TestCase` to follow the `addRoom TestCase` in the TestSuite tree view of the soapUI project.

You can do this by clicking on the `getRoomDetails TestCase` and dragging it to be placed after `addRoom Test Case`.

For More Information:

**getRoomDetails TestCase**

Similar to the *addRoom TestCase*, *getRoomDetails TestCase* also consists of a single SOAP request TestStep. Perform the following steps to update the *getRoomDetails* TestStep:

1. Double click on the *getRoomDetails* TestStep. The SOAP request, which will be sent to the *getRoomDetails* operation, will be opened up.
2. Replace ? with the same room number we specified in the previous TestCase as follows:
   
   ```xml
   <typ:roomNumber>101</typ:roomNumber>
   ```

**deleteRoom TestCase**

We can execute the whole TestSuite once we complete the configuration of *deleteRoom TestCase*. *deleteRoom TestCase* includes a single SOAP request TestStep. The room which has been added after executing *addRoom TestCase* is supposed to be removed from the system by executing the *deleteRoom TestCase*. Therefore, follow the same steps as in the preceding TestCases and specify the same room number which has been previously added in the SOAP request:

```xml
<typ:roomNumber>101</typ:roomNumber>
```

**Running the first TestSuite**

In the preceding section, we have added three TestCases under the *RoomManagementService TestSuite*. SoapUI provides users with the facility to execute each TestCase individually as well as everything together.

In each TestCase, you will find the small green arrow icon which can be used to execute the TestCase alone as shown in the following screenshot:

---

For More Information:

Instead of executing each TestCase one by one, we are going to execute the whole TestSuite by performing the following steps:

1. Double-click on `RoomManagementServiceTestSuite` in the left-hand side menu of our soapUI project. This will open a detailed view of the TestSuite where you can see three TestCases which consists of the TestSuite.

2. We have completed updating all our TestSteps in the preceding section. Therefore, just click on the run icon (the small green arrow) which appears at the top-left corner of the TestSuite view.

3. Once the test execution is over, you will see something similar to the following screenshot:

4. All TestCases are marked in green denoting the success of the test. If you double-click on the green bars, the associated TestCase will be opened. Then click on the relevant TestStep. You can see the SOAP requests and responses which were submitted to the web service.

5. If you check the response of the SOAP request TestStep of `addRoom TestCase`, you will notice that the room has been added correctly to the system. Similarly, if you check the SOAP response of `getRoomDetails TestCase`, it will include the information of the added room as follows:

```xml
  <ns:roomNumber>101</ns:roomNumber>
  <ns:roomSize>Double</ns:roomSize>
  <ns:roomType>Standard</ns:roomType>
</ns:return>
```

For More Information:
With this, we can confirm that the room has been added to the system.

Now, is this the correct approach of verifying the success or failure of our test? Do we need to open the response messages of each and every request in TestSteps to find out what goes wrong or not? If this is the way we verify the status of tests, can this be considered as automated testing? If the preceding is all we can get from soapUI, what is the advantage of including SOAP requests under TestSuites and TestCases instead of directly sending them as we did in Chapter 3, First Steps with soapUI and Projects?

By now, we all have a lot of questions like these. We expect to find answers for all of these concerns before ending this chapter.

Let’s do another simple test. In the preceding RoomManagementService TestSuite, disable deleteRoom TestCase (right-click on the test case and select the Disable Test Case option).

Add another room (for example, room number 102) by executing the TestSuite. After executing the test, you will notice that the new room is added to the system. Now, execute the TestSuite again.

The test is successful again! You will notice that both addRoom TestCase and getRoomDetails TestCase are shown as passed with a green status bar. Click on the finished addRoom TestCase and double-click on addRoom TestStep. This will open the SOAP request and response as we saw earlier. This time, you will notice that we got a SOAP Fault as the response as we tried to add a room which has already been added. The SOAP Fault would be as follows:

```xml
<soapenv:Fault>
   <faultcode>soapenv:Server</faultcode>
   <faultstring>Room already Exists</faultstring>
</soapenv:Fault>
```

Though we got a SOAP Fault, why does soapUI show it as a passed test?

soapUI does not read our mind. We need to instruct it to fail tests if some conditions are not satisfied. In other words, we need to have a mechanism to validate the responses which we get as a result of TestStep execution. We can validate them by manually reading the responses as we did before. However, when executing complex test suites automatically, we cannot look and read each and every response manually to figure out the status of tests. Test assertions come into action in this situation.
Adding test assertions

Assertions allow users to validate the responses by comparing some properties of the message with expected values. In soapUI, assertions are applied to TestSteps. There are many predefined assertions available for us to use in soapUI tests. Some assertions are applicable only for a specific set of TestSteps whereas some are common for any TestStep.

You can add any number of assertions to a TestStep. After the TestStep is executed, all of the associated assertions are applied to the response. The TestStep is failed if any of the assertions fail.

Let's continue our discussion on assertions with our sample TestSuite.

We are going to add an assertion to addRoom TestStep in our project as follows:

1. Open addRoom TestStep by double-clicking on addRoom TestStep under the addRoom TestCase in soapUI project.
2. You will notice the add an assertion to this item icon at the top-left corner of TestStep editor. Click on that icon. The Select Assertion dialog box will open as shown in the following screenshot:

![Select Assertion](image)

3. You can find all assertions provided by soapUI in the preceding dialog box. During the course of this book, we will cover most of these assertions. In this example, let's use a few simple assertions.

Not SOAP Fault assertion

First, let's check whether we get a valid SOAP response instead of a fault. For that, we need to add a Not SOAP Fault assertion which evaluates the response to check whether it is a SOAP Fault or a valid SOAP response. If the response is a SOAP Fault, the TestStep will be marked as failed. To add a Not SOAP Fault assertion, perform the following steps:

1. Select Not SOAP Fault assertion from the Select Assertion drop down and click on OK.

For More Information:

2. Execute the RoomManagementService TestSuite again. This time the status of the TestSuite will be marked as failed in a red color. You will also notice that the reason for the TestSuite failure is the addRoom TestCase.

3. Select the addRoom TestCase and open the addRoom TestStep. The assertion result will be given at the bottom of the TestStep result view as shown in the following screenshot:

4. As you can see, we got a SOAP Fault as we tried to add an existing room to the system. We added an assertion, Not SOAP Fault, to check whether the response is a SOAP fault or not. In this case, the assertion evaluated the test to be failed as the response was a fault.

5. We know that if we execute this particular TestSuite again and again without any modification, we should get a SOAP Fault as we did earlier. Now, let's check whether we get the correct fault string in SOAP Fault. In order to check that, we can use multiple assertions. We will use XPath Match assertion first.

For More Information:
www.packtpub.com/web-services-testing-with-soapui/book
**XPath Match assertion**

An XPath Match assertion is used to compare the result of an XPath expression with a predefined value. We are going to check the SOAP response of the `addRoom` TestStep to evaluate whether it contains the expected fault string in case of a SOAP Fault.

1. In the same `addRoom` TestStep which we just used, select the **XPath Match** assertion from the **Select Assertion** dialog box.

2. Specify the XPath expression and expected result as follows:
   - **XPath Expression**: `//soapenv:Fault/faultstring`
   - **Expected Result**: `Room already Exists`

   The XPath Match Configuration window will look like the following screenshot:

3. When you specify the XPath expression as shown in the preceding screenshot, make sure to declare any namespace prefix which you use in the expression. In our case, we declare the namespace prefix of `soapenv` as follows:

   ```xml
   declare namespace soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   ```

   For More Information:
4. Note that all namespaces must be declared before they are used in the XPath expression.

5. If you are adding an XPath assertion based on a valid response message, the namespaces can automatically be declared by selecting the **Declare** button in the XPath expression editor. soapUI adds all namespace declarations of the current message to the XPath expression.

6. You can specify the expected result of the evaluation of the XPath expression in the **Expected Result** editor. According to the SOAP Fault message in our example, the expected output of the `//soapenv:Fault/faultstring` expression is a string value, Room already Exists.

   Similar to the namespace declaration, if you specify the expected result based on a valid response message, the result can automatically be retrieved by clicking on **Select from current** button in the **Expected Result** editor. soapUI evaluates the XPath expression which is specified in the XPath expression editor against the current message and returns the expected result.

7. After configuring the XPath expression and the expected result, click on **Save** to add the new assertion into the **addRoom** TestStep.

8. Execute the **RoomManagementService TestSuite** again. In the **addRoom** TestStep, you could observe two assertion results; a Not SOAP Fault assertion with a failed status and an XPath Match assertion with a pass state.

9. Here, the XPath Match assertion has been evaluated to **true**, as we got a SOAP Fault with the fault string, Room already Exists.

10. We have added two assertions for the **addRoom** TestStep. We have tested both of them for the failure case. If we execute this TestStep again with a new room number value, we will get a failure for XPath Match assertion as it checks the content of a SOAP Fault message. For now, just disable this XPath assertion by right-clicking on the assertion.

11. You can add another XPath assertion to check the success case of our test. For that, you can simply check the content of the SOAP response by an expression as follows:

   - **XPath Expression**: `declare namespace ns='http://sample.com/reservation/room/types';
   //ns:addRoomResponse/ns:return`

   - **Expected Result**: true

   **For More Information:**  
The Contains assertion

The Contains assertion is another simple and straightforward assertion which can be used to verify the existence of some text in response messages. Let's add a Contains assertion to the `getRoomDetails` TestStep in our example by performing the following steps:

1. In the `getRoomDetails` TestStep, select the Contains assertion from the Select Assertion dialog box.
2. The Contains Assertion dialog box will be shown where we can specify the content to be checked in response.
3. The response of `getRoomDetails` can always contain a string value, Standard, Luxury or Suite depending on the room type. Therefore, we can check the existence of those strings using a regular expression as follows:

   - Regular Expression: `(\?s).*?(Standard|Suite|Luxury).*`

   The Contains Assertion window would look like the following screenshot:

   ![Contains Assertion Window](image)

4. In the Contains assertion, the content which we look for can either be a string value or a regular expression. If we use a regular expression as in this example, we must check the Use token as Regular Expression check box, otherwise the expression we specify as the content will be considered as a pure string value.
5. Execute the `RoomManagementService TestSuite` again with a new room number. The `getRoomDetails` TestStep will be marked as passed.
6. We have done some preliminary modifications in our first TestSuite. However we are not done yet. We have not executed our whole test scenario yet. Before doing that, let's discuss another important construct in soapUI functional tests – properties.

For More Information:

Adding properties to soapUI tests

Properties can be considered as place-holders in a soapUI project. Properties are used to parameterize the execution of tests. In soapUI, properties can be defined at many levels in a project. You can define the properties which are common to your project at the project level. TestSuite and TestCase specific properties can be defined at their respective levels. Let's dive into the details of properties with our example project.

In our project, the project specific properties can be defined in the Custom Properties tab as shown in the following screenshot:

For example, we can define a property called Test at the project level as shown in the preceding screenshot. This property can be accessed from anywhere in our project through property expansions. For example, ${#Project#Test}.

A property can also be considered as a variable in a computer program. If we want to define something which can be used somewhere else, we can use properties. Therefore, in our sample HotelReservation project, we can make use of the properties at various levels.

We have organized our project into TestSuites, and each TestSuite deals with one web service. So, the web service specific properties can be defined at the TestSuite level by performing the following steps:

1. Select RoomManagementService TestSuite in the left-hand side menu in our soapUI project. You would observe the Test Properties tab at the bottom where we can define custom properties specific to the TestSuite.

2. In RoomManagementService TestSuite, we have three test cases and three test steps. In all these elements, we send the SOAP request message to a common web service endpoint. Also, the web service endpoint always consists of the following format: http://<host>:<port>/axis2/services/<serviceName>.

For More Information:
www.packtpub.com/web-services-testing-with-soapui/book
Working with Your First TestSuite

3. Without repeating this everywhere in our TestSuite, we can easily define some properties at the TestSuite level so that in case the service is moved into a different host or different environments, or the port or service name is changed, we do not need to change these in all the references in our project. We can simply change the value of properties.

4. Add three properties under RoomManagementService TestSuite as follows:
   
   - host = localhost
   - httpport = 8080
   - servicename = RoomManagementService

5. Now, we can access these properties through property expansions from anywhere in our TestSuite. For example, go to each of the three TestSteps and open the associated requests, addRoom, getRoomDetails, and deleteRoom. Go to the Test Properties tab of the request and edit the endpoint (or you can choose the [edit current..] option from the endpoint URL) as: http://${#TestSuite#host}:${#TestSuite#httpport}/axis2/services/${#TestSuite#servicename}.

Reading property values from a file

Usually, the properties are managed externally to the projects so that the property values can be updated without affecting the project settings. In soapUI, without defining the property value at the TestSuite, TestCase or TestStep level, we can read them from an external file. To read the properties from an external file, perform the following steps:

1. Go back to the Test Properties section under the RoomManagementService TestSuite.
2. Remove the existing values of all three properties.
3. Create a file called roommgtservice.properties in the filesystem. The content of the property file can be key-value pairs as follows:
   
   host=localhost
   
   httpport=8080
   
   servicename=RoomManagementService

4. Click on Loads property values from an external file icon which appears at the Test Properties tab as shown in the following screenshot:
5. The **Load Properties** dialog box will be opened. Browse for the `roommgtservice.properties` file (the one just created) in your filesystem and click on **OK**.

6. If the properties are loaded successfully, a message will be prompted as: **Added/Updated 3 properties from file**.

7. Replace the endpoint URL with the property expansions as we did earlier. Execute a TestStep (for example, `getRoomDetails`) and see the HTTP headers. You will notice that the endpoint URL is constructed by reading the properties from the file.

Note that the preceding property values are not dynamically loaded from the property file into the soapUI project. If you make an update in the property file, you should reload the file in order to reflect the change.

**Transferring property values between TestSteps**

Think about a scenario where you need to extract some value from a response message and include it in the subsequent request(s). In such a case, we need to have a mechanism to retrieve a specified value and transfer it to the other elements of the project. soapUI brings this functionality through the Property Transfer TestStep.

Let’s walk through our sample project to understand this important feature. We still have not associated TestSuites for `GuestManagementService` and `ReservationService`. In order to complete our end-to-end room reservation scenario, let’s add TestSuites for both these services by performing the following steps:

1. As we did in the *Creating a TestSuite* section, add two TestSuites, one each for `GuestManagementService` and `ReservationService`.

2. Now, you will have three TestSuites with each having three TestCases.
3. Add a new room by executing the `addRoom` Test Case of the `RoomManagementService Test Suite`. Specify the room details as follows:
   - room_number=201
   - room_type=Suite
   - room_size=Double

4. Add a new guest by executing the `addGuest` Test Case of the `GuestManagementService Test Suite`. Specify the following details for the new guest:
   - name=saman
   - address=Colombo
   - age = 32

5. Now, we need to reserve a room for the new guest by invoking `ReservationService`. We can do this by executing the `addReservation` Test Case of the `ReservationService Test Suite`. You can specify the following inputs in the `addReservation` SOAP request:
   - Room Number=201
   - Guest Name=Saman
   - Reserved From = 2012-01-25
   - Reserved To = 2012-01-27

What are we going to do here? We have implemented the reservation service in a way so that we would use it to explore some interesting features in soapUI. Therefore, let's assume the checkout procedure of our system is as follows:

The new guest Saman has requested to check out from the hotel. The reception of the hotel finds out which hotel room was occupied by the guest. Then they access our hotel reservation application and do a web service call to get the reservation details associated with the room number. Then, the hotel staff verify that the guest had occupied the specified room, the check-in and check-out dates, and so on.

Finally, the hotel staff remove the associated reservation record.

Think about this scenario with the web services we used in our system. The details associated with a reservation is obtained by calling the `getReservationDetails` operation of `ReservationService`. In order to remove the reservation from the system, we will call the `removeReservation` operation. The `removeReservation` operation can only be invoked with a `reservationID`. The `reservationID` is an auto generated identity primarily used as the key of a particular reservation. By invoking the `getReservationDetails` operation, we can get the `reservationID` and then use that particular ID to invoke the `removeReservation` operation.

For More Information:
www.packtpub.com/web-services-testing-with-soapui/book
As a tester of this system, you can manually call each of these operations and do what is necessary. But how should we use soapUI to correlate the requests and responses like these?

Our objective is to execute getReservationDetails Test Case and extract the reservationID from the response, then use that reservationID value in removeReservation Test Case. So, we are going to define a property which is common to both getReservationDetails Test Case and removeReservation Test Case. The ideal place to define this property is at the TestSuite level. To achieve this, perform the following steps:

1. Select ReservationService TestSuite from the soapUI project tree and click on the Test Properties tab.
2. Add a new property resID and keep the empty value.
3. Now, select the getReservationDetails Test Case. We are going to create a Property Transfer TestStep which will be used to transfer a specified value from a previous TestStep to the subsequent requests. Right-click on the TestStep element and select Add Step | Property Transfer.
4. You will be asked to specify a name for the step. Enter Reservation Property Transfer.
5. The Reservation Property Transfer window will be opened for us to add one or more transfers as shown in the following:
Working with Your First TestSuite

Here, in the property transfer window, the left-hand side pane lists down the transfers configured in this TestStep. Let's add a new transfer and discuss the rest of the features associated with it.

6. Click on the **Adds a new property transfer** icon at the top-left corner in the preceding property transfer window. You will be prompted to enter a name for the transfer. Enter ReservationIDTransfer.

7. Once the transfer is created, you can use **Source** and **Target** panes to specify the relevant XPath expressions to extract and replace property values. First, let's have a look at the Source pane. In the drop-down box next to Source, you can see various levels of soapUI projects which can be used as the source of property transfers. By default the closest TestStep will be shown. In our case, it is the **getReservationDetails** TestStep. The drop-down list next to Property shows the source property which is used in the transfer, which can either be request, response, or service endpoint.

8. We will select **Response** as we need to extract the reservationID from the response message. The Source pane would look like the following screenshot:

![Source Pane Screenshot]

9. Now, we need to specify the XPath expression to extract a value from the **getReservationDetails** SOAP response. The response will be similar to the following:

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns="http://sample.com/reservation/res/types">
    <ns:getReservationDetailsResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <ns:return xsi:type="ns:Reservation">
            <ns:guestName>saman</ns:guestName>
            <ns:reservationID>16</ns:reservationID>
            <ns:reserved_from>2012-01-25</ns:reserved_from>
            <ns:reserved_to>2012-01-27</ns:reserved_to>
            <ns:roomNumber>201</ns:roomNumber>
        </ns:return>
    </ns:getReservationDetailsResponse>
</soapenv:Body>
</soapenv:Envelope>
```

For More Information:
www.packtpub.com/web-services-testing-with-soapui/book
As we already have this response in soapUI (if not, run getReservationDetails TestStep once), we can declare namespaces associated with the XPath expression by clicking on the ns icon on the toolbar of the property transfer window. Once you are done with it, specify the XPath expression as //ns:getReservationDetailsResponse/ns:return/ns:reservationID.

10. Now, we need to specify the target where we want to transfer the value extracted from the above XPath expression. We can use the Target pane at the bottom of the property transfer window for that.

11. Similar to the Source, we can select one of the levels of the soapUI project to which the property value must be transferred to. In our example, we need to transfer the property value to the resID property which we have defined at the TestSuite level. Select ReservationServiceTestSuite as the Target and select resID from the drop-down list next to Property. With this, our property transfer configuration will be like the following screenshot:

For More Information:
www.packtpub.com/web-services-testing-with-soapui/book
In soapUI, the properties can be transferred to various targets based on the requirement. If we want to transfer the extracted value from a response to the subsequent requests, then we can specify the XPath expression of the request as the target. But in our example, we do not transfer the reservationID to another request in the same TestCase. Instead, we extract a value from a TestStep in getReservationDetails TestCase and pass it to the removeReservation TestStep which is inside removeReservation TestCase. In other words, we do not pass property values in between TestSteps in the same TestCase level. Because of this reason, we created a property which is common to both TestCases and assigned the property a value.

12. We have completed the property transfer settings. Now, we can parameterize the value of the <typ:reservationID> element in removeReservation SOAP request to read resID TestCase level property from the preceding property transfer. To do that, update the removeReservation SOAP request of the removeReservation TestCase as follows:

\[
<typ:reservationID>${\#TestSuite#resID}</typ:reservationID>
\]

13. Select the ReservationTestSuite and click on the run icon in the detailed TestSuite view (Make sure to disable addReservation TestCase since we have already added the same reservation previously). You will see that all TestCases will be executed. Have a look at the Test Properties tab at the TestSuite. You will notice that a value is assigned to the resID property, which is the autogenerated reservationID given in the getReservationDetails response.

We went through all the services in our sample hotel room reservation system and explored various functional testing features of soapUI. As an exercise, you could put all those together and create a comprehensive TestSuite with various assertions and try on your own.
Summary

Functional testing is a key for a success of any software development project. We can do functional testing manually or using automated tools. soapUI provides users with support for functional testing through various approaches. In this chapter we looked at building our first functional TestSuite to test the sample application which we developed in Chapter 2, The Sample Project. We discussed the basic constructs of any soapUI project, TestSuites, TestCases, and TestSteps. We made use of different types of assertions to verify the output of the tests. Finally, we looked into the soapUI properties and the transferring of property values between requests and responses.

The next chapter will be an extension of this and we are planning to discuss some advanced topics of web services testing using soapUI.
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.