

1. **Understanding Web services** (must read)

What are Web Services?

Web services are web components that transfers data between client and server. Client sends a web request to the server and the server then responds to client. This response will differ based on the web service request type.

Web services in SoapUI:

SoapUI is designed for validating web services easily.

Let's look at an example: A flight ticket booking application that runs in City 1 and is being accessed from City 2 to book a ticket. A user enters all the information such as boarding point, destination point, date of journey etc, and then as soon as the "Book Now" button is clicked, the web service from City 1 gets invoked and it passes all the information that is entered to the application server that processes the user request. The Reservation application will then send a response to the User's request.

Most of the online payment transactions are processed through web services only because of the enhanced security this method offers. An input parameter will be sent to the payment gateway website and which would be processed subsequently. An acknowledgement will be sent to the client regarding payment status finally.

All these activities can be seen through SoapUI request and response screens. SoapUI helps us to evaluate these web services.

Now let's see the important components of the web services. They are,

- **WSDL** – Web Service Description Language
- **SOAP** – Simple Object Access Protocol
- **UDDI** – Universal Description, Discovery and Integration
- **RDF** – Resource Description Framework

#1. WSDL (Web Services Description Language):

A WSDL is a document that should be written using XML. This document describes the following details about the web service:

- *Origin of the web service*
- *Header information*
- *Port type*
- *Input and output messages*

Each of the above information is represented as a tag in the WSDL file, such as:

1. **<types>** – XML Schema data types
2. **<message>** – the actual request and response data being communicated
3. **<portType>** – the target / end points where the actual web service is hosted to perform the operation
4. **<binding>** – the protocol information is given for the data format
5. **<definitions>** – the parent tag for the above mentioned tags

Now let's look at a sample WSDL file:

```

<definitions name="HelloService"
  targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <message name="SayHelloRequest"> 
    <part name="firstName" type="xsd:string"/>
  </message>
  <message name="SayHelloResponse">
    <part name="greeting" type="xsd:string"/>
  </message>

  <portType name="Hello_PortType"> 
    <operation name="sayHello">
      <input message="tns:SayHelloRequest"/>
      <output message="tns:SayHelloResponse"/>
    </operation>
  </portType>

  <binding name="Hello_Binding" type="tns:Hello_PortType"> 
    <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="sayHello">
      <soap:operation soapAction="sayHello"/>
      <input> 
        <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:examples:helloservice"
          use="encoded"/>
      </input>
      <output> 
        <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:examples:helloservice"
          use="encoded"/>
      </output>
    </operation>
  </binding>

```

© www.SoftwareTestingHelp.com

Your WSDL file should follow the W3C standard as above. Through web services we can convert into web based application. Web services are constructed on top of XML, HTTP, TCP

/ IP, Java, HTML and so on. Since web services are XML based language so that we can have these applications as local, distributed and web based environments.

Role of WSDL:

Validating web services using SoapUI is easy and is only possible with WSDL document because to configure web services in SoapUI, WSDL document is mandatory. If the WSDL document is not valid, SoapUI will throw an exception immediately. Now let us look at **UDDI** component.

#2. UDDI (Universal Description, Discovery and Integration):

This is a global repository where we can search the web services spread over the globe. In order to get or search web services just visit <http://uddi.xml.org/> web site. Here you can also register your own web service and make it available to global users.

UDDI is the place where the WSDL is described in detail. This will communicate through the SOAP protocol which will be explored later in this tutorial. Say for example, if you wish to advertise your products to the global customers you could create a web service and host it through UDDI. This can now be accessed by global users and from there the business could be established.

#3. SOAP (Simple Access Object Protocol):

Generally, it uses XML based data to interact with web applications.

Here are some points to remember:

- SOAP is language and platform independent as it is written by using XML.
- It creates the platform to communicate with the applications that are running in different operating systems using different technologies.
- Most of the Internet applications interact with each other over Remote Procedure Calls that use DCOM (Distributed Component) and CORBA (Common Broker Architecture)
- These technologies are different than the HTTP.

RPC (Remote procedure calls) are sometimes blocked by firewalls and proxy servers. To overcome these issues, SOAP was designed. There are some standard rules to be followed while building SOAP requests.

Let's take a look at sample SOAP document.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Header>
    ...
  </soap:Header>

  <soap:Body>
    ...
    <soap:Fault>
      ...
    </soap:Fault>
  </soap:Body>

</soap:Envelope>
```

© www.SoftwareTestingHelp.com

As you can see, a **SOAP document must contain the following elements:**

1. **Envelope element** is the top most tag which identifies the XML document as a SOAP message.
2. Followed by Envelope element, you see the **header element** that has header information.
3. The **Body element** specifies the call and response information.
4. Finally, you have a **Fault element** which contains errors and status information.

The above said elements should be declared with default namespace for the SOAP envelope.

Generally, a protocol is a set of standard rules that transfers the data between two regions in the Internet over the web services. There are many protocols that are used in the Internet applications. They are, Transmission Control Protocol (TCP) which serves as a packet between two connections. Internet Protocol (IP) that sends and receives the messages between two destinations.

Let us see some other important protocols:

- Hyper Text Transfer Protocol (HTTP)
- File Transfer Protocol (FTP)
- Border Gateway Protocol (BGP) and
- Dynamic Host Configuration Protocol (DHCP)

These protocols are used according to the requirements.

#4. RDF (Resource Description Framework):

RDF contains the description of the web resources such as title, author, content, and copyright information. This framework was designed so that computers can be read and understood easily by the web.

RDF is completely written by using XML language.

RDF data can be transferred between different types of computers using different operating systems and programming languages. Generally, RDF uses Uniform Resource Identifiers (URIs) on the web and it describes the resources along with the property and property values.

Take a look at the **sample RDF document** to understand better:

```
1  <? xml
   version="1.0"?>

2  <RDF>
```

3

```
<Descriptionabout="http://www.softwaretestinghelp.com/rdf">
```

4

```
<author> Wilfred R. Myers  
</author>
```

5

```
<homepage>http://www.  
softwaretestinghelp.com</homepage>
```

6

```
</Description>
```

7

```
</RDF>
```

What is XML?

XML (eXtensible Markup Language) is a mark-up language that is used for storing, sharing and formatting data. In general, an XML document is built by the tags. Let us see the sample XML content for a user's personal information.

<Firstname> Joel </Firstname>

<Lastname> King </Lastname>

<Address> 1432 Valley Drive </Address>

<City> New York </City>

<Country> United States </Country>

<Zipcode> 19714 </Zipcode>

Meaning of "eXtensible" and "Markup":

In the above sample, **First name, Last name, Address** etc. are enclosed by less than (<) and greater than (>) symbols. These labels are known as tags and the one with forward slash (/) along with the text, that is called closing tag. Tags are also called as mark-ups. These are customized as needed. This customization is not possible in other mark-up languages like SGML, HTML and so on. This is why XML is an **extensible** language.

XML focuses on the data for storing, sharing and exchanging as required, and HTML deals with the **format** of the data like applying colours, adding images, changing fonts, styles and so on.

XML and HTML can be used together in applications. For example, if you take a book, there will be textual data and graphical representation formatted. Hypothetically, XML can handle storing actual data and HTML applies the format for the content. Thereby the text book could have information as well as attractive images and colours.

How XML works with SoapUI?

As XML is a common language on Internet, it can be integrated with SoapUI because web services are mostly written in the form of XML. Also, if we pass XML input parameter to the web service, the response itself will be in the form of XML. SOAPUI can configure these web services

Conclusion:

So far in this tutorial, we took a look at:

- Web services and its several components like WSDL, UDDI, RDF SOAP
- Importance of WSDL document and its body of content
- XML and its usages in SoapUI

2. **Features of SoapUI & SoapUI Pro**

#1. User Friendly GUI

Even without prior familiarity, SoapUI is very comfortable for new users to work with. For example, if we wish to create a SoapUI project, just click on the File menu and then click New SOAP Project option and then provide valid WSDL file path. That's it. Similarly, if you take any assignment in SoapUI tool, we can do it as easily as Microsoft suites.

#2. Easy for Functional Testing

SoapUI provides drag and drop options for creating test suites, test steps and test requests to build complex test scenarios without writing any background scripts. Once a project is created, then we can add test suites under it. Test suite includes test steps and test requests based on the services.

A project can be used several times for smoke testing and functional testing. If we need any test suites for other projects, SoapUI offers the feature called cloning which enables us to duplicate existing test suites and put them into other projects.

*SoapUI also provides options for **test debugging** that lets us watch the test execution step by step. With the help of SoapUI, we can also perform data driven testing within a short period of time. All of these are going to be addressed in greater detail later.*

#3. Vulnerability Testing

SoapUI and SoapUI Pro tools provide options to protect the websites from hackers and viral software applications. Vulnerability testing is a type of testing that helps us to identify the weak areas of web applications.

With the SoapUI family tools, we can protect applications by executing Test Generator, SQL Injection and XML Bomb methods. Test Generator is a SoapUI Pro feature. It helps to create complete vulnerability test suites.

Similarly, SQL Injection feature allows us to provide some standard SQL queries and methods to identify the weak areas of the application and data base side.

For example, see the below SQL query:

Select *from Customers where CustomerId = "C2014" or **1=1**

The above query will return all the customers since the 1=1 condition is always true. This way hacker can get all the user name and passwords easily with this sample query. SoapUI tool can simulate these queries so we can understand the hack-proof-ness of the site.

XML bomb is in SoapUI that allows us to test services by passing huge XML data and examines the overflow of the application.

In addition to these, SoapUI tool has many more features like **cross-site scripting, passing random string data to identify the string vulnerabilities, boundary level testing, etc.**

#4. Load Testing using LoadUI

SoapUI can also estimate a web application's load balancing capacity. To do so, SoapUI includes an option called LoadUI that is available at the toolbar. After creating a project with proper test suites we can move to load testing by just clicking on the LoadUI option. SoapUI then, navigates to the LoadUI tool (it should have pre-installed on your computer for this to work) and then to the tests can be configured based on the need.

After executing the load test, LoadUI will generate a report that helps determine whether the application can run with a heavy load or not.

#5. Automation with Groovy

As discussed before, we can use SOAP and REST based services to validate in SOAPUI. SoapUI user interface is designed as a simple and comfortable interface for all the users.

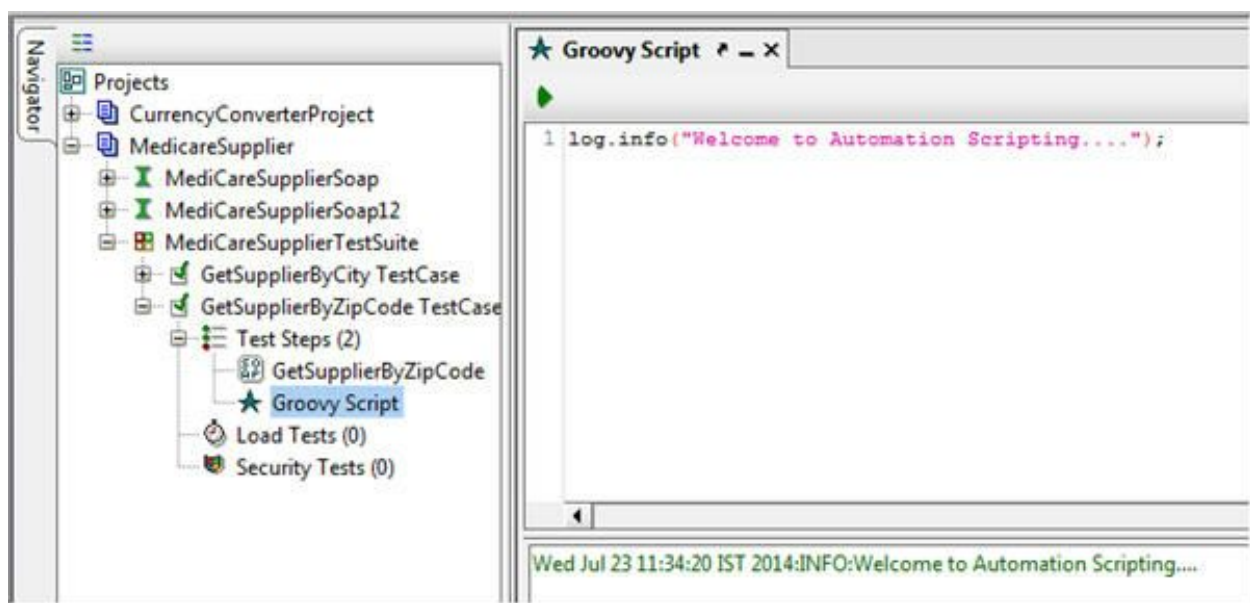
To write automation scripts in SoapUI, we need to add Groovy Test step under the test suite. Groovy script has built-in libraries and allows us to integrate java based libraries too.

So, it will be very helpful if you are familiar with Core Java. We can write complex scenarios using Groovy script and java.

For example, consider a situation where you need a response from one test request and then pass it as an input to another request. To accomplish this, we could store the response data in global properties and then reuse them through the scripts.

See the sample screenshot which shows **Groovy script test step and a sample script:**

(Click on image for enlarged view)



© www.SoftwareTestingHelp.com

#6. Data Driven Testing

SoapUI Pro supports data driven testing. It lets us perform bulk insert, delete and update related testing. We could upload Excel/CSV format test data to perform bulk testing.

In order to perform data driven testing in SoapUI, we will have to add *DataSource* and *DataSourceLoop* test steps under the test suite. *DataSource* test step deals with the

external data source configuration and DataSourceLoop fetches the data row by row from the external data source. More information on this is coming up in the future articles.

#7. Assertions

Assertions are another key feature in SoapUI. It basically validates the response message while executing the test steps by comparing it to any part of the response message or entire message.

For example, if we have a authentication web service which should authenticate the login credentials given by the user. Let's assume the web service response is in JSON format. So if the authentication is done successfully means, service will return successful message to the user.

Here's a sample response:

Successful Authentication:

Response [

{

"Message": "Successfully Authenticated",

"Status": "true"

}]

Failure Authentication:

Response [

{

"Message": "Authenticate Failed",

"Status": "false"

}]

*In the above responses, we have "**Message**" and "**Status**" elements. So, it is easy to validate these responses using either "**Message**" or "**Status**" value. For that, we need to configure in the respective assertions appropriately as XPath Match assertion, XQuery, Contains and Not Contains etc.*

SoapUI NG Pro:

SoapUI recently released the latest version of SoapUI Pro. It is basically developed on core SoapUI so you can continue using SoapUI existing projects with this version as well.

You can compare features of SoapUI and SoapUI NG Pro on this page: [Feature comparison of SoapUI and SoapUI NG Pro](#).

SoapUI NG Pro Important Features:

- 1. SoapUI NG Pro gives the complete functional testing capability for SOAP API, REST and other protocols*
- 2. SoapUI NG Pro is introduced in "Ready! API platform" which determines the actual functionality of the API service and its expected behavior.*
- 3. Ready! API platform provides the skeleton that determines our service inputs to generate the test coverage report which evaluates the functionality coverage implicitly.*
- 4. It allows ad-hoc testing or command line interface to test our APIs effectively.*
- 5. All the REST, SOAP API and other service components can be used by simply drag and drop method*

6. In SoapUI NG Pro, data driven feature is little enhanced in retrieving information from external data sources for example, excel, XML, JDBC data sources and file / directories etc. Then these retrieved data will be converted into SoapUI NG Properties test step.

7. We can transfer the property test step values to xpath-queries, scripts and so forth.

8. SoapUI NG Pro offers the feature called **point-and-click** to generate test scenarios quickly

9. SoapUI NG Pro allows the end user to customize their services easily even they are new to SoapUI Pro or development experience.

10. Few more important features available in SoapUI NG Pro:

- *Test Coverage: To analyze the API tests along with the functionality as expected*
- *Multi-environment Support: Allows to change the testing environment based on our requirements*
- *Test Debugging: This feature helps to analyze the test step-by-step debugging. It also includes variables, properties, input requests etc.*
- *Complex Scenarios: SoapUI NG Pro makes it easier the APIs which are involved in client server architecture*
- *Drag and Drop Test Creation: As it exists, it is easy to create and run the test scenarios by drag and drop feature*
- *SoapUI team also introduced LoadUI NG tool for LoadUI Pro users. It is used for performing load testing on Ready! API platform. It basically simulates the SoapUI NG Pro test cases and determines the load of the application server*

3. **Installation of SoapUI and SoapUI Pro**

Installing SoapUI:

Download SoapUI free version:

Go to <http://sourceforge.net/projects/soapui/files/> web site and there you can see the several download links related to the SoapUI. Along with SoapUI, certain plugins can be used as needed. They are,

- *soapui-netbeans-plugin*
- *soapui-intellij-plugin*
- *soapui-eclipse-plugin*
- *soapui-maven-plugin*

Let us see of the plugins briefly:

NetBeans is a Java editor where we can write, debug and execute java code. Along with the NetBeans, SoapUI NetBeans plugin is integrated for testing web service functionality from NetBeans itself. So, it behaves like both development and testing environments.

Soapui IntelliJ Plugin allows testing web service functionality within IntelliJ IDEA. SoapUI also provides the command line runners to execute the test suites and test cases.

With current SoapUI, we have **SoapUI Workspace Navigator** on left side of the window that helps organize projects, test suites, etc. Another useful window is **Log tabs** and is located at the bottom of the SoapUI screen. We will see these in details in our upcoming tutorials.

SoapUI Eclipse Plugin is slightly similar to SoapUI NetBeans plugin but it provides drag and drop components whereas SoapUI Eclipse does not. As it is integrated with the SoapUI libraries, we can test Java web services from SoapUI Eclipse plugin itself.

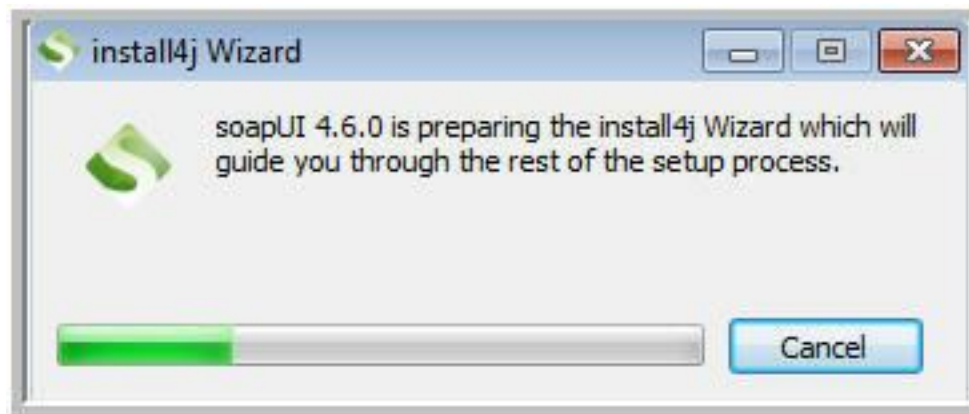
SoapUI Maven Plugin specially designed for build management process. During the deployment, scheduled builds can be started at the specified time. The build processes are written through maven script. Likewise, we can execute the scheduled test suites or test cases as mock services through this plugin as it is integrated with SoapUI.

How to Install SoapUI on Windows System?

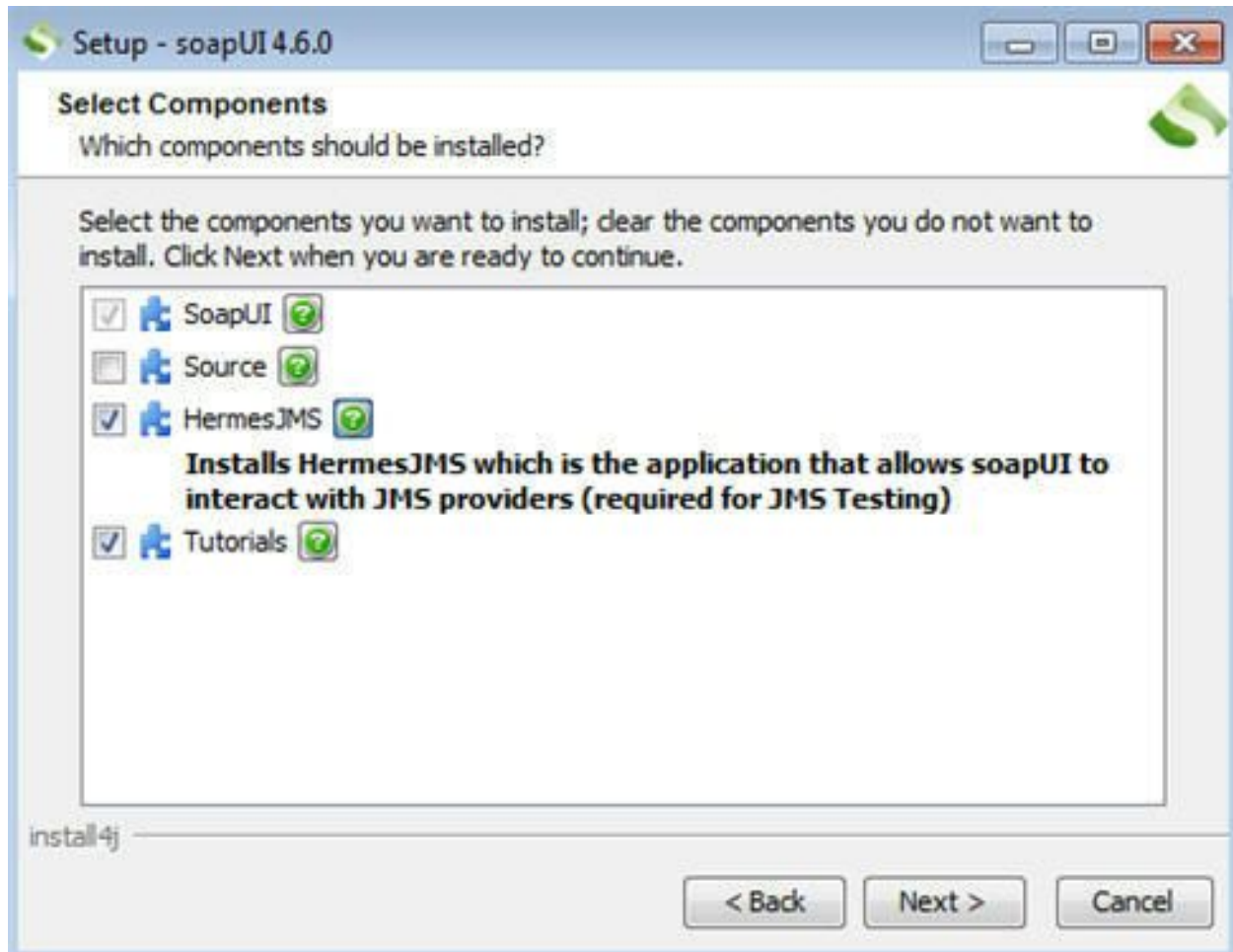
So far we discussed various SoapUI plugins and their purposes. Now let us go to install SoapUI on Windows machine.

We need to download SoapUI and plugins from <http://sourceforge.net/projects/soapui/files/> web URL. You can also download all the files from SoapUI website. Once all the installable files are downloaded, double click on the SoapUI executable file.

Installer will start the process as shown in the below screen:



In the welcome wizard click on Next button to move to license wizard. Accept the terms and conditions as described in the text area after reading it. Then, click Next. Specify the destination folder where SoapUI can extract the supporting files and install. Click Next to choose additional components. See below screenshot for your reference.

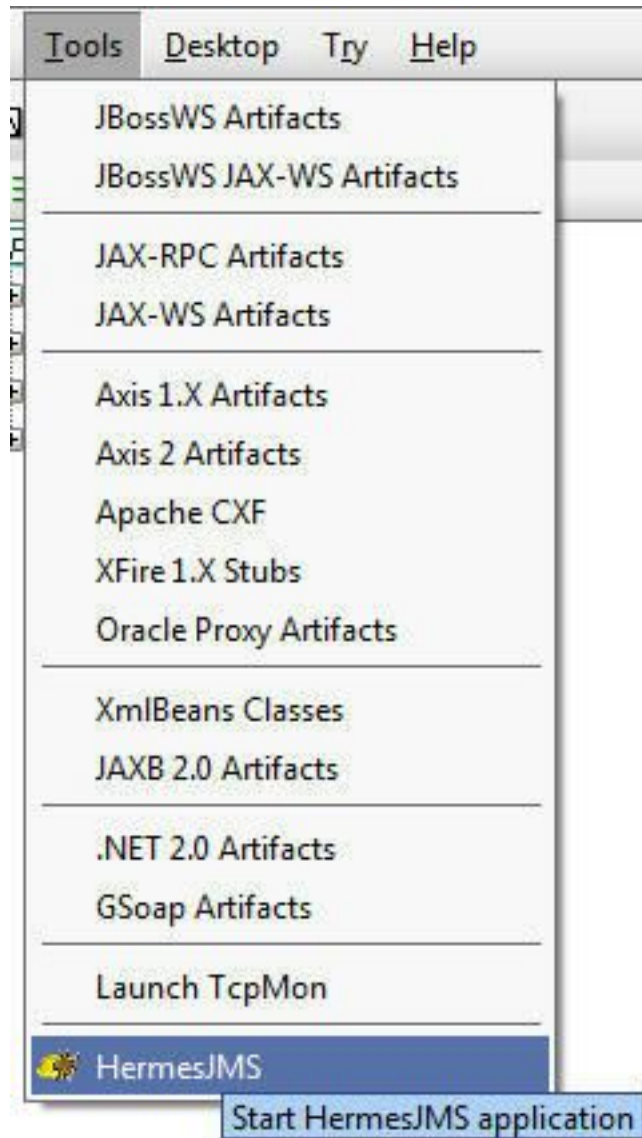


In the screenshot, we can see some of the components apart from SoapUI.

Source component contains the complete source code for the SoapUI tool. You may install this component and analyze the SoapUI source code.

Tutorials component provides the SoapUI tutorial related files.

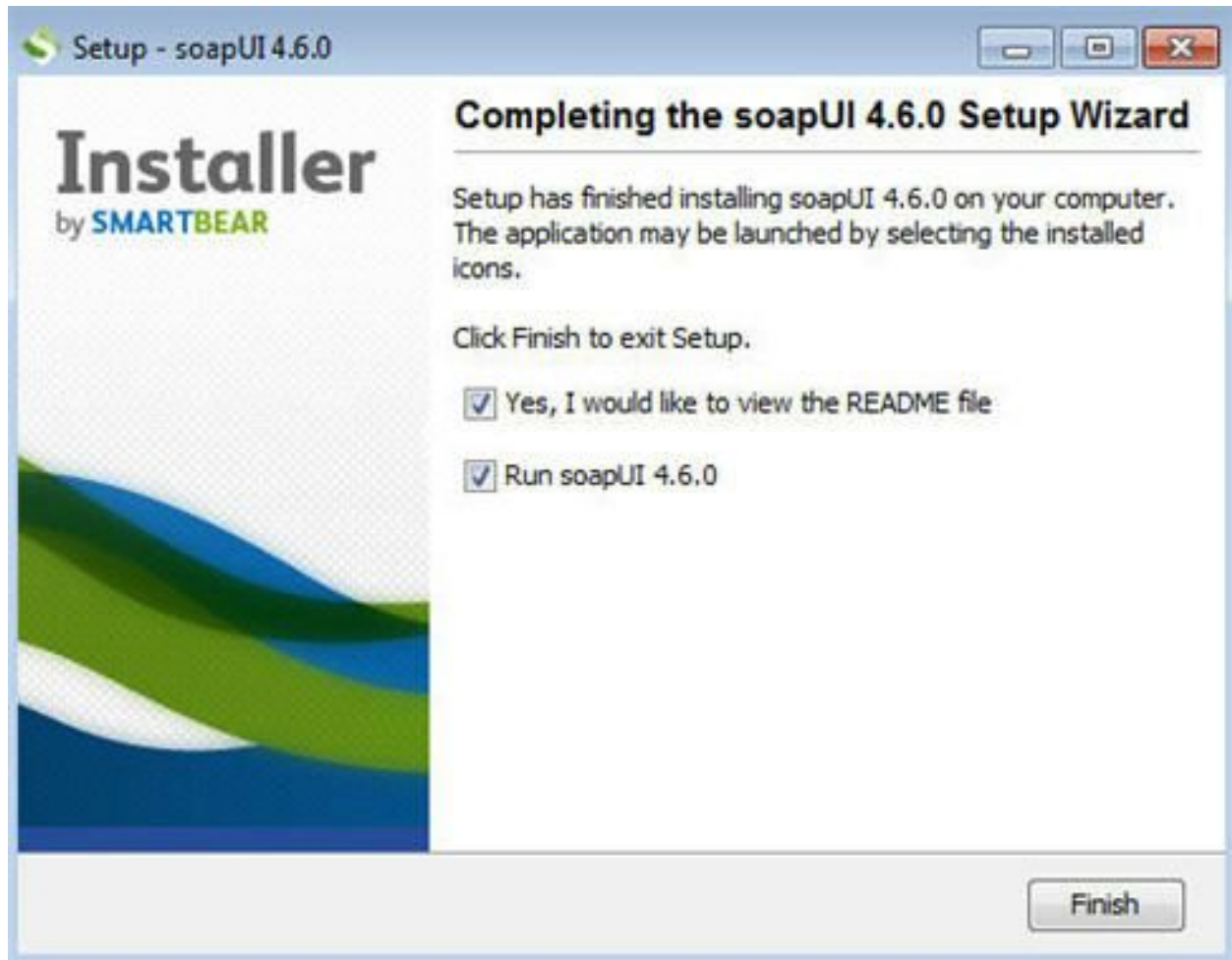
HermesJMS component is used for sending and receiving messages through Java Messaging Service. This could be configured in case of testing JMS related services. We can configure *HermesJMS* from the Tools menu as below:



To install HermesJMS component, once again we need to accept the license agreement. So click on the Next button.

The following wizard will prompt us to specify the shortcut to be present under which program in the start menu. Later, we have to check desktop icon if needed. That's it!

On Next button click, installation begins. Once complete, it will show this window:



Installing SoapUI Pro:

=> Download SoapUI Pro from this page.

Fill the form and click Download Trial.

The trial license key will be sent to the given email address. It will be valid for two weeks. Once license is expired additional pro features will be disabled but basic features can be used forever.

Below is the page that we will be redirected to after completing the registration.

Note that below screen may change based on the latest changes from SmartBear Software.
You can follow the simple download instructions and select the appropriate install version
based on your Windows, Linux or Mac version.

Trial Application Received

User Rating: ★★★★★ / 317

Success!










We have received your application for a SoapUI Professional Trial License. The Trial License will be valid for 30 days. Meanwhile, you should make sure that you've got the latest version of SoapUI Professional.

If you have signed up for a trial previously with this email, please contact us at sales@soapui.org.

Latest Release

SoapUI Pro 5.0.0

(Released on 9 April 2014)

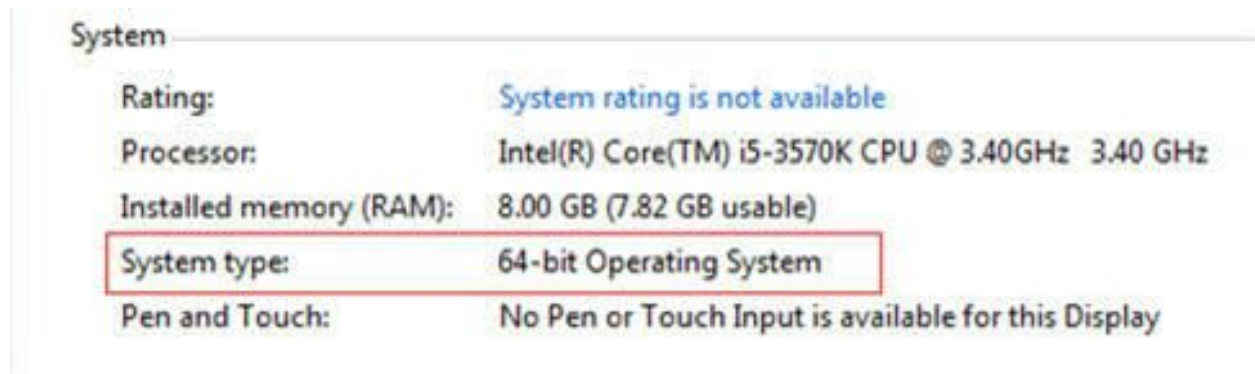
 Windows installer (32-bit)	» Download
 Windows installer (64-bit)	» Download
 Mac OS X installer (64-bit)	» Download
 Linux installer (32-bit)	» Download
 Linux installer (64-bit)	» Download
 Windows zip (32-bit)	» Download
 Windows zip (Java not included)	» Download
 Mac OS X zip (Java not included)	» Download
 Linux tarball (Java not included)	» Download

Choose the download link depending on your system specifications. For this tutorial we are going to install SoapUI Pro on windows machine by clicking Windows installer (64-bit) download link

To know your computer type, follow these steps:

- Right click on **My Computer** icon present on your desktop
- From the context menu, click **Properties**
- In the right side panel of the properties screen, have a look at **System Type** under the **System** section.

Refer the following screenshot for better understanding:

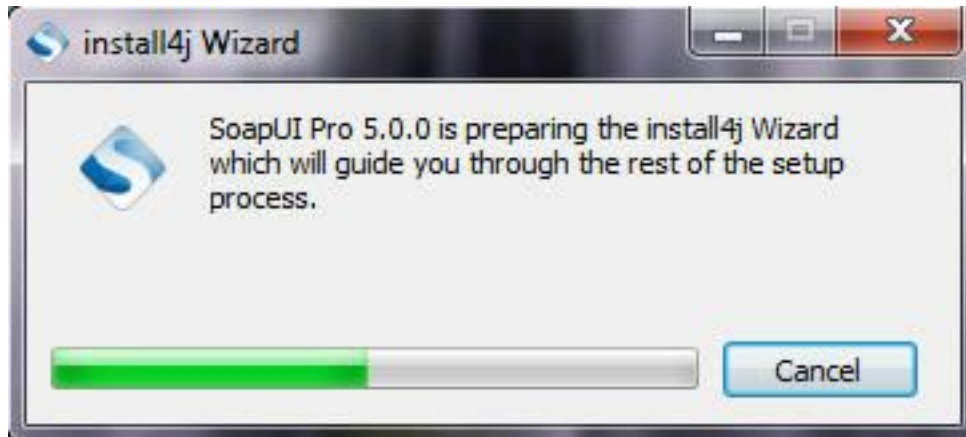


Browser starts downloading and may take few minutes to complete. Downloading time may vary depending on your Internet speed. You can see the download progress on your browser's download section.

When SoapUI Pro download is completed, we can see the executable file in our default download file location.

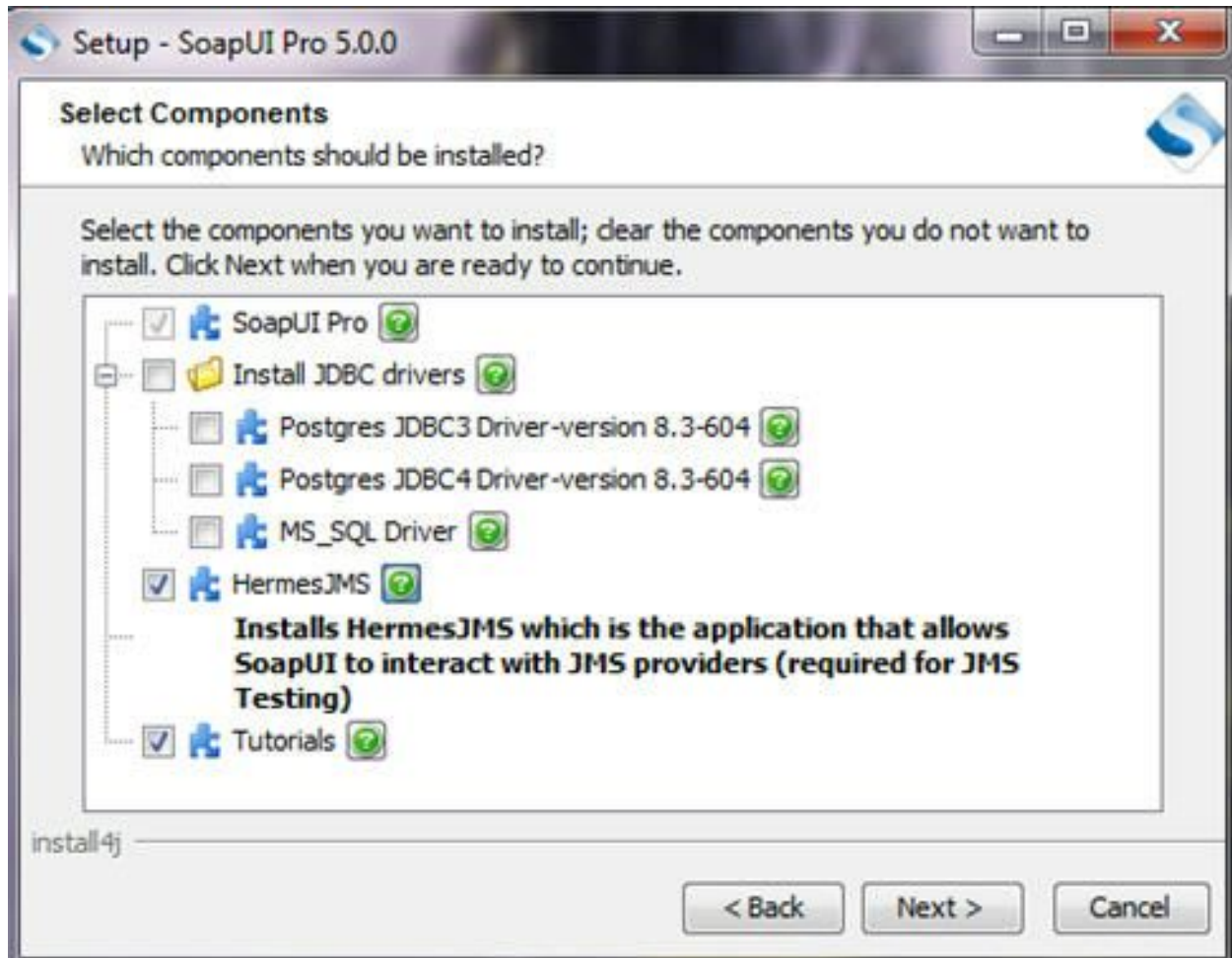
To install SoapUI Pro, follow these steps:

Double click on the setup EXE file. Windows installer initializes the installation process



Accept the license agreement by clicking "I accept the agreement" radio button and then click Next button.

Specify destination directory. By default, system will install in C:\ drive. If we want we can change the destination folder. Select the target folder and then click Next button. This wizard provides several components to be selected as required.



JDBC Drivers: If we are handling any database related testing like executing SQL queries and passing data to database, this component will be useful.

We have already seen about other components while installing SoapUI itself.

So you can decide your required components and click Next button.

After that, installation wizard will prompt you to install LoadUI. If we check this checkbox, first it will download the LoadUI from the Internet and then install it. We are not going to install LoadUI now. So click Next.

Note: If you are installing the SoapUI NG Pro from "Ready! API" platform, everything including SoapUI NG Pro, LoadUI NG Pro, Secure Pro and ServiceV Pro will come as a bundle. You can download the and install the trial version of Ready! API from this page.

This license agreement wizard is shown for HermesJMS as we have seen in the previous section. So accept the license agreement and click next.

Now we have to specify the tutorial location because I have checked **Tutorials** component in the **Select Components** wizard. Then move to the shortcut creation wizard in the start menu. Click on Next button after the short cut name is given. Click Next button again.

SoapUI Pro installation will start and it takes few seconds to complete. At the end click Finish button to launch SoapUI Pro.

Activating Trial License for SoapUI Pro:

Go to your email inbox. Find SoapUI Pro activation email from SmartBear Software and click on activation link. Once you activate your email address you can download the trial License key as shown in the below screenshot:

Email verified

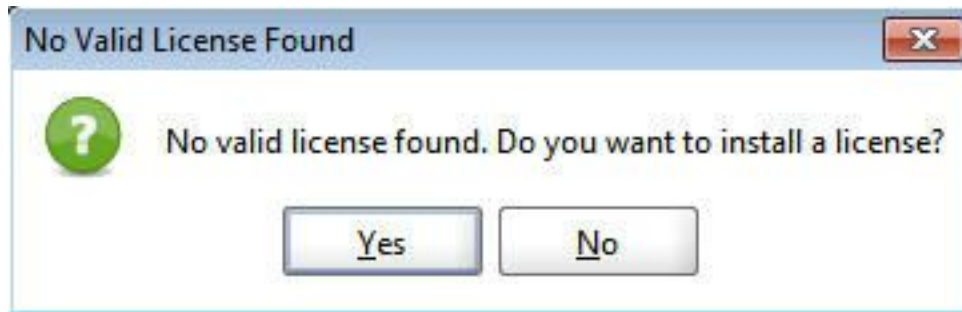
The email address **chandrakumar@gmail.com** has been verified. You can now start your trial inside Ready! API.

Please go to Ready! API and activate your trial.

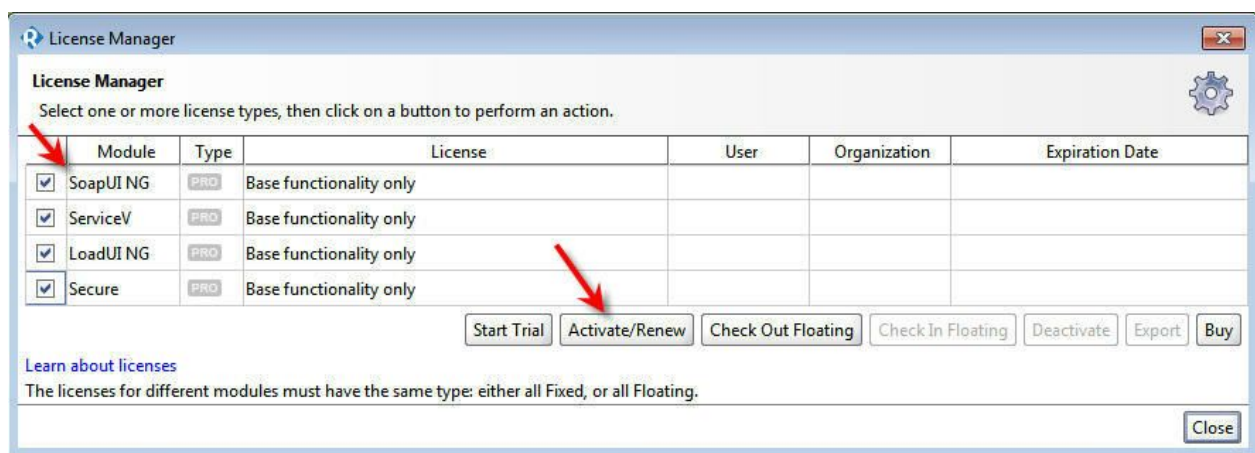
If you can't activate trial in Ready! API or if you are using SoapUI Pro, you can download the license file [here](#) and install it.

-

Download and extract the trial license key zip file. Now go and start the SoapUI Pro program from your all installed programs. It will ask to activate your installation.

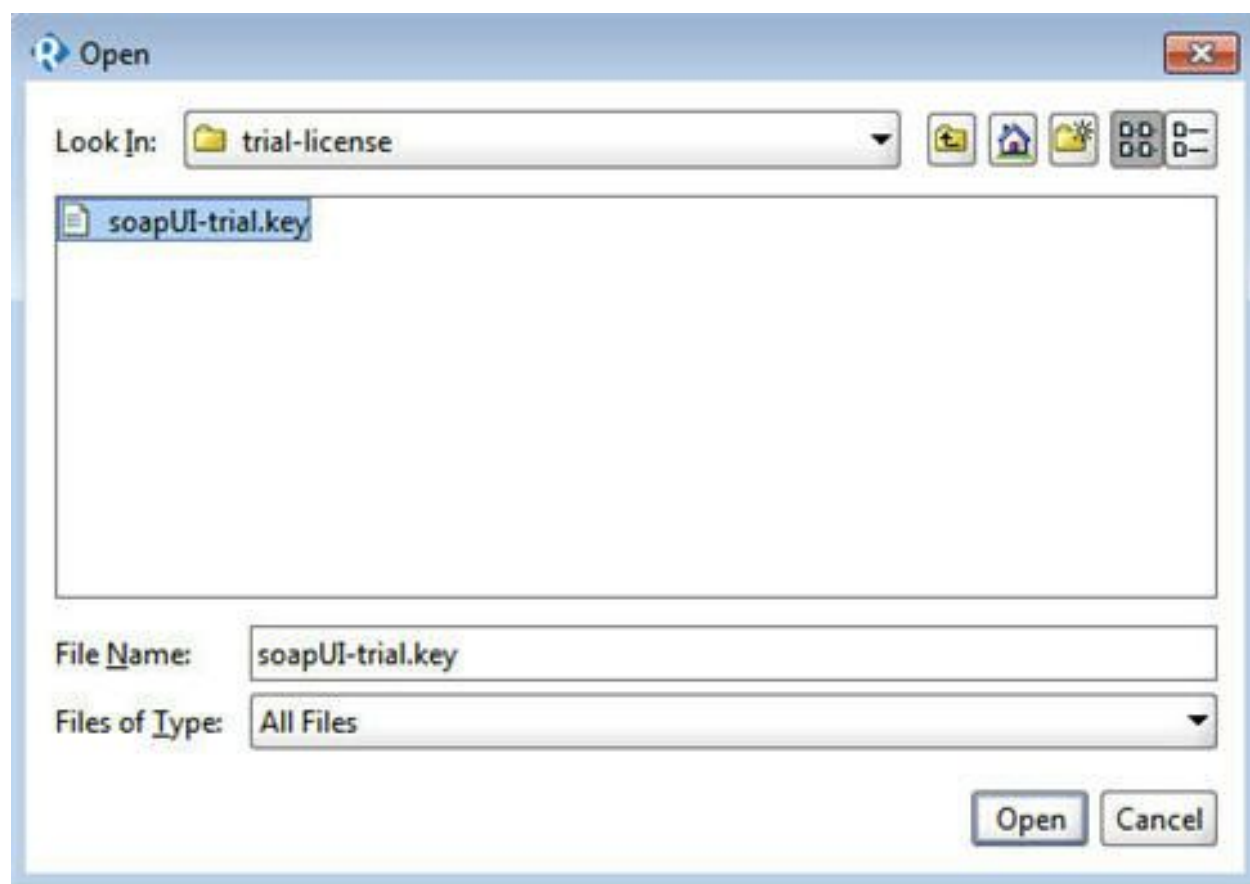


Click yes and provide the path of downloaded license key as shown in below screenshots:



Note: Only select SoapUI pro in this window as we are not installing other versions.





4. **Working with Projects** (must read)

Working with SoapUI Projects:

Creating a new project by adding the WSDL:

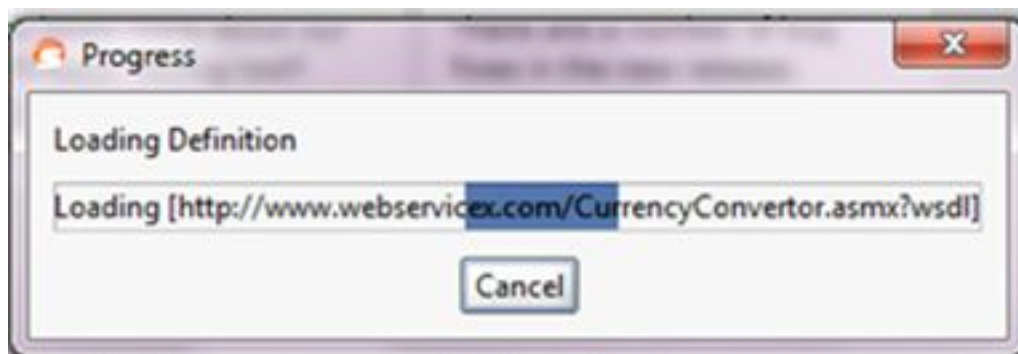
#1. Open SoapUI application and follow the instructions to proceed with the license process

#2. Click **New SOAP Project** option from **File** menu or press **CTRL+N** shortcut key.

#3. Enter the project name (meaningful one is better)

#4. Then specify the valid WSDL URL in the given text box. Let's use currency converter URL. i.e. <http://www.webserviceex.com/CurrencyConvertor.asmx?wsdl>. (There are many other sample WSDL urls available. Please check for the open source Web services available for variety)

#5. The remaining setting can be left default and then click OK. The below WSDL processing progress shows up (**Note:** internet connection is mandatory for this to work)



#6. Once WSDL URL processing has been successful, SOAP project will be created along with the service requests.

The URL we used in this tutorial can be called from anywhere through Internet. This web service is hosted on a web server and on calling the URL the hosted server is searched and SoapUI project gets loaded with the services contained within it as you can see below:



Project creation done!

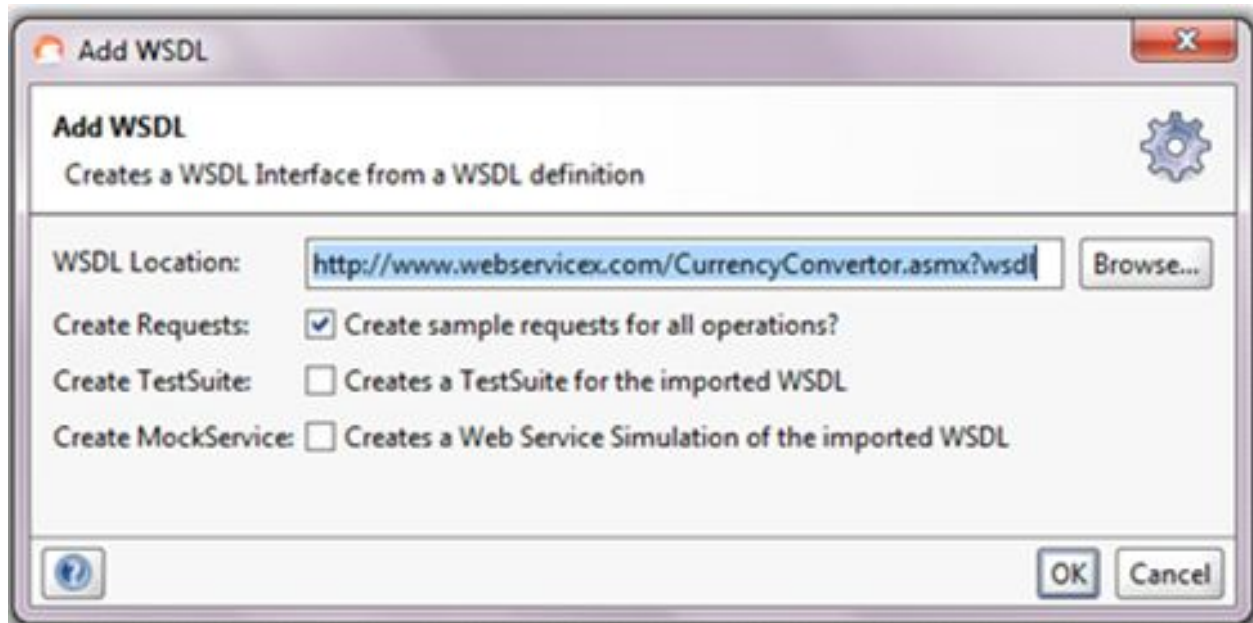
Adding a WSDL to an existing project:

#1. Right click on the **Project Name** in the Navigator panel

#2. Click **Add WSDL** option or hit CTRL + U

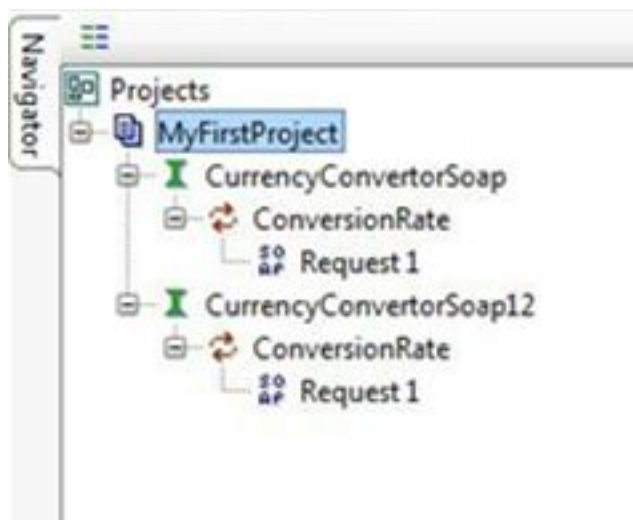
#3. Add WSDL dialog appears on the screen.

#4. Enter valid WSDL URL in the text field as seen below:



#5. Click OK

#6. The URL is processed and the respective services get loaded into the SOAP project as below:

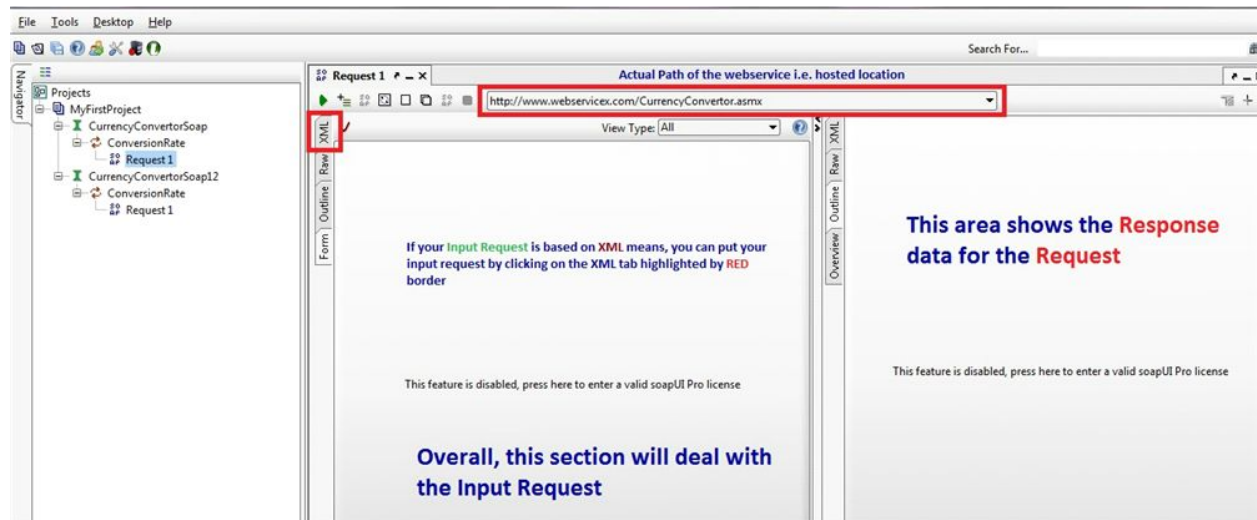


Executing Services & Response Verification:

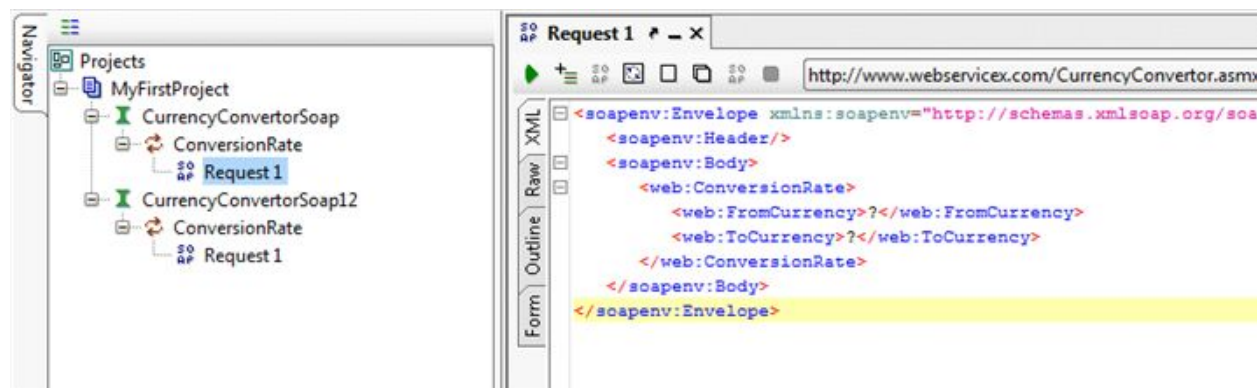
#1. Expand the **CurrencyConvertorSoap** in the tree (click on the +)

#2. Double Click **Request1** (the service name, this can be changed if needed)

Please take a look at the screenshot for more information: (Click on image for an enlarged view)



#3. Click on the XML tab from request section. It will show the input request for currency convertor web service as shown here in the screenshot. (Click on image for an enlarged view)



In the above screen question mark (?) symbols are in the input request. These are the input parameters for the currency convertor web service.

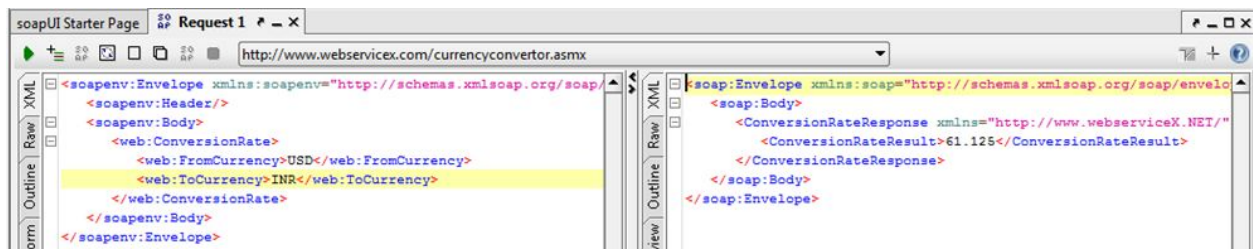


When run / start icon is clicked, SoapUI will call the currency convertor web service along with the input parameters that were provided in the request. Then, the web server will receive these input parameters and process them. Once done, the server will send the response back to SoapUI.

Sometimes the response may contain error messages. For example, while processing the input request, server may be down or Internet connection could not be established from our side. During that time, we will get a response that is an exception.

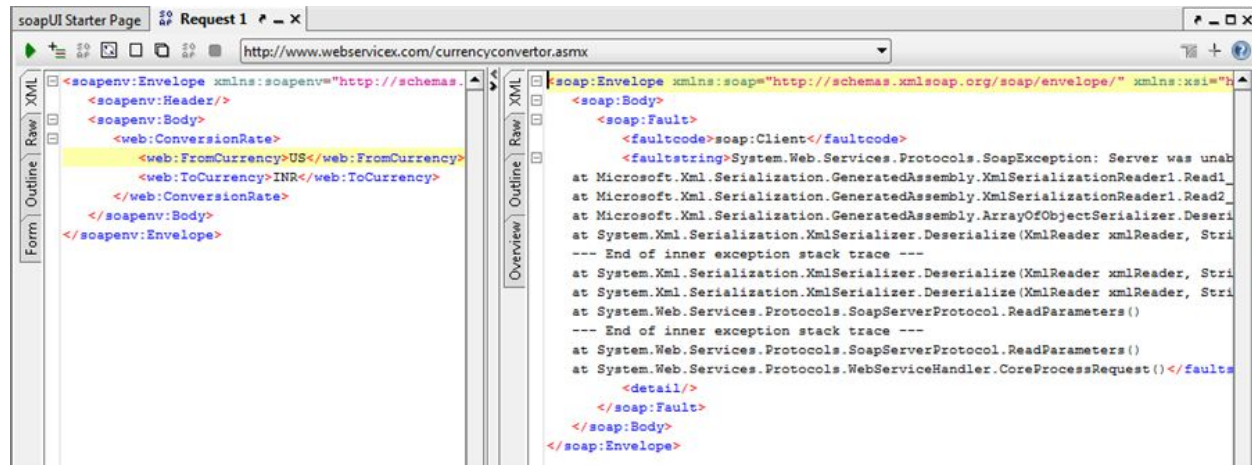
For instance, let us enter USD for <<From Currency>> and INR for <<To Currency>> with valid values as below and call the service. As can be seen below, the correct response is obtained.

(Click on image for an enlarged view)



To test a negative scenario, let me change the <<From Currency>> as **US** and execute the service.

(Click on image for an enlarged view)



To this we received an unknown error messages because our input was wrong. The same error messages will be shown in the **error logtab**.

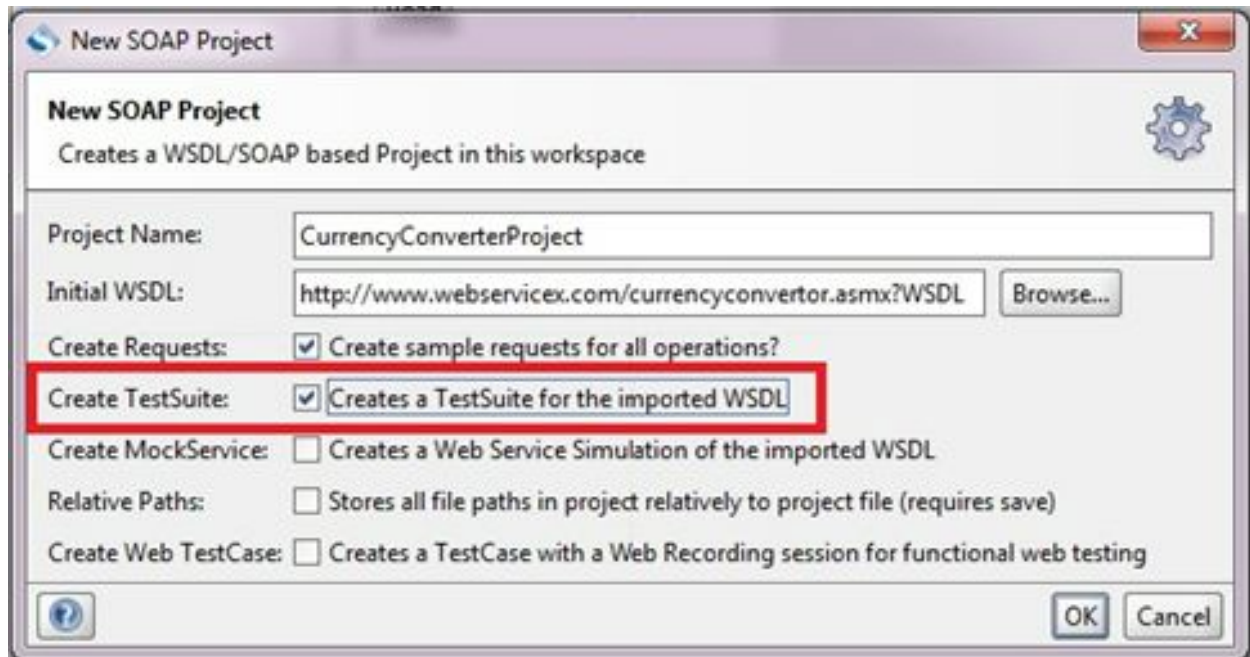
TestSuite, TestCase and TestStep in SoapUI:

A test suite is a common repository that contains a number of test cases. It is a collection of Test cases that represent the application flow. Test cases are the descriptive data about the application flow. Each test case contains individual actions called Test steps.

In SoapUI, test suite will be a root node that has to be created explicitly and test cases can be added under it and to test cases we can add test steps. It is a tree structure of sorts. If the test suites are well built, a bunch of webservices can be executed in one go. These test suites can be used for smoke, performance, regression testing etc. Once executed SoapUI Pro generates a report for analyzing results.

Adding a TestSuite during project creation:

#1. Click **New SoapUI Project** option (or press **CTRL + N**) from **File** menu. Check the options as above and click OK.

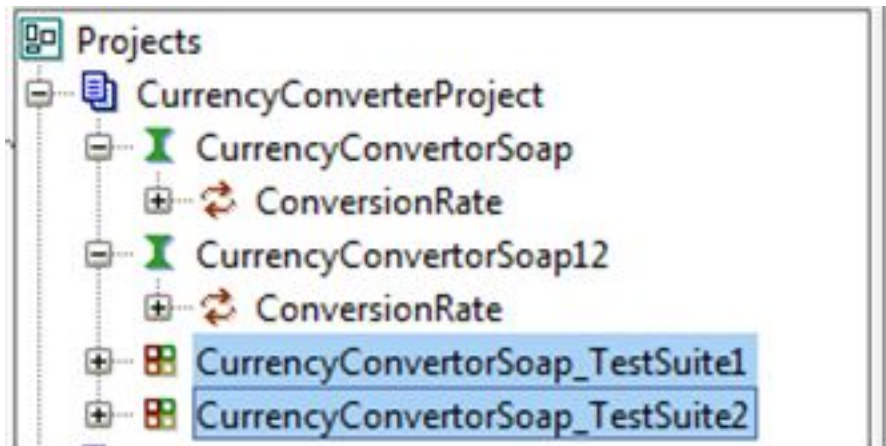


#2. Another pop-up to set the test case details would be displayed, set the properties as below, and click OK

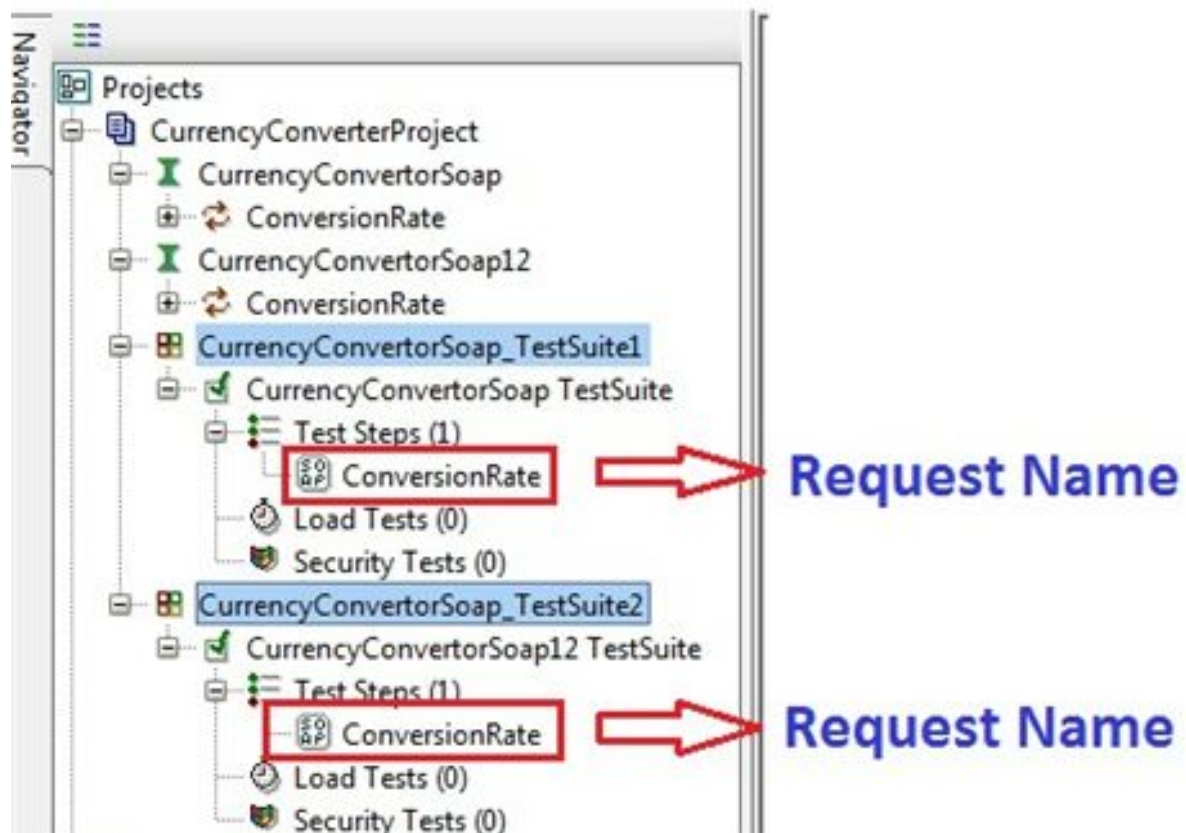
*#3. Enter the TestSuite name. By default, a sample name will be automatically assigned and that can be changed. Lets say it is: **CurrencyConvertorSoap_TestSuite1** and click OK*

#4. Based on the services count under the project, it will add that many test suites. Multiple test suites can be created.

#5. Finally the project tree will look like below after creating the test suites.



#6. Now we have two test suites. Each test suite will contain test steps, load test step and security test step as below:



As discussed earlier, test steps go under the test case. Within the test steps, the actual web service steps get added. If you double click on the service name, it opens the request and response sections in the right side of the navigator panel.

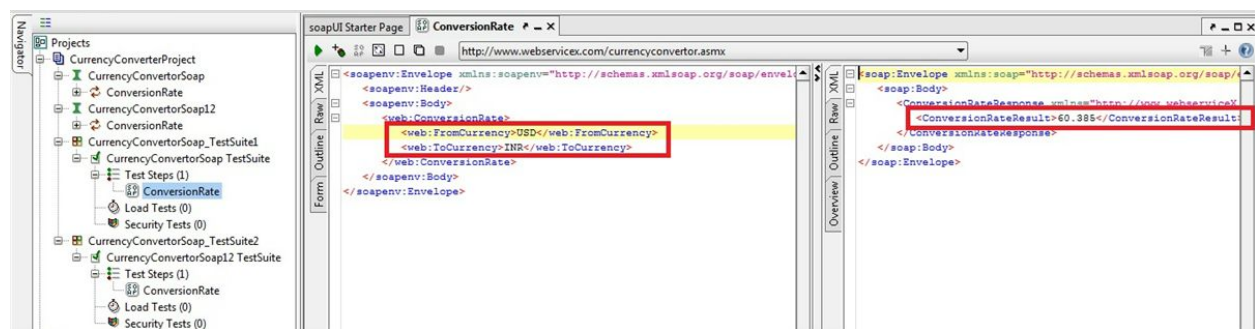
(Click on image for an enlarged view)



#7. In the input request, replace the '?' with valid input data.

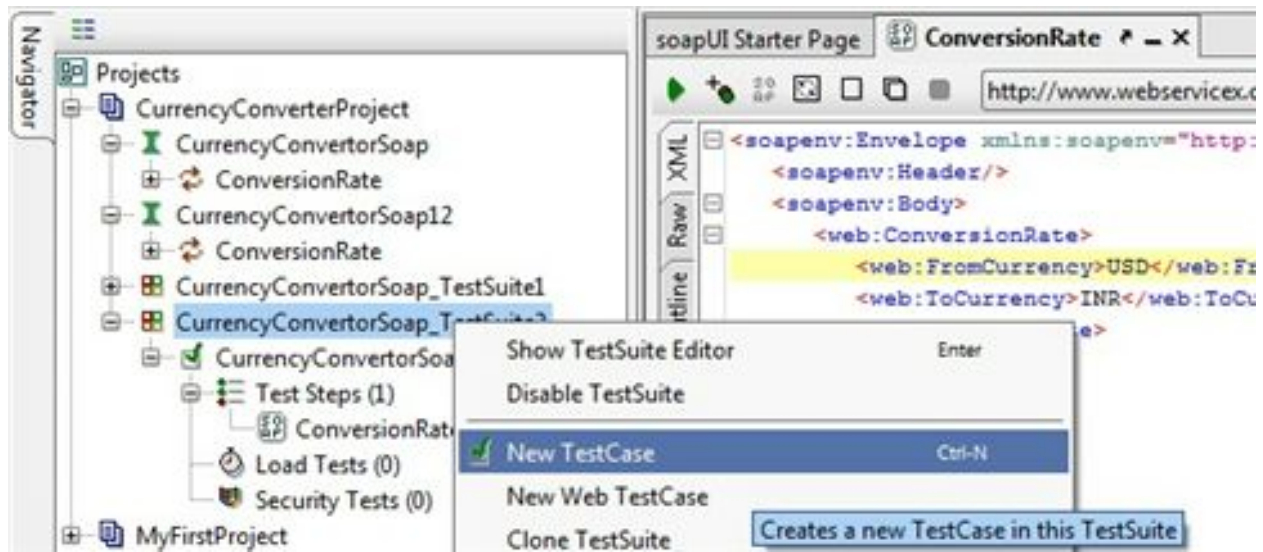
#8. Click run icon to execute the test suite. The response can be seen on the right side of the screen as below:

(Click on image for an enlarged view)

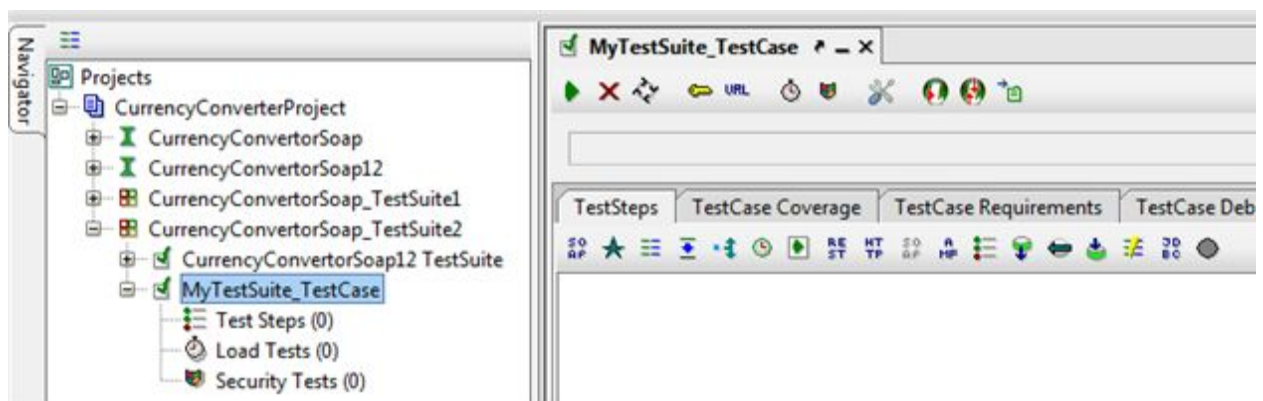
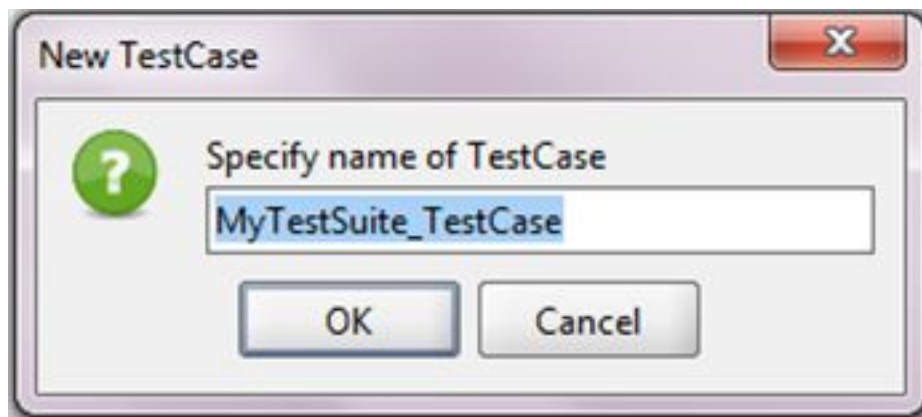


Adding new TestCases to already existing TestSuites:

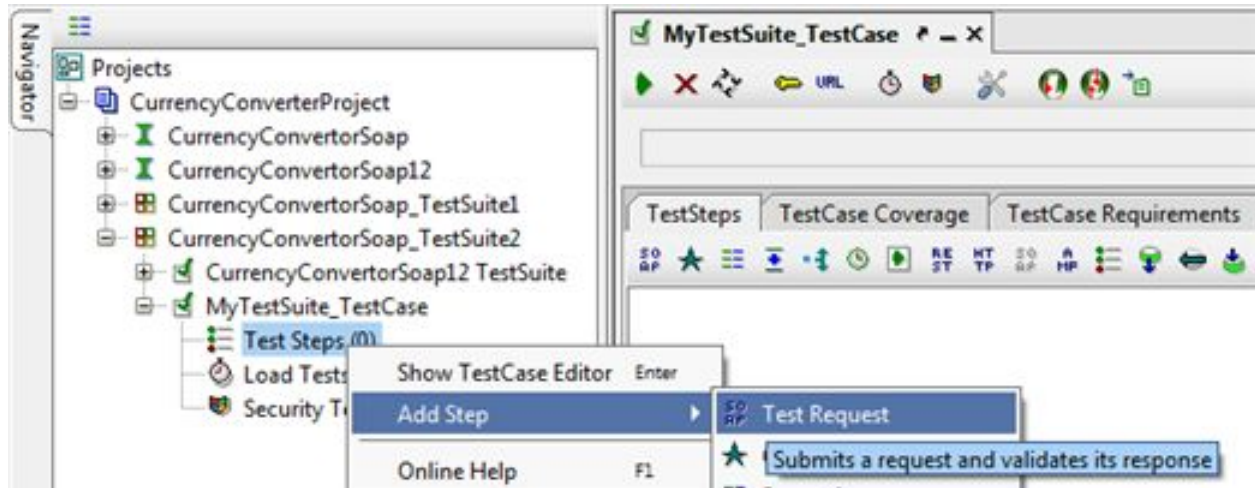
#1. Right click on the test suite name



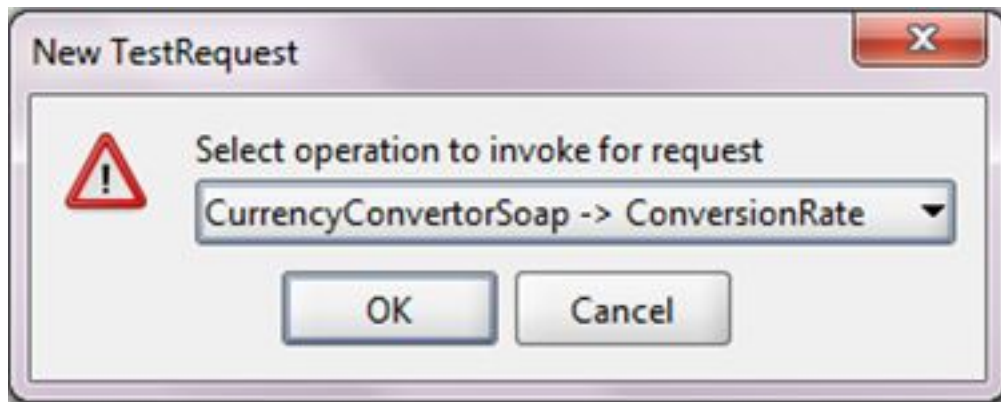
#2. Enter the TestCase name and click OK



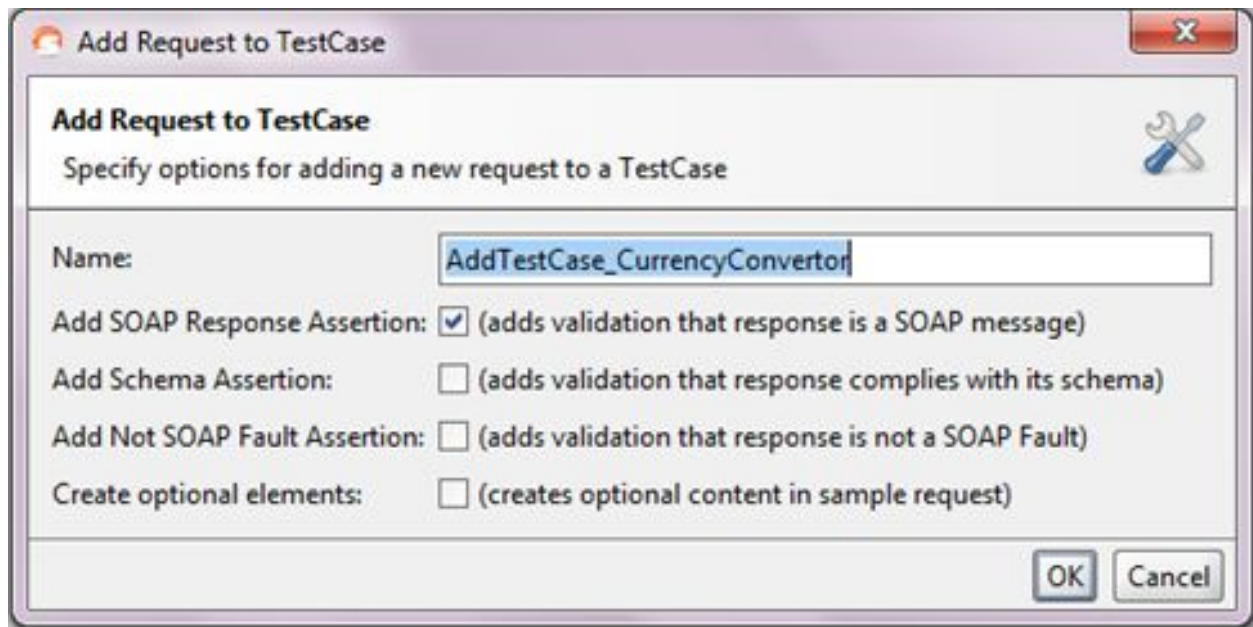
#3. Test steps can be added to the test case by Right clicking on the test steps and then click **Add Step: Test Request** option from the context menu as shown below and follow the steps through.



#4. After choosing the name, choose service name from the drop down if needed or it can be left empty and click OK



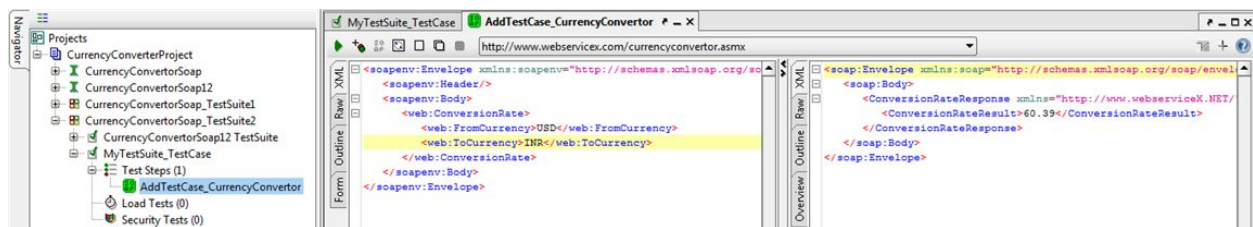
#5. In the following wizard, we can rename the request if required, with the other settings as default click OK



#6. The input request name can be seen under the test steps. When the request name is double clicked, the same input request and response section will open (Click XML tab to see the input and response requests).

#7. Enter the input data and execute the service to receive the response.

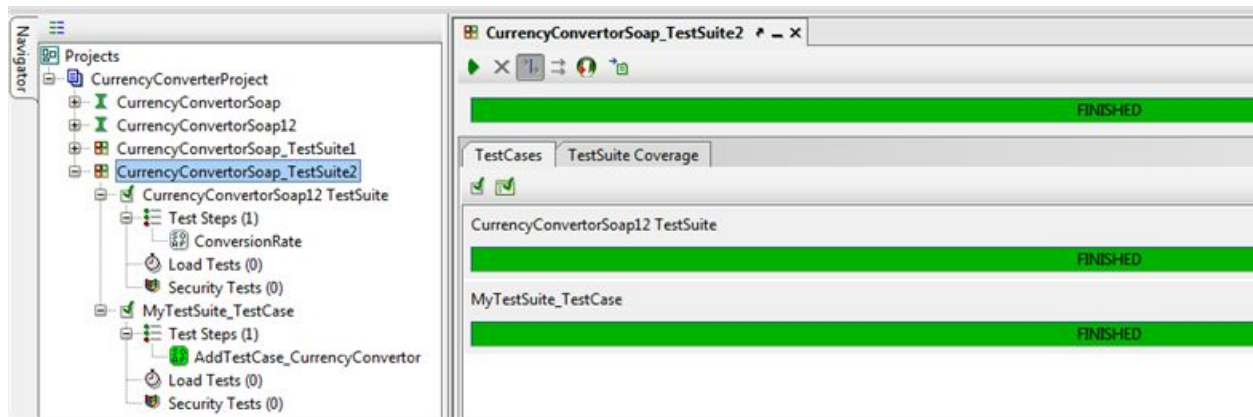
(Click on image for an enlarged view)



#8. To execute all the test cases together, double click on the test suite name and click Start Icon

The list of test cases in the test suite and their execution statuses can be seen as below.

(Click on image for an enlarged view)



Additional information:

- **Cloning objects:** Only Test suites, test cases and test steps can be cloned. Right click on the particular tree node and then click Clone test suite or test case or test step.
- **Rename or delete projects and its components:** Right click on the respective object and select "Rename" option from the context menu, enter a new name and click OK. To delete, choose the remove option from the menu and confirm the deletion. Once deleted, the operation cannot be undone.

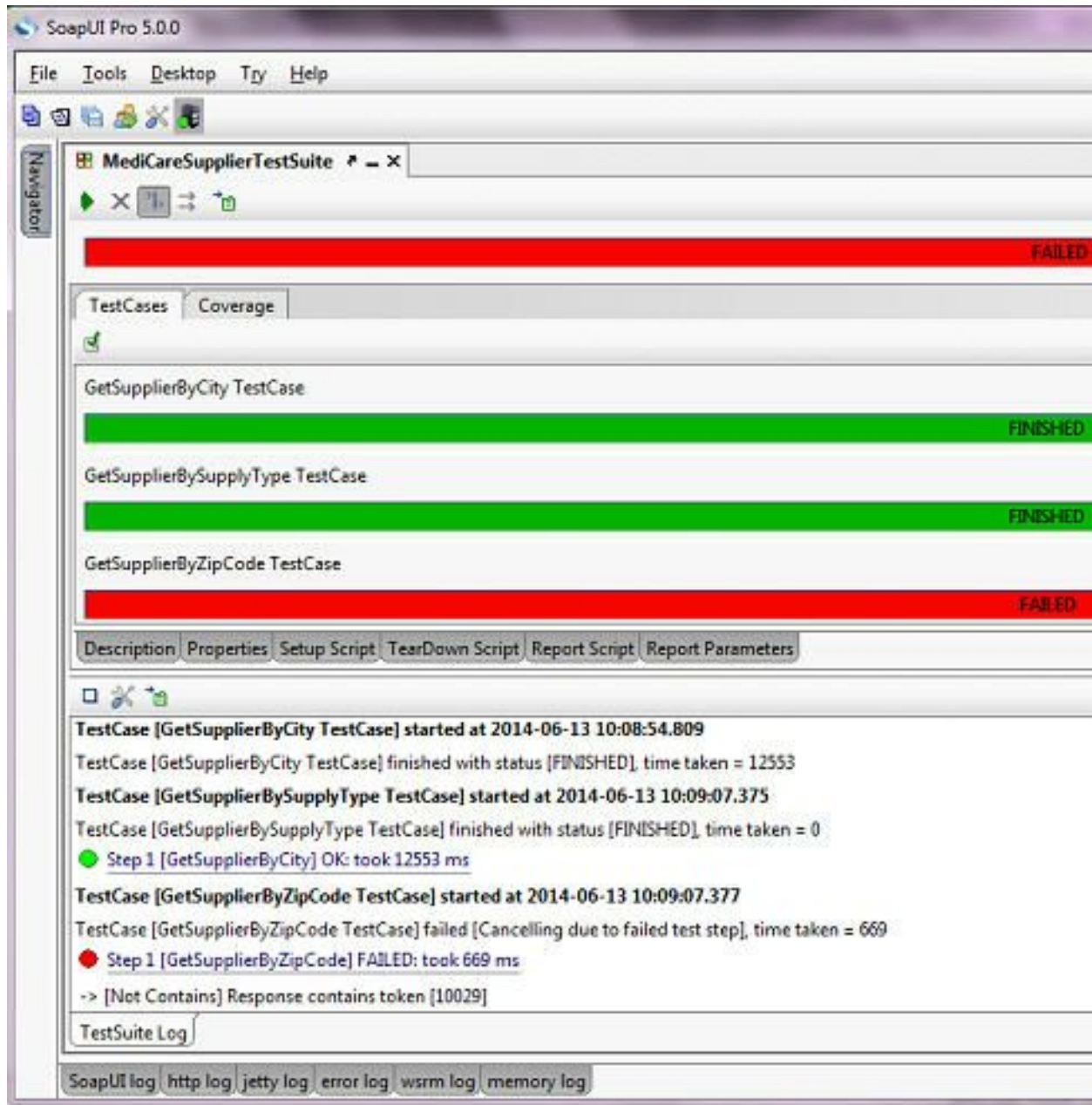
5. **Understanding Assertions in SOAPUI**

As with any testing, we have to compare what we want the system to do and what it actually is doing, to arrive at a certain validation or assertion, which is what it is called in the context of web services. As testers, it does not matter if we executed 1000 or even million test steps, but for us the result comparison is what determines the outcome of a test.

Therefore, we will spend this entire article on understanding how we can do that with SoapUI, although web services can be asserted manually. Also manual assertion is time consuming when there are multiple responses and responses with large data. SoapUI assertions are excellent at overcoming these shortfalls.

SOAPUI Assertions compare the parts/all of the response message to the expected outcome. We can add a variety of assertions provided by SoapUI to any test step. Each type of assertion targets specific validations on the response such as matching text, comparing XPATH or we could also write queries based on our need.

When the test steps get executed, then the associated assertions receive the response for the respective test steps. If any response is failed then the respective assertion will be processed and the corresponding test step will be marked as failed. This notification can be viewed in the test case view. Also we can find failed test steps in the test execution log. The sample test step assertion screen looks as below:



In the above image, some of the test steps have FAILED and some of them have PASSED. The reason is assertion.

As we discussed earlier, if the assert condition is not met with the expected results then the result is FAILED.

Working with different kinds of assertions in SoapUI:

Let us now see how to work with different types of assertions like:

- *Contains and Not Contains assertions*
- *XPath match and*
- *XQuery match assertions.*

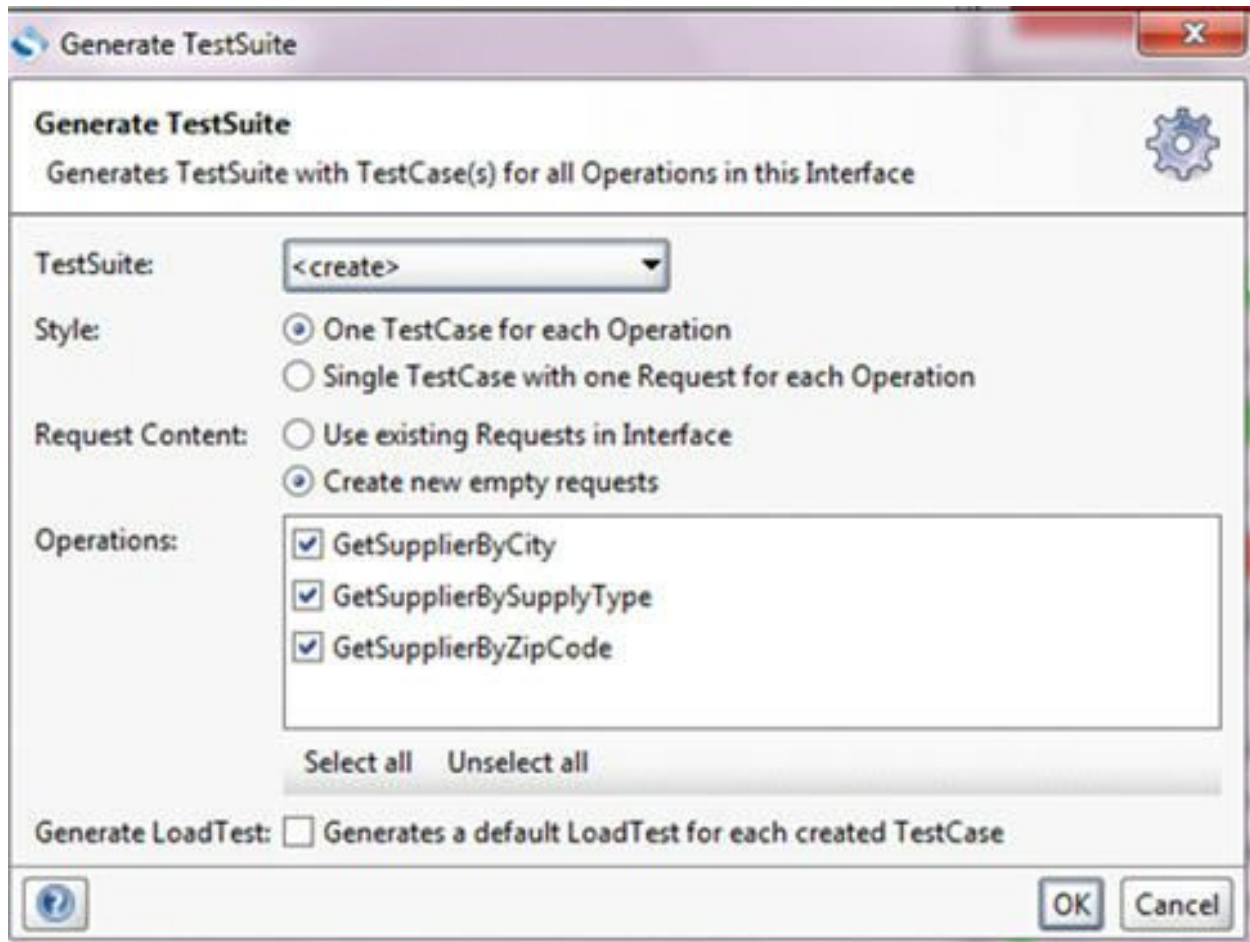
Firstly, we need a valid WSDL schema location. For that, we can use Medicare Supplier web service that is available at <http://www.webservices.net/medicareSupplier.asmx?WSDL>.

Follow the steps as below:

Step 1. *Create a new SOAP project by pressing CTRL + N and follow the steps. After creating the project, SOAPUI generates the list of interfaces and the corresponding requests.*

Step 2. *To add test suite to this project follow these steps:*

- *Right click on the interface name **MedicareSupplierSoap***
- *Click **Generate Testsuite** option from the context menu*
- *Click OK on the below window that comes up:*

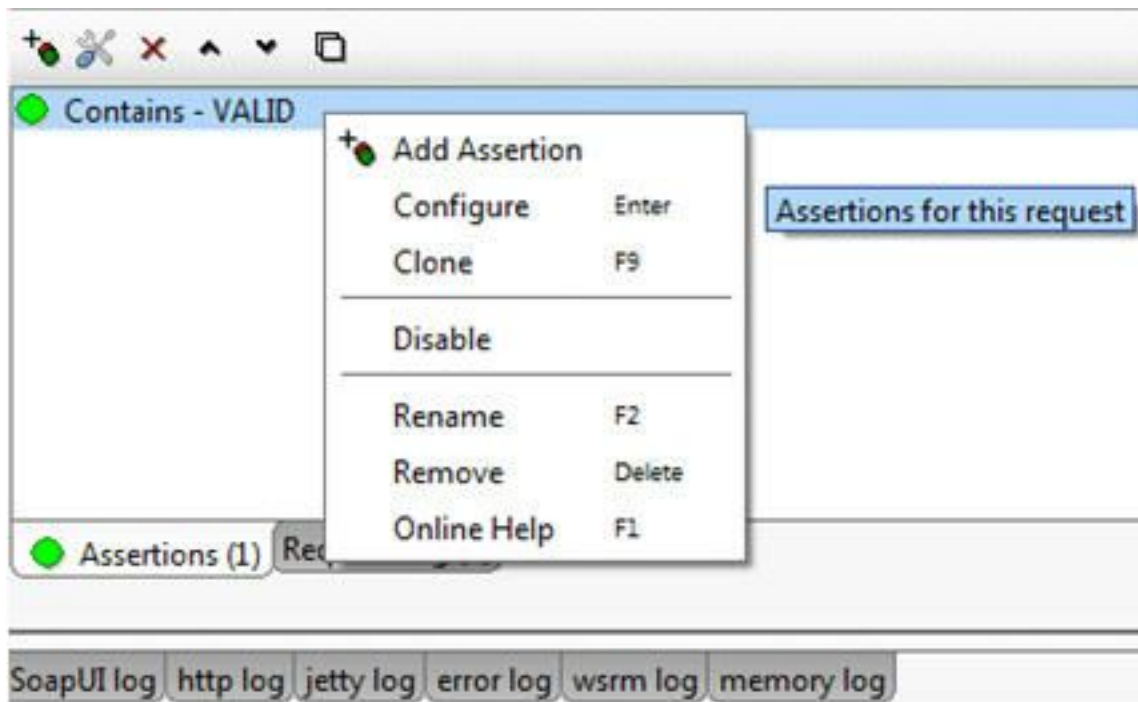


- In the next popup, need to enter your desired test suite name and click OK
- SOAPUI PRO will generate the test suite along with the requests in the navigator panel.
- Under the test suite, you will see some of the test steps with SOAP request step.

Step 3. To execute this test suite, double click on the request step and specify the input value in the respective location. For instance, open **GetSupplierByCity** request and enter **New York** between the city tags.

- Start this request by clicking on RUN icon – this will receive the response
- Now let's add assertions. For that, click **Assertions** tab present at the top of log tabs.

- On right clicking, a popup menu will appear with some basic assertion related options as below:

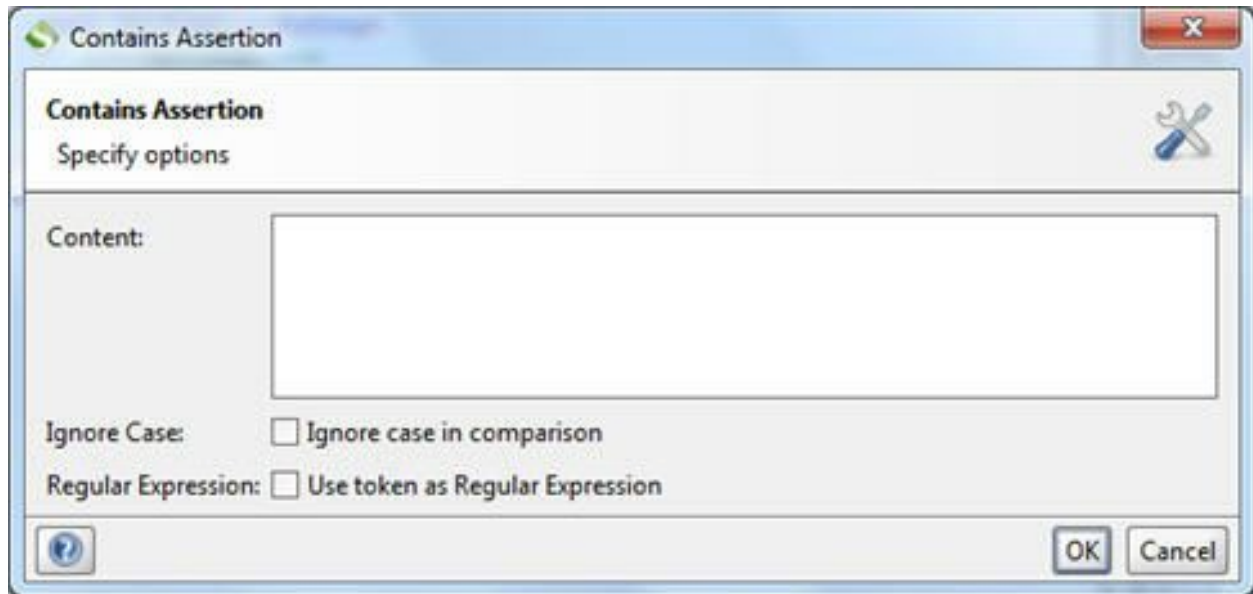


#1. Contains Assertion:

Click Add Assertion option or click it from the tool bar – **Add Assertion** window appears on the screen with different types of assertions.

1. Click **Property Content** category from the list – associated assertion types and their description is displayed
2. Click **Contains** assertion and click Add button
3. This is the assertion configuration window. Here itself we have to specify the expected condition based on the response.

For example, let me enter **New York** text in this text field. **Ignore case in comparison** checkbox will ignore even if the expected value is in upper case or lower case.



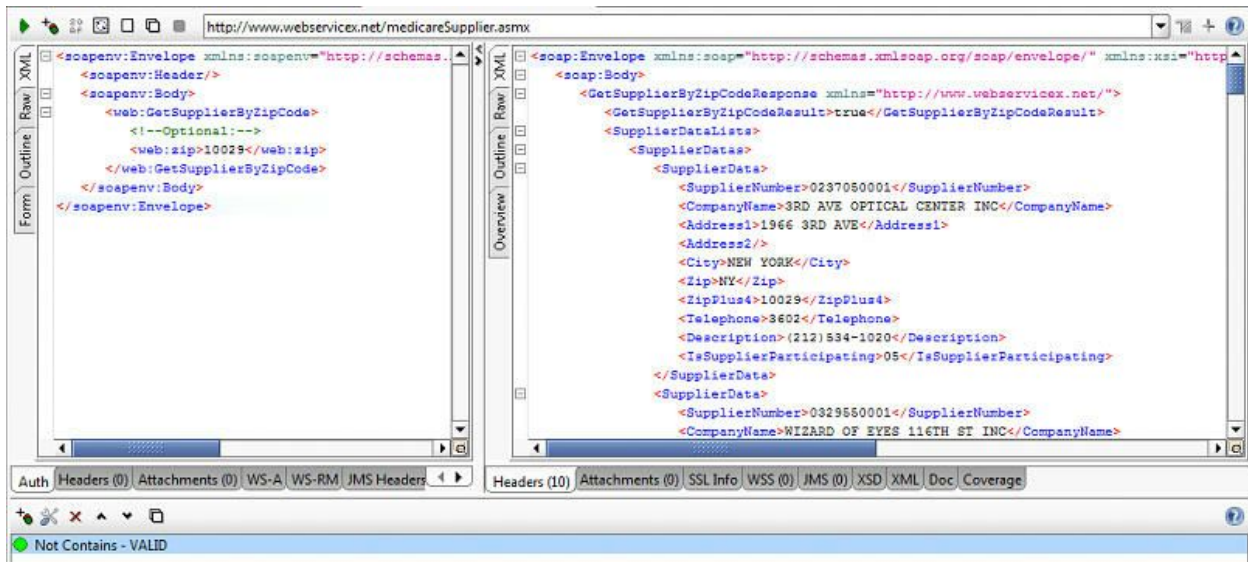
4. Now execute the test suite and verify the results. As you have seen in the test suite window, green indicates the successful execution and red denotes failure.

#2. Not Contains Assertion

We can use "not contains" assertion for validating request in negative scenarios. We can use **GetSupplierByZipCode** request to learn that.

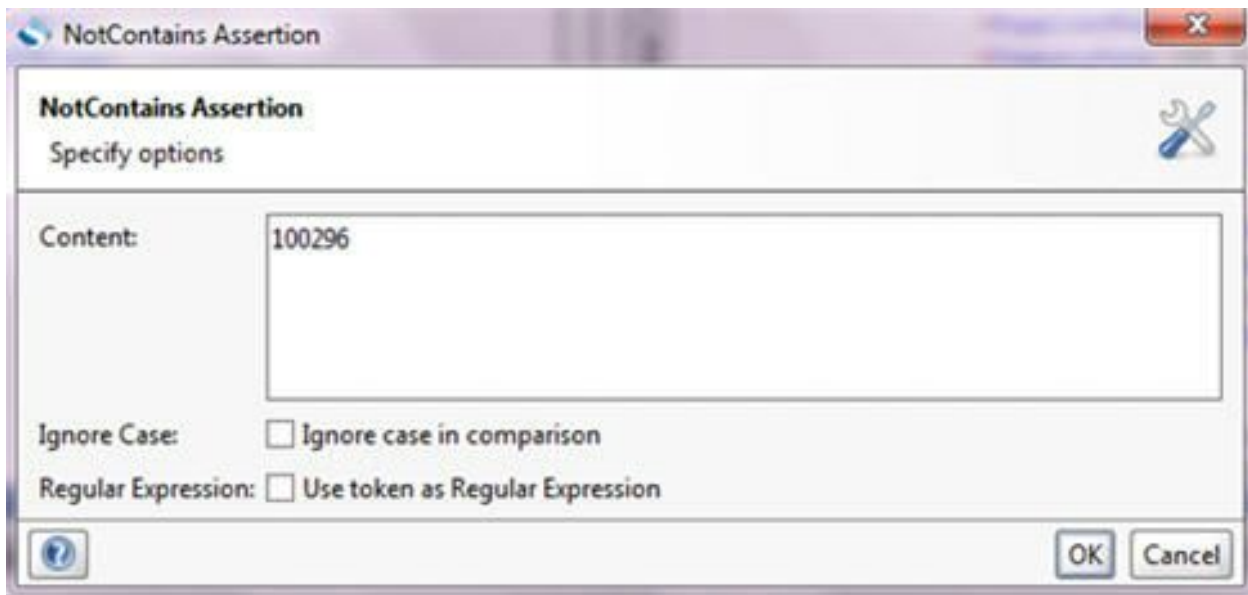
Open the request tab by double clicking on it. In the input request, enter invalid zip code in the appropriate location, such as **10029**. Run this request now. Check the response data which has the supplier details for the given zip code – take a look at the image below:

(Click on image for enlarged view)



The "not contains" assertion, it is highlighted in green color as it is executed successfully.

In the configuration window, we have configured with positive expected value as below:



It returns true if the expected conditional value is not found and returns false if the expected value is found in the response message.

Similarly we can change the condition and run the request once again. It generates the results accordingly.

#3. XPath Match assertion

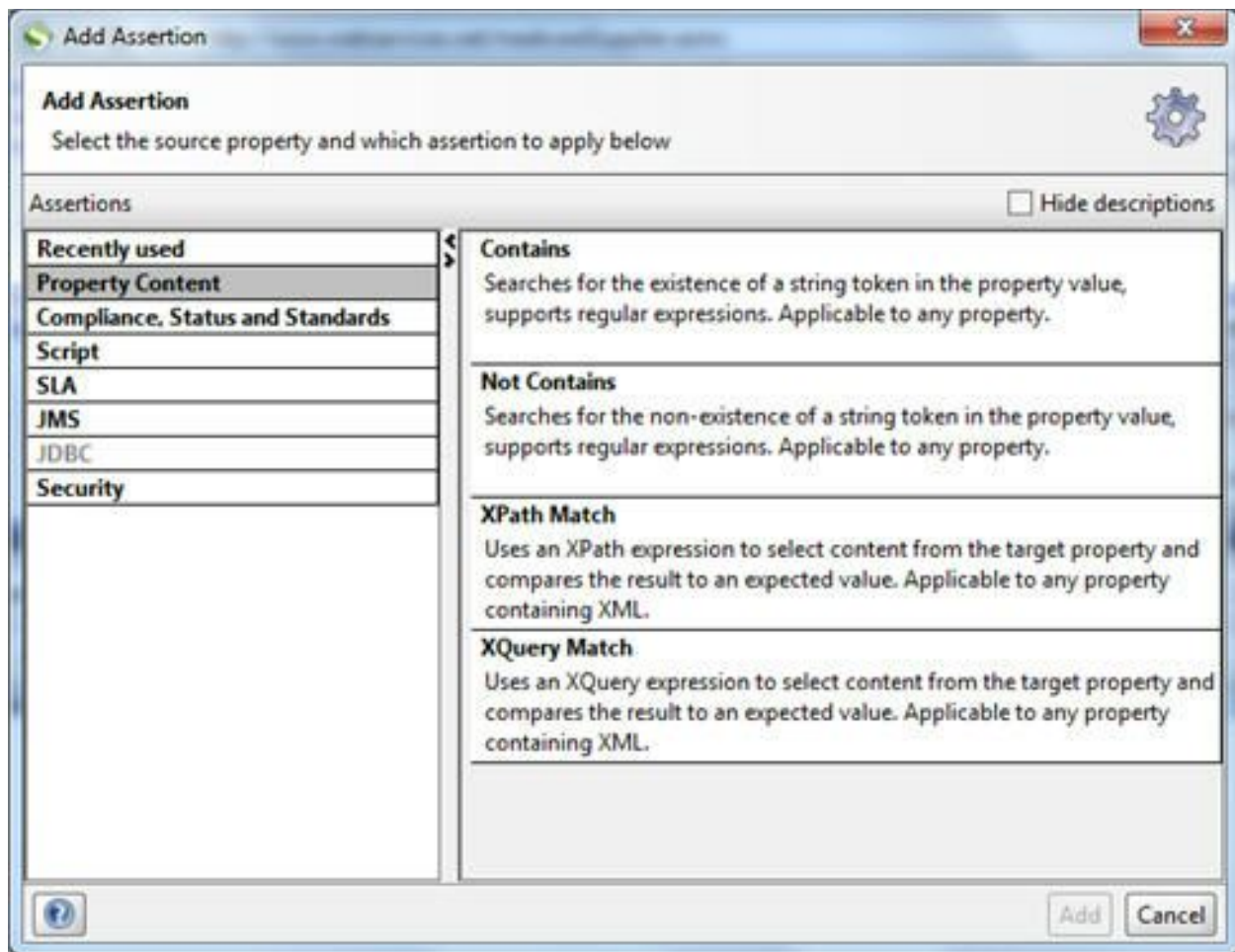
XPath match assertion is little bit different in terms that it will assert the response using actual response data.

For example, if we have login authentication web service that will authenticate the user credentials and send the acknowledgement to the client with some Boolean type of data which may be TRUE or FALSE in the form of XML.

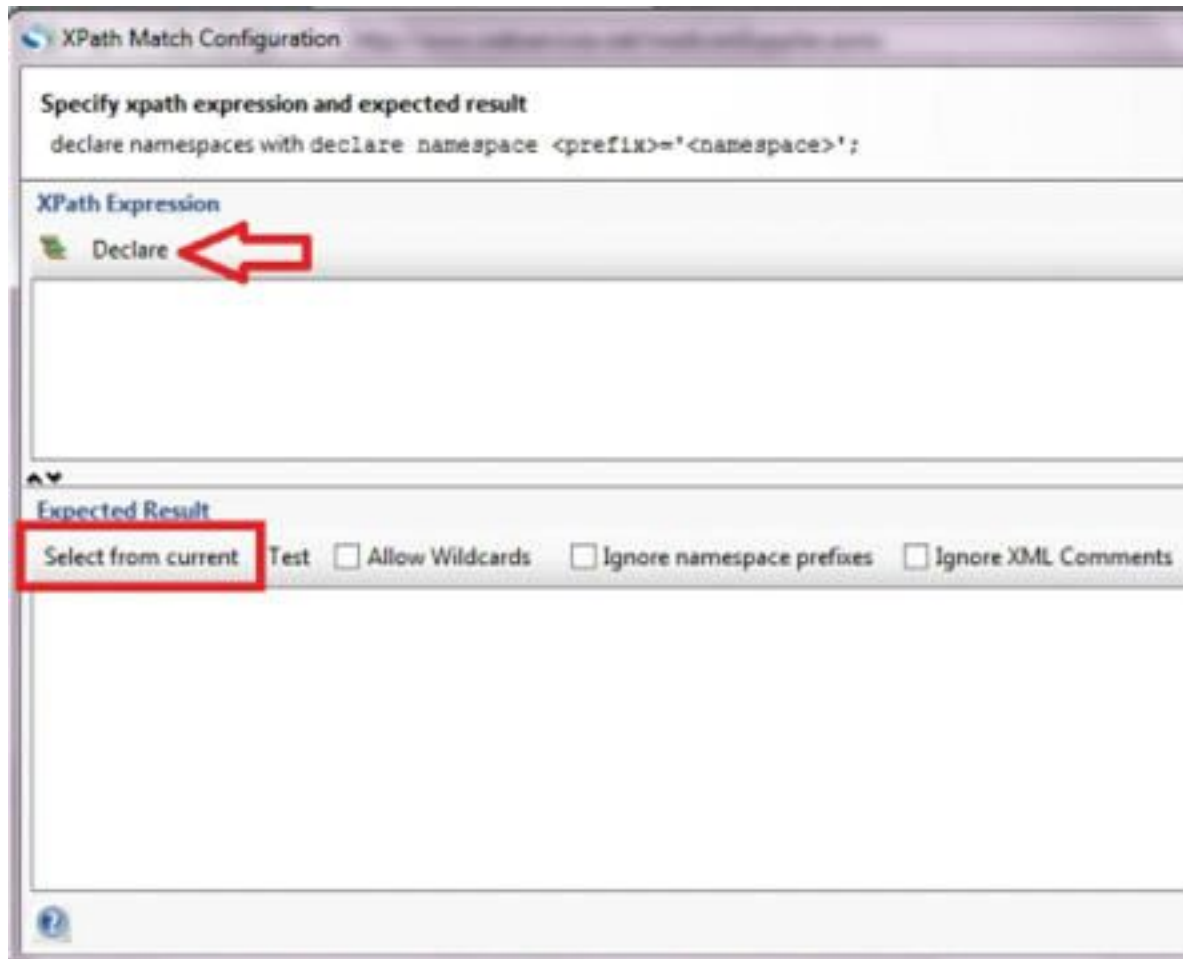
As you know XML documents are built by tags. So when specifying the expected value in the configuration, it should be in the form of XML.

Let try to do that:

Add one more assertion for the **GetSupplierByCity** request. In the Add Assertion window, click **Property Content** category and then click **xpath match** assertion.



The below window is displayed:



The top section is the declaration part and bottom section is expected result part.

When we click on the **Declare** option we will get some auto generated declaration scripts like below:

```
declare namespace soap='http://schemas.xmlsoap.org/soap/envelope/';
```

```
declare namespace ns1='http://www.websvicex.net/';
```

In the above scripts, first line denotes the response that should be XML data and enclosed SOAP tags. In the next line, the whole response will be assigned or copied into the **ns1** namespace variable during the execution. If we want to filter particular data from the whole response, we have to add the following script.

//ns1:SupplierData[1]

As you know, if you execute **GetSupplierByCity** request, it will produce the response which contains the list of supplier's personal data that belongs to **New York City**. Here, we have used **Xpath Match** expression to extract the specific supplier's personal details from the bulk response. For that purpose we have used **ns1** variable. Now click on the **Select from Current** button.

Then SOAPUI generates the following result:

```
1      <SupplierData
      xmlns="http://www.webservicex.net/"

2      xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"

3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

4      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      >

5

6      <SupplierNumber>0127051883</SupplierNumber>

7      <CompanyName>KMART
      CORP</CompanyName>

8      <Address1>250 W 34TH
      ST</Address1>

9      <Address2/
      >

10     <City>NEW
```

```

YORK</City>

11 <Zip>NY</Zip>

12 <ZipPlus4>10119</ZipPlus4>

13 <Telephone>0002</Telephone>

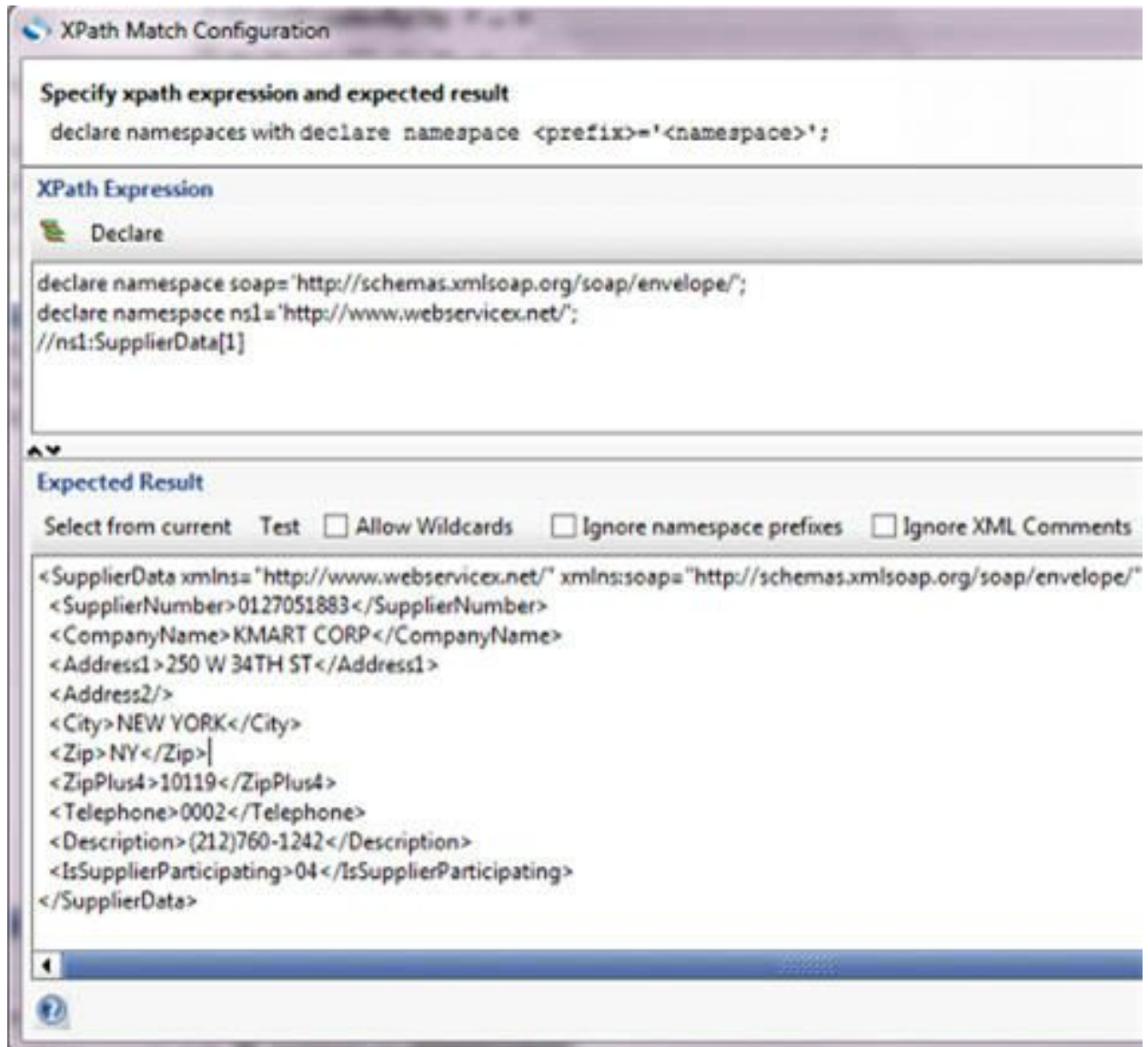
14 <Description><span class="skype_c2c_print_container
nottranslate">(212) 760-1242</span><span
id="skype_c2c_container"class="skype_c2c_container nottranslate"
dir="ltr"
tabindex="-1"onmouseover="SkypeClick2Call.MenuInjectionHandler.showMenu(this,
event)"onmouseout="SkypeClick2Call.MenuInjectionHandler.hideMenu(this,
event)"onclick="SkypeClick2Call.MenuInjectionHandler.makeCall(this,
event)" data-numbertocall="+12127601242"
data-numbertype="paid"data-isfreecall="false" data-isrtl="false"
data-ismobile="false"><span
class="skype_c2c_highlighting_inactive_common"
dir="ltr"skypeaction="skype_dropdown"><span
class="skype_c2c_textarea_span"id="non_free_num_ui"><span
class="skype_c2c_text_span">(212) 760-1242</span><spanclass="skype_c2c_free_text_span"></span></span></span></span></span></Description>

15 <IsSupplierParticipating>04</IsSupplierParticipating>

16 </SupplierData>

```


Please refer this screenshot:

The image shows a 'XPath Match Configuration' dialog box. It has a title bar with a blue icon and the text 'XPath Match Configuration'. The main area is divided into two sections. The top section is titled 'Specify xpath expression and expected result' and contains a text area with the text: 'declare namespaces with declare namespace <prefix>=<namespace>';'. Below this is a section titled 'XPath Expression' with a 'Declare' button and a text area containing: 'declare namespace soap="http://schemas.xmlsoap.org/soap/envelope/";', 'declare namespace ns1="http://www.webservice.net/";', and '//ns1:SupplierData[1]'. The bottom section is titled 'Expected Result' and contains a text area with an XML snippet: '<SupplierData xmlns="http://www.webservice.net/" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">', '<SupplierNumber>0127051883</SupplierNumber>', '<CompanyName>KMART CORP</CompanyName>', '<Address1>250 W 34TH ST</Address1>', '<Address2/>', '<City>NEW YORK</City>', '<Zip>NY</Zip>', '<ZipPlus4>10119</ZipPlus4>', '<Telephone>0002</Telephone>', '<Description>(212)760-1242</Description>', '<IsSupplierParticipating>04</IsSupplierParticipating>', and '</SupplierData>'. Below the text area are four checkboxes: 'Select from current', 'Test', 'Allow Wildcards', 'Ignore namespace prefixes', and 'Ignore XML Comments'. At the bottom left is a blue question mark icon.

Specify xpath expression and expected result

declare namespaces with declare namespace <prefix>=<namespace>;

XPath Expression

 Declare

declare namespace soap="http://schemas.xmlsoap.org/soap/envelope/";
declare namespace ns1="http://www.webservice.net/";
//ns1:SupplierData[1]

Expected Result

Select from current Test ☐ Allow Wildcards ☐ Ignore namespace prefixes ☐ Ignore XML Comments

<SupplierData xmlns="http://www.webservice.net/" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<SupplierNumber>0127051883</SupplierNumber>
<CompanyName>KMART CORP</CompanyName>
<Address1>250 W 34TH ST</Address1>
<Address2/>
<City>NEW YORK</City>
<Zip>NY</Zip>
<ZipPlus4>10119</ZipPlus4>
<Telephone>0002</Telephone>
<Description>(212)760-1242</Description>
<IsSupplierParticipating>04</IsSupplierParticipating>
</SupplierData>

Here in the response data, you can see only one supplier personal data. Based on the number that is present inside the angle brackets, the output will be generated.

By so far, this is about choosing the portion of the response that is required, how can/are we using **Xpath Match** assertion?

Let get to that: Click on the Save button once you are OK with the response.

Initially if you would have executed this service after configuring xpath match assertion with no changes, the result will be a successful response, status highlighted in green.

But let's change the input parameter in the input request to something that is invalid city -"XYZ or ABC". Run the request and check the results as well as the assertion status. We will get failure response and red status indication for assertion. Because we had already specified that the particular supplier data should be present in the service response in the expected result configuration and when the city name is invalid, that supplier is clearly not present.

This is how we can assert the XML response using Xpath Match expression assertion. Agreed that this is quite simple to start with but if you try with different service responses, you will get a much better idea.

We can also use aggregate functions in XPath Match expression. They are Sum, Min, Max, Count and Avg.

***For example,** if we want to know the total number of suppliers count in the expected results, write the following script.*

***count (//ns1:SupplierData)** and it returns **536** as a result. Remember all the aggregate functions should be in lowercase.*

#4. XQuery Match Assertion:

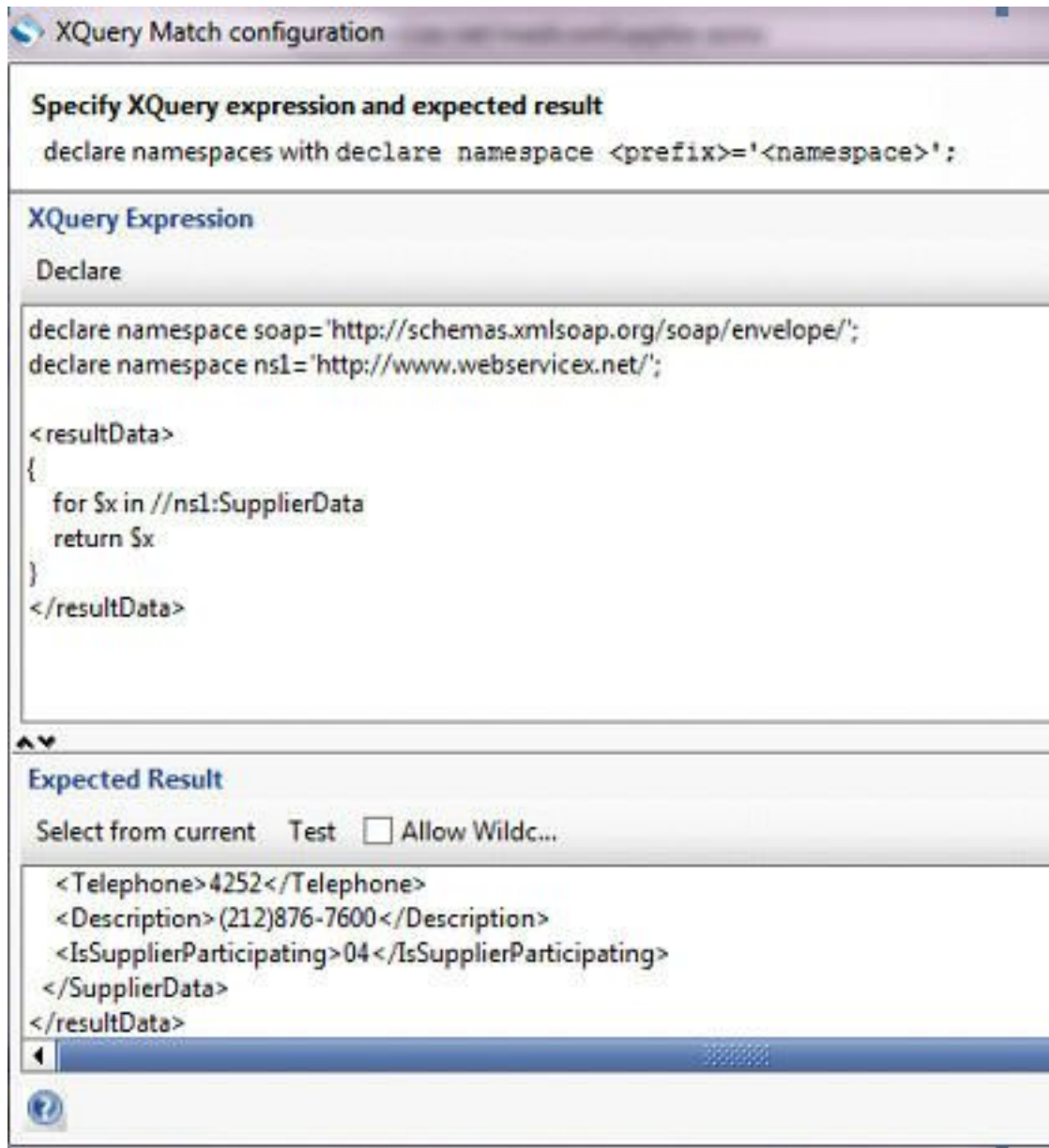
This is slightly similar to XPath Match assertion. As we have seen in the Xpath Match assertion configuration, there will be two sections – declaration and expected result.

- *Add XQuery Match assertion for the request*
- *In the configuration window, click **declare** button and write the following script*
- *Now click **Select from Current** button*
- *SOAPUI generates the response for the script*

XQuery expression also supports XPath Match expression but it has its own scripting syntax which cannot be used in XPath match assertion.

For Example:

We will see one example to fetch all the supplier data response using XQuery expression. Look at this sample screenshot to understand better.



Actual Script:

```
1      declare namespace
      soap='http://schemas.xmlsoap.org/soap/envelope/';

2      declare namespace
      ns1='http://www.webservices.net/';

3

4      <resultData
      >

5      {

6      for $x in
      //ns1:SupplierData

7      return
      $x

8      }

9      </resultData
      >
```

There are more built-in functions available to use in XQuery expression. They are **where**, **order by**, **for**, **return** and so on. To get more information about these functions, visit www.w3cschools.com website.

6. **Working with Operators**

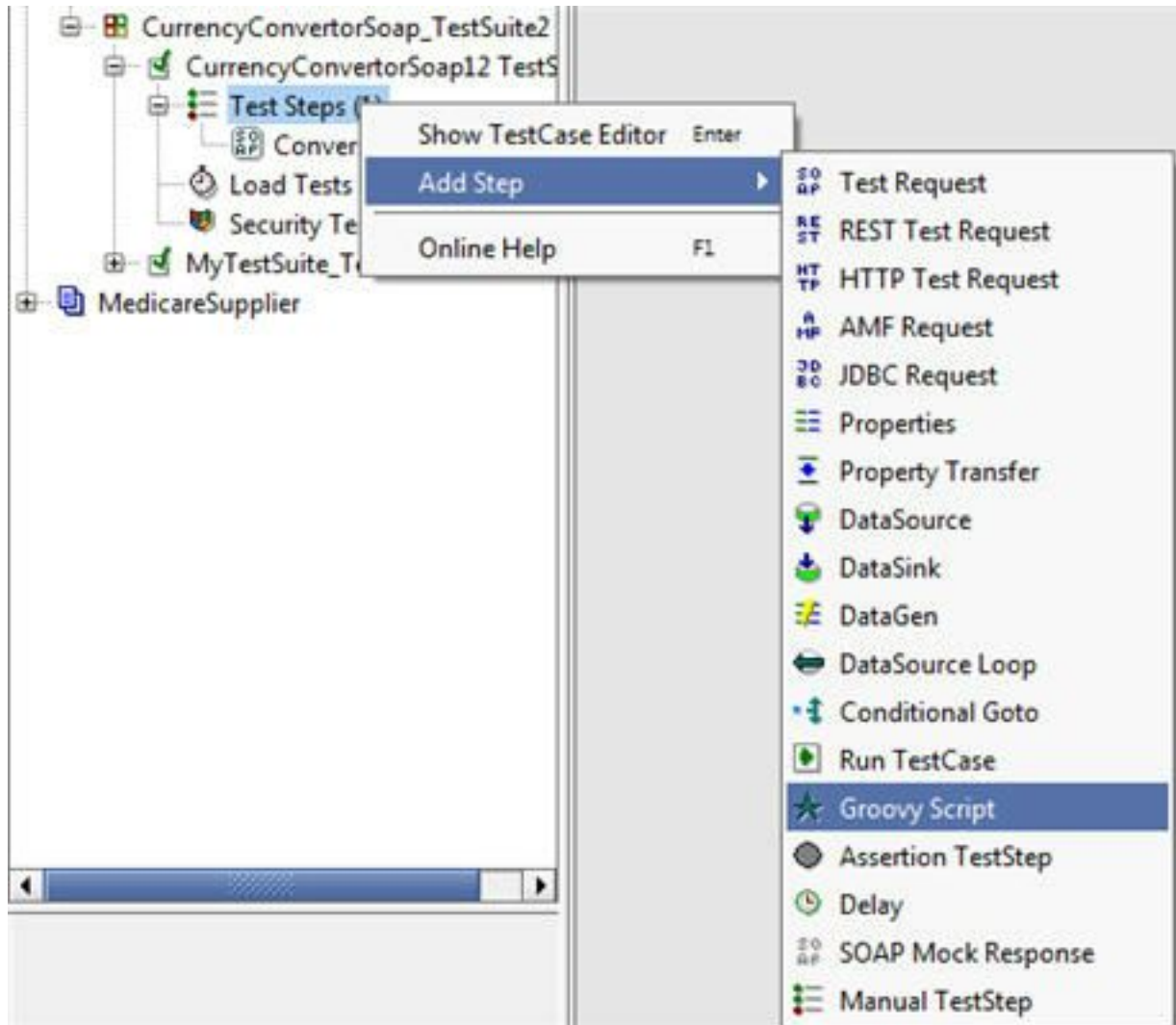
SoapUI Groovy Script Test step:

Groovy Script test step is included for custom automation test script creation in SoapUI / Pro. It can be used for functional/ load/regression.

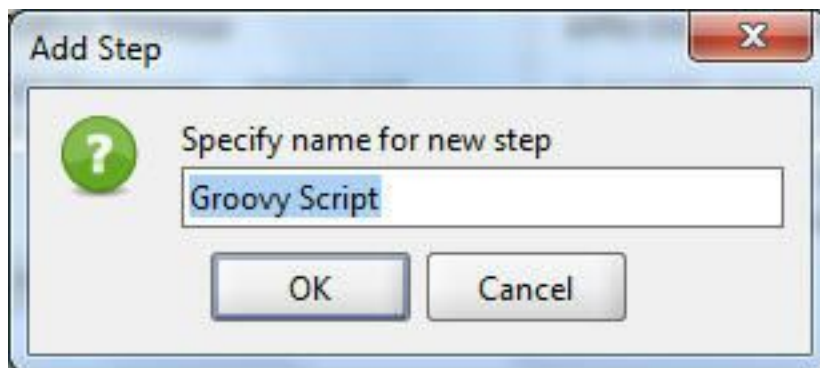
Groovy is a scripting language which internally includes all the java libraries, therefore all java related keywords and functions can be used in the groovy script directly. The Java libraries come with SoapUI and are integrated during the SoapUI Pro installation itself.

Here is how Groovy script can be added to a test:

Step #1. *In SoapUI Pro create a SOAP project with valid WSDL document. Under the project, create a test suite with a desired name. Inside the test suite, add groovy script test step as shown below:*

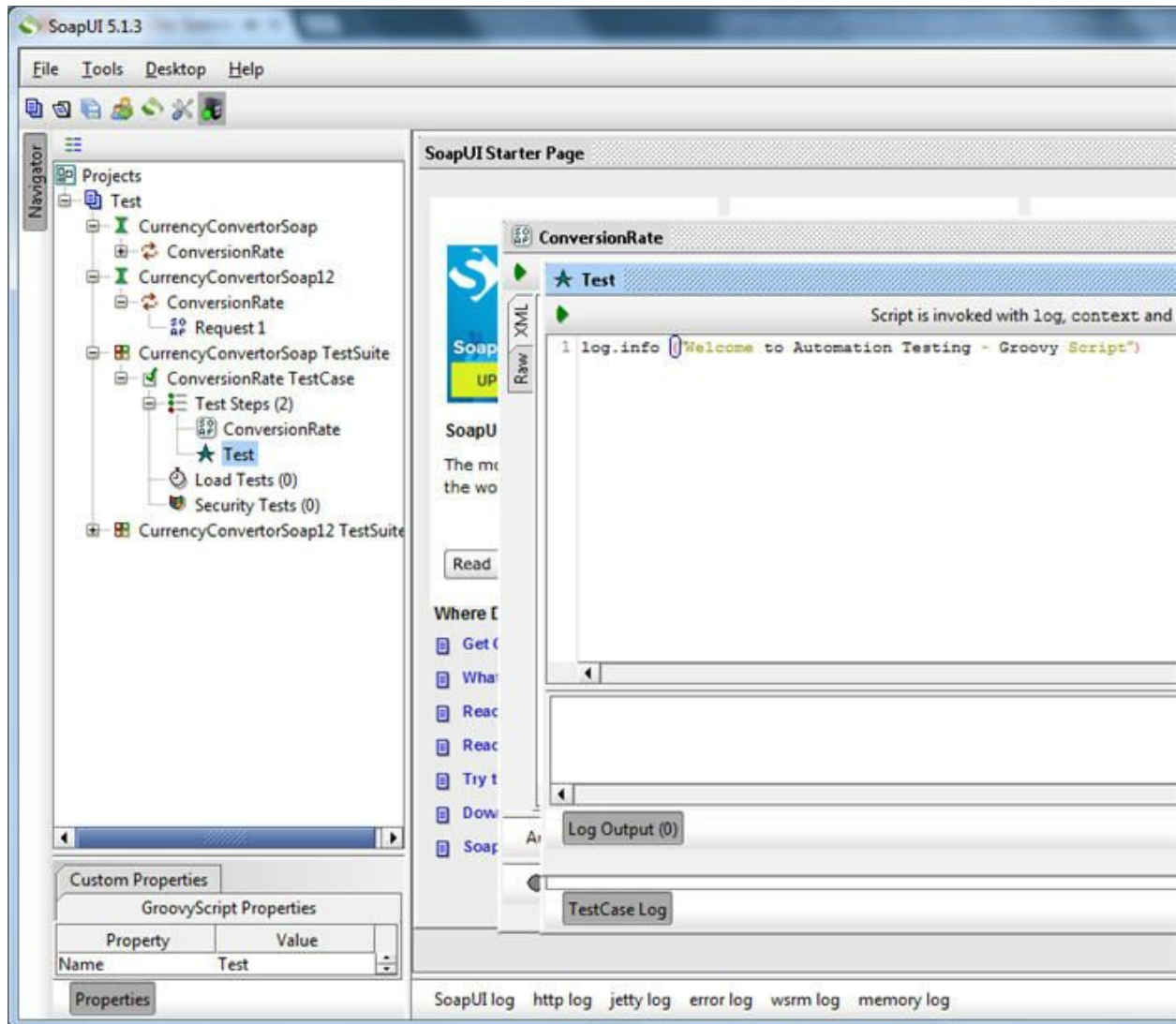


Step #2. Enter the name of the step in the dialog that comes up as below and Click OK



Step #3. An editor where you can write your script is displayed. Alternately you can double click on the groovy step name from your test case (Groovy step is the one that has a star prefix to it).

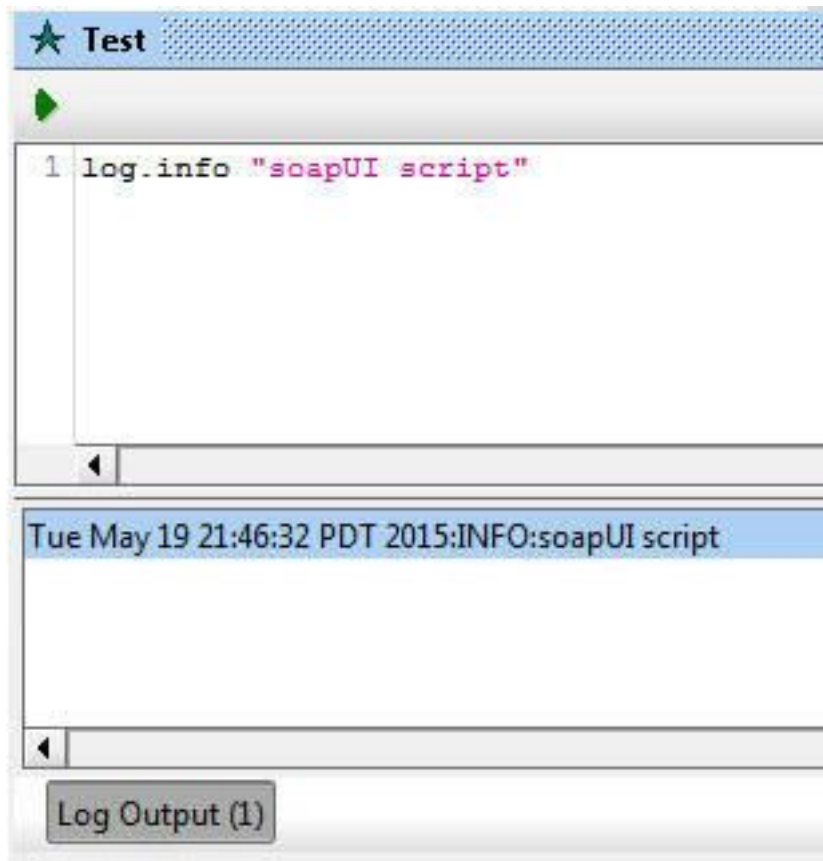
(Click on image for enlarged view)



For example: Let us write a simple script that shows a message in the log. Here is the one line script.

log.info "soapUI script"

Step #4. To execute the above script in SoapUI Pro, click on the Run icon and see the results in the Log Output section.



Few points:

- *Test script execution: When the run button inside the groovy editor is clicked, the code inside the groovy step will only get executed. On the other hand, when the Run button is clicked for the entire test case, all the steps are executed in an order.*
- *This way any kind of programming can be done to the test scripts to add validations as required.*
- *There can be any number of groovy test steps to a test case.*
- *With Groovy script it is not required to compile and interpret separately to execute the code like other programming languages such as C, C++, Java, etc.*

- Steps can be enabled or disabled inside a test suite by using the comment feature. To do so, use the following:

// – indicates single line comment and

/ <some script> */ – denotes multi-line comment*

Arithmetic Operations:

In the groovy step editor all of the below can be performed:

/ Adding Two numbers */*

int a;

int b;

int c;

// Assigning integer value to the variables A and B

a = 100;

b = 200;

// Adding A value and B value and assign the resultant value to the variable C

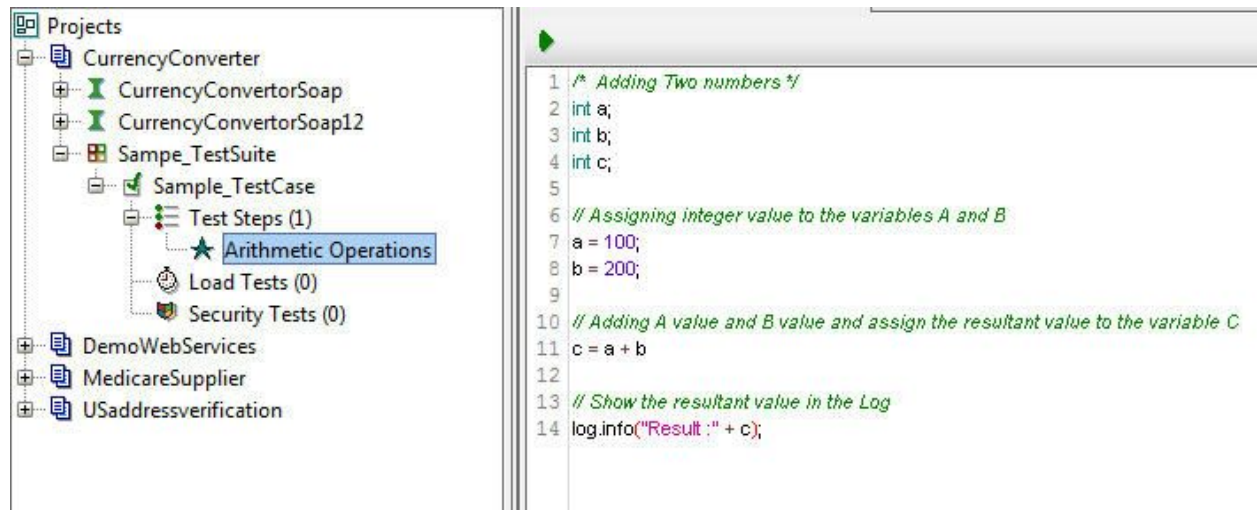
c = a + b

// Show the resultant value in the Log

log.info("Result : " + c);

In the above script, A, B and C are the variables which are used to store or transfer the values.

(Click on image for enlarged view)



Note: Variables in Groovy script are case sensitive. Exercise caution when using them.

The following are the operators supported in Groovy:

Arithmetic Operators:

+ Addition operator / String concatenation

- Subtraction operator

***** Multiplication operator

/ Division operator

% Remainder operator

// Arithmetic Operators Samples

// Addition Operator

```
int x1 = 100 + 200
```

```
log.info ("Addition Result :" + x1);
```

```
// Concatenation of Two Strings using PLUS ( + ) operator
```

```
String city ="Timothy E." + " Shepherd";
```

```
log.info("String Concatenation:" + city);
```

```
// Subtraction Operator
```

```
int x2 = 200 - 100
```

```
log.info ("Subtraction :" + x2);
```

```
// Multiplication Operator
```

```
int x3 = 10 * 200
```

```
log.info ("Multiplication :" + x3);
```

```
// Division Operator
```

```
int x4 = 200 / 10
```

```
log.info ("Division :" + x4);
```

```
// Modulus Operator
```

```
int x5 = 10 % 3
```

```
log.info ("Reminder or Modulus:" + x5);
```

The following is a screenshot of all of the above scripts and the respective results:

```

1 // Arithmetic Operators Samples
2 // Addition Operator
3 int x1 = 100 + 200
4 log.info ("Addition Result :" + x1);
5
6 // Concatenation of Two Strings using PLUS ( + ) operator
7 String city ="Timothy E." + " Shepherd";
8 log.info("String Concatenation:" + city);
9
10 // Subtraction Operator
11 int x2 = 200 - 100
12 log.info ("Subtraction :" + x2);
13
14 // Multiplication Operator
15 int x3 = 10 * 200
16 log.info ("Multiplication :" + x3);
17
18 // Division Operator
19 int x4 = 200 / 10
20 log.info ("Division :" + x4);
21
22 // Modulus Operator
23 int x5 = 10 % 3
24 log.info ("Reminder or Modulus :" + x5);

```

```

Mon Jul 21 11:31:01 IST 2014:INFO:Addition Result :300
Mon Jul 21 11:31:01 IST 2014:INFO:String Concatenation:Timothy E. Shepherd
Mon Jul 21 11:31:01 IST 2014:INFO:Subtraction :100
Mon Jul 21 11:31:01 IST 2014:INFO:Multiplication :2000
Mon Jul 21 11:31:01 IST 2014:INFO:Division :20
Mon Jul 21 11:31:01 IST 2014:INFO:Reminder or Modulus :1

```

Log Output (6)

Unary Operators:

Unary operators are the ones that work with only one operand. For example: **++** – it is called as **Increment operator** which increments the current value by 1

Here's the example:

```
int A = 100;
```

```
A++;           // Equivalent to A = A + 1
```

```
log.info (A);
```

The above script will produce the output as 101. This increment operation is called post increment. Similarly we can use this operator as a pre-increment operation as below:

```
int A = 100;
```

```
log.info (++A);
```

There is also (-) the decrement operator. It will decrease the current value by **1**. We can implement this operator to the above discussed examples.

```
int A = 100;
```

```
A--;           // Equivalent to A = A - 1
```

```
log.info (A);
```

The above script will produce the following output:

```
Mon Jul 21 18:02:16 IST 2014:INFO:99
```

The pre and post operations can be used with decrement operator as well.

Assignment Operators:

*The basic assignment operator is equal sign (=). Likewise, there are other useful assignment operators available. They are +=, -=, *=, /=, %=.*

Let us see the samples.

```
int A=100;
```

```
A += 10;           // Similar to A = A + 10
```

```
log.info(A);
```

The above script produces 110. If we use minus equal to operator in the below script, output will be 40.

```
int B=50;
```

```
B -= 10;
```

```
log.info(B);
```

Likewise we can use the remaining operators like this.

```
int C=10;
```

```
C *= 10;
```

```
log.info(C);
```

And,

```
int D=50;
```

```
D /= 10;
```

```
log.info(D);
```

Here's the remainder operator is used as

```
int E=10;
```

```
E %= 3;
```

```
log.info(E);
```

This will divide the value 10 by 3 and the remainder will be assigned to the variable "E".

7. **Dealing Properties with Groovy Script** (must read)


Properties are the central repository to store our information temporarily. These can contain login information like username and password, session data like session id, page context, header information and so on.

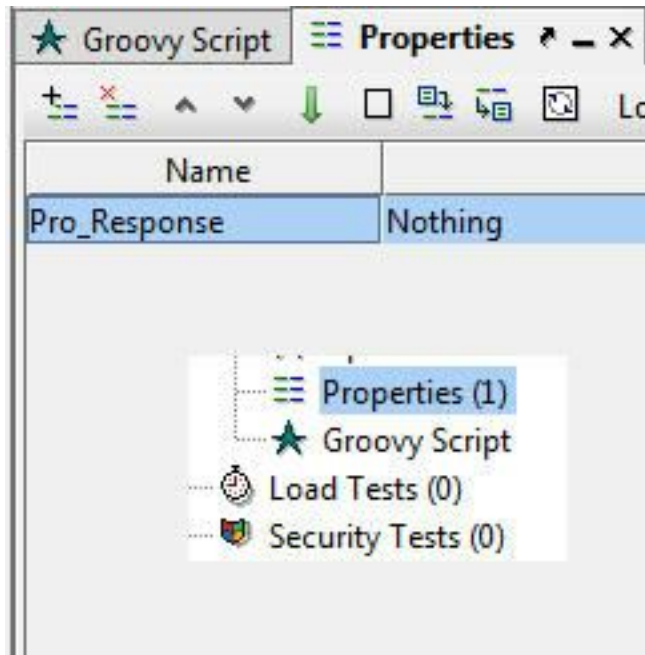
This is the 7th tutorial in our SoapUI free online training series.

Let's see how to add property test step and then we will discuss assigning values to the property and show them in the log.

How to Add Properties in SoapUI:

Here are the steps.

- *Right click on the Teststeps node*
- *Click **Add Step** and **Properties** option from the context menu*
- *Enter the property name as desired and click OK*
- *In the properties screen, click  icon to add property*
- *Enter your desired property name and click OK button. For example let me enter **Pro_Response***
- *Type any default value for the property if you wish. For example, I enter "Nothing"*
- *Then, add a Groovy Script test step next to the property step. Refer below screen shot.*



We can transfer the property data across the test steps during the test execution. For that, SoapUI Pro provides Property Transfer test step. Look at the below screenshot.



In the groovy script, add the following script. This script will assign a string text to the property and then it will show in the log after executed the test case.


```

1      String testString =
      "TestString"

2      testRunner.testCase.setPropertyValue(
      "Pro_Response", testString )
  
```

```
3      def getLocalPropValue =  
        testRunner.testCase.getPropertyValue("Pro_Response")
```

```
4      log.info(getLocalPropVal  
ue)
```

- Once written the above script in the editor, double click on the test case name step.
- Run the test case by clicking on the  icon and the see the results in the script log tab.

Accessing property:

There are several ways to access test case, test suite and project properties for setting and getting their data through the script. Here are the samples for retrieving the property data.

```
1      def  
getTestCasePropertyValue =
```

```
2      testRunner.testCase.getPropertyValue(  
"LocalPropertyName")
```

```
3      def  
getTestSuitePropertyValue =
```

```
4      testRunner.testCase.testSuite.getProperty  
Value
```

```
5      ( " LocalPropertyName "  
)
```

```
6      def  
getProjectPropertyValue =
```

```
7      testRunner.testCase.testSuite.project.getPropert  
yValue
```

```
8      ( " LocalPropertyName "
      )
```

In order to access a global property, this is the script:

```
1      def
      getGlobalPropertyValue =

2      com.eviware.soapui.SoapUI.globalProperties.getPropertyValue

3      ( "GlobalPropertyName"
      )
```

These script lines are used to set the value to the local and global property.

```
1      testRunner.testCase.setPropertyValue( "
      LocalPropertyName ", someValue )

2      testRunner.testCase.testSuite.setPropertyValue( "
      LocalPropertyName ", someValue )

3      testRunner.testCase.testSuite.project.setPropertyValue( " LocalPropertyName ", someValue )

4      com.eviware.soapui.SoapUI.globalProperties.setPropertyValue

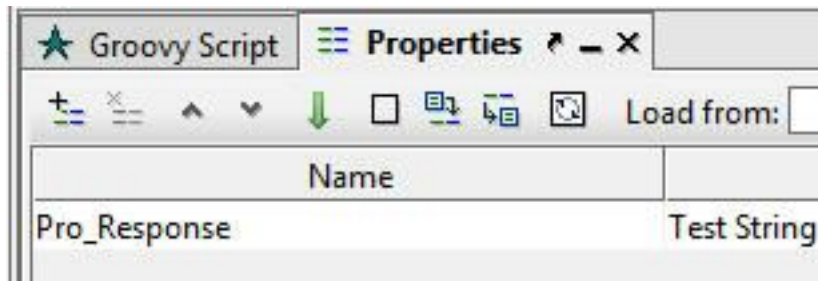
5      ( " GlobalPropertyName ",
      someValue )
```

*Here in these scripts, **testRunner** is common object which might be test suites, test cases or project. **setPropertyValue** and **getPropertyValue** are the methods or functions.*

As we mentioned the above script, we can assign data to the properties.

testRunner.testCase.testSteps["Properties"].setPropertyValue("Pro_Response", testString)

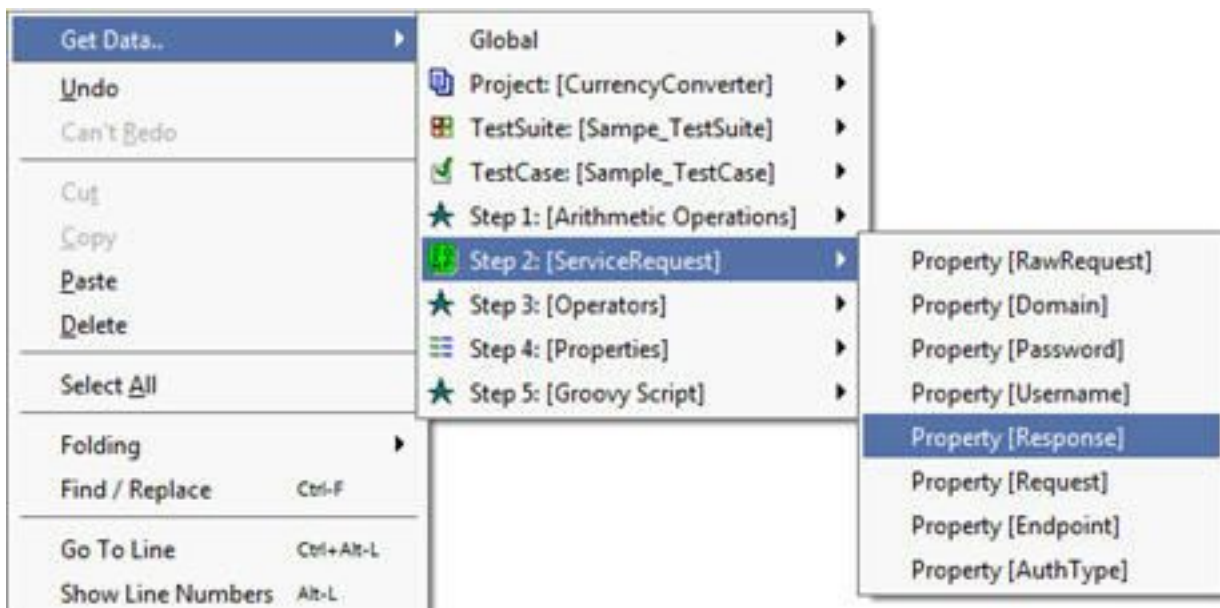
After executing the above script, the property will get updated in the property test step. Refer the following screenshot.



Receiving response data:

Now let us discuss how to get the response data through the script. To do this,

- Execute the service request once and verify the result
- Go to Groovy script editor and then right click on the editor as shown in the below screenshot



Now SoapUI Pro generates the script as below after specifying the property name.

```
def response = context.expand( '${ServiceRequest#Response}' )
```

As we know, "**def**" is a groovy script keyword that represents defining properties / objects. By default, SoapUI Pro has the property name as "**response**" in the **Get Property** popup. If we want we can change this name. Remaining portions of the script are auto generated.

Let us merge the above script in our earlier discussed script. Here's what you would see:

```
1      def response =
      context.expand( '${ServiceRequest#Response}' )

2      testRunner.testCase.setPropertyValue(
      "Pro_Response", response )

3      def getLocalPropValue =
      testRunner.testCase.getPropertyValue("Pro_Response")

4      log.info(getLocalPropVal
      ue)
```

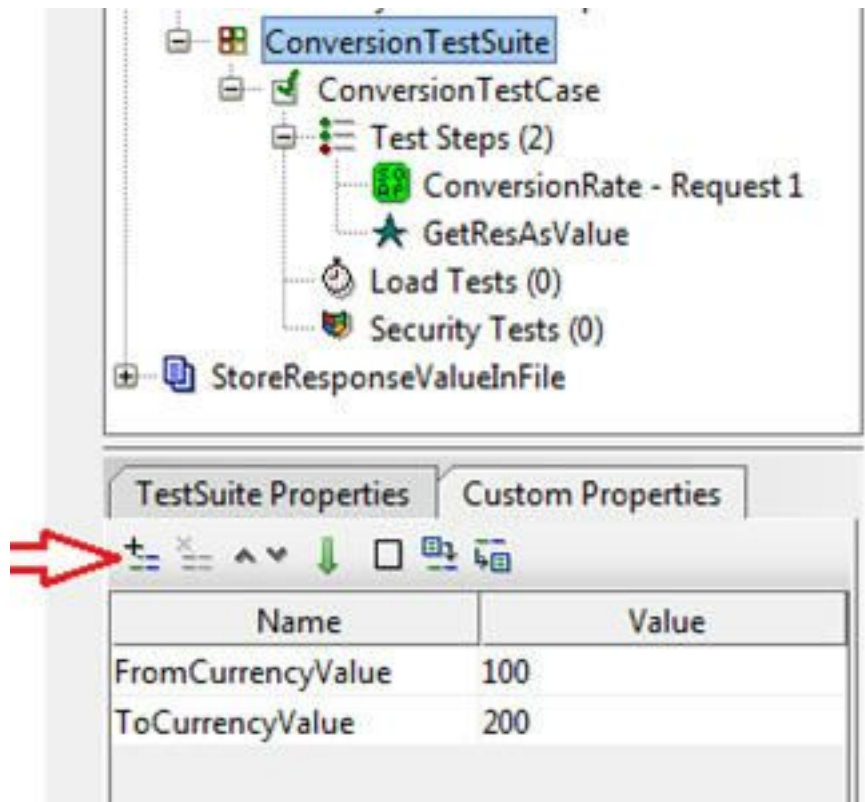
If we execute the above script separately, it will log the entire response data in the **log** section. Even when execute this along with the test case, it will show the same output in the **script log**.

Creating properties from the navigator pane:

There is another way to create properties locally through the property panel which will be appear when we click on the nodes under the project tree. Let's see how:

- Add currency converter service request and a groovy script test step under the test suite **ConversionTestSuite**.
- Click on the **TestSuite** name under the project (i.e. **ConversionTestSuite**)
- At the bottom of the Navigation panel, we can see a Property panel. It contains TestSuite **Properties** and **Custom Properties** tabs.
- Go to **Custom Properties** tab by clicking on it

- Then click on the plus (+) icon to add property as shown below:



- Enter property name and provide default input value as shown in the above screen shot.
- Now execute the currency converter service request once. Only then we can get the property information when right click on the editor.
- Enter the following script in the editor

```
def getPropValue = context.testCase.testSuite.getPropertyValue("FromCurrencyValue")
```

- Click on the **Run** icon

This script gets the property value and assign to the variable "getProValue". To print the value of the property, we can write the following script :

```
Log.info (getPropValue);
```

Global Properties:

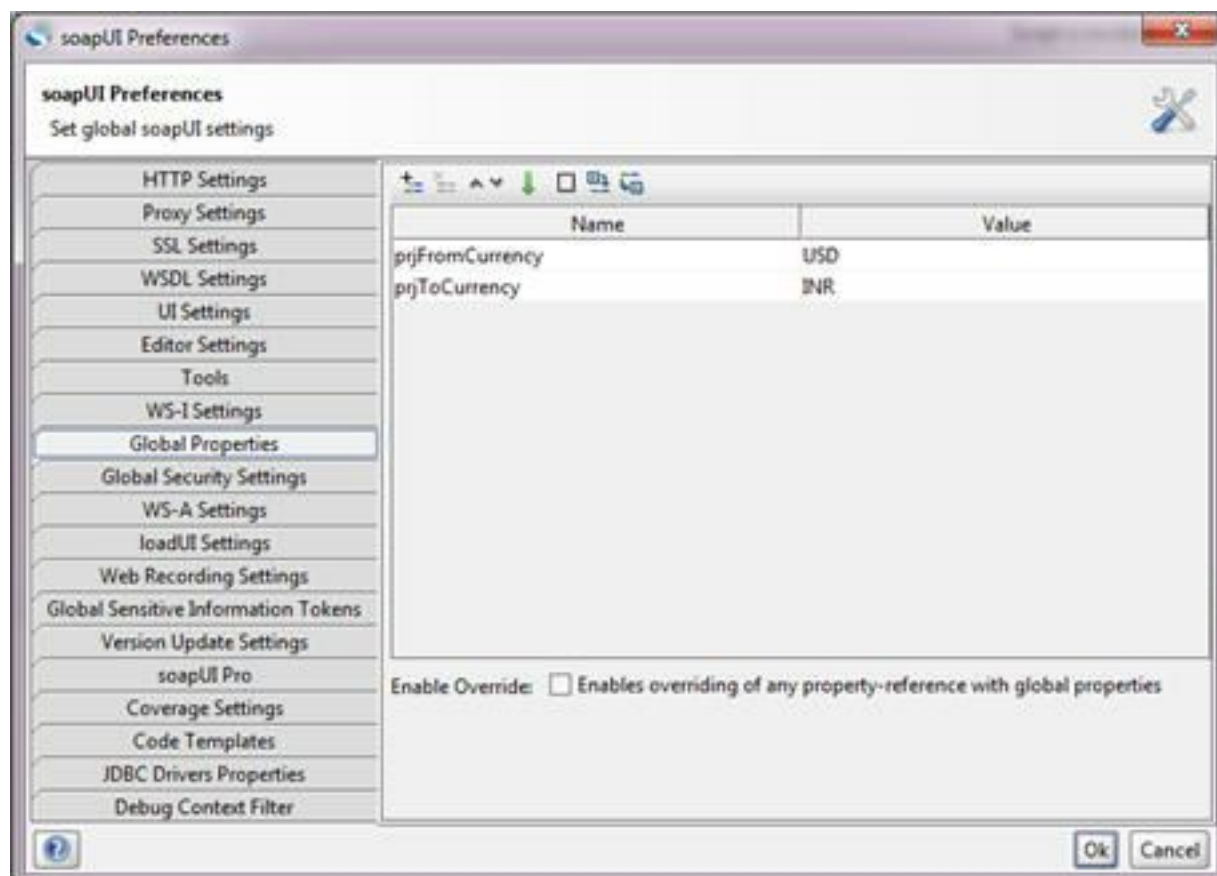
Now let us discuss about global properties. These properties are defined in one place and we can access them across the project components like test suite, test case, test steps etc.

Here are the scripts for writing data to the global properties.

```
1      com.eviware.soapui.SoapUI.globalProperties.setPropertyValue
2      ( "prjFromCurrency",
        "USD" )
3      com.eviware.soapui.SoapUI.globalProperties.setPropertyValue
4      ( "prjToCurrency",
        "INR" )
```

Once we execute the above test step script, the mentioned properties will be created and the respective values will be assigned to those properties. Let us see how we can verify it.

- Click on the **File** menu
- Then, choose **Preferences** option
- In the left side, click on the **Global Properties** tab.
- Verify the properties in the property sheet on the right side. Refer the screenshot below:



8. Working with Properties

This tutorial is all about SoapUI properties. In last SoapUI tutorial we saw how to add properties in Groovy script.

A property in SoapUI is similar to a variable/ parameter and in this tutorial will talk about how to use one in a service request and assign response value to it through scripting. Later, we will move on to property transfer test step and then importing properties.

This is the 8th tutorial in our SoapUI online training series.

What you will learn from this SoapUI Tutorial?

- Different Faces of Properties
- Integrating Properties in Service Request
- Understanding Property Transfer Teststep
- Load Properties Externally

There are two types of properties in SoapUI:

1. **Default Properties:** included in the SoapUI installation. We can edit some of the default properties but not all.
2. **Custom/user-defined properties:** These are defined by us at any level needed, such as global, project, test suite, test case or test step.

*Most often, properties are used to **store and retrieve the data** while executing the test cases. Internally property will store the value in key pair format.*

For example, in the below statement, "Local_Property_FromCurrency" is a key name and "USD" refers value. To access property value, we need to use property name.

testRunner.testCase.testSteps["Properties"].setPropertyValue

(**"Local_Property_FromCurrency"**, **'USD'**)

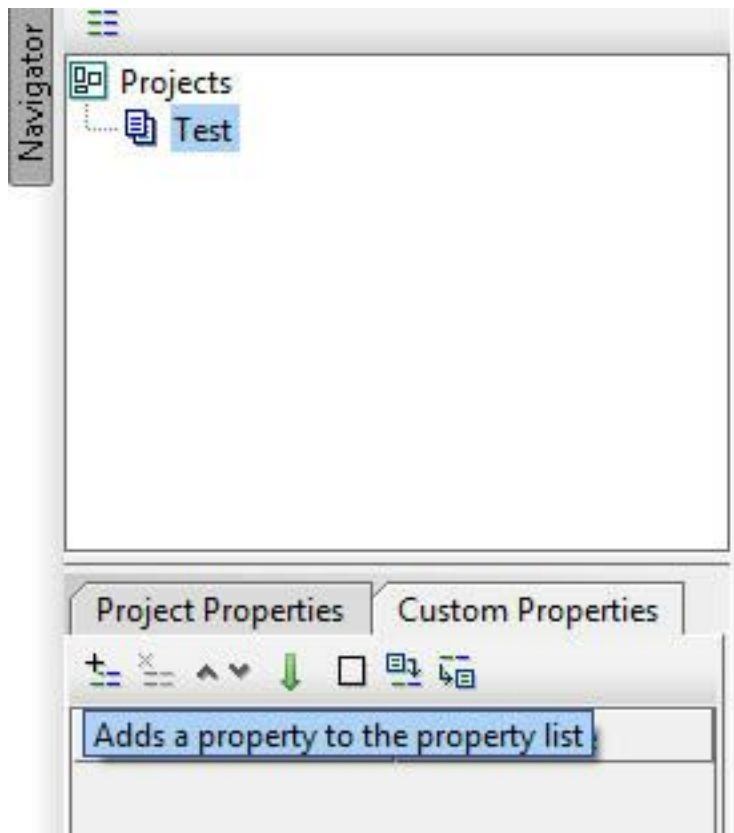
Various property levels in SoapUI Pro

Let us discuss the various property levels in SoapUI Pro. In SoapUI there three levels of properties available.

Level #1. Project and Custom Properties

In this level, properties are divided into two sections. They are project properties and custom properties. These will appear at the bottom of the navigator panel when we click on the Project name. Project properties section has default properties which are created during the project creation for example, Name, Description, File etc.

In order to create our own properties, we can use custom properties tab. Click on the plus icon to create properties:



There are many other options available such as remove, move up, move down and sorting next to add. Any number of custom properties can be added and used by any sections(test suite, test cases) within the project.

Level #2. Test Suite and Custom Properties

These properties are visible only under the test suite. A test suite can contain any number of properties and they can be accessed from any test steps which belong to the said test suite.

*Test suite properties appear when click on the respective test suite name under the project.
To add custom properties as needed, click on custom properties tab and click on the '+' sign under it.*

TestSuite Properties		Custom Properties	
Property		Value	
Name		TestSuite 1	

Property #3. Test Case and Custom Properties

Test case properties are accessible within the test case. They are not accessible by other test case steps or even test suite under the project.

More details about properties with examples

Properties can store end points, login details, header information, and domain etc. even though we have discussed about writing and reading data to/from the properties, we are yet to discuss this topic in details with example.

The above discussed levels of properties are used in scripting to read the data.

#1. Reading properties:

We will look at how we can read properties in groovy script. In order to access different level properties, the following is the syntax:

Project: Syntax: `${# Project Name # Value }`

Example:

```
def projectPro = testRunner.testCase.testSuite.project.getPropertyValue
```

```
( "Project_Level_Property" )
```

```
"Project_Level_Property" )
```

```
log.info (projectPro)
```

Test suite: Syntax: `${# TestSuite # Value }`

Example:

```
def testPro = testRunner.testCase.testSuite.getPropertyValue('Testsuite_Property')
```

```
log.info (testPro)
```

Test case: Syntax: `$ {# TestCase # Value }`

Example:

```
def testcasePro = testRunner.testCase.getPropertyValue('Testcase_Property')
```

```
log.info (testcasePro)
```

Refer the screenshot below:



```
1 def projectPro = testRunner.testCase.testSuite.project.getPropertyValue("Project_Level_Property")
2 log.info (projectPro)
3
4 def testPro = testRunner.testCase.testSuite.getPropertyValue("Testsuite_Property")
5 log.info (testPro)
6
7 def testCasePro = testRunner.testCase.getPropertyValue("Testcase_Property")
8 log.info (testCasePro)
```

Wed Jul 30 13:13:00 IST 2014:INFO:Project Level Property String
Wed Jul 30 13:13:00 IST 2014:INFO:Test Suite Property Testing
Wed Jul 30 13:13:00 IST 2014:INFO:Testcase Property String

#2. Writing to properties:

To do this, we have to use **setPropertyValue** method.

Syntax: **setPropertyValue ("property name","value")**

If we assign values to unknown properties, then SoapUI will create these properties newly. For existing properties will receive the values during the assignment.

#3. Removing Properties through Script:

This can be done by right-clicking on the property name from the property panel. Then click on the Remove option from the context menu.

To do this using script for removing the custom properties use the following statements for project, test suite or test case levels respectively:

testRunner.testCase.testSuite.project.removeProperty("Testcase_Property");

```
testRunner.testCase.testSuite.removeProperty( "Testcase_Property" );
```

```
testRunner.testCase.removeProperty( "Testcase_Property" );
```

The above scripts are not optimum when we have multiple properties in each level as these steps have to be repeated several times for each property. An alternative is to iterate the properties through the script as below:

```
testRunner.testCase.properties.each  
  
{  
  
    key,value ->  
  
    testRunner.testCase.removeProperty(key)  
  
}
```

The above script will iterate till the last property available under the test case. "**Key**" refers the name of the property where as "**value**" denotes actual value of the property. We can modify the above script to remove the bulk property list present in various levels.

#4. Add property:

AddProperty method is used for this whose syntax is:

```
addProperty ( property name );
```

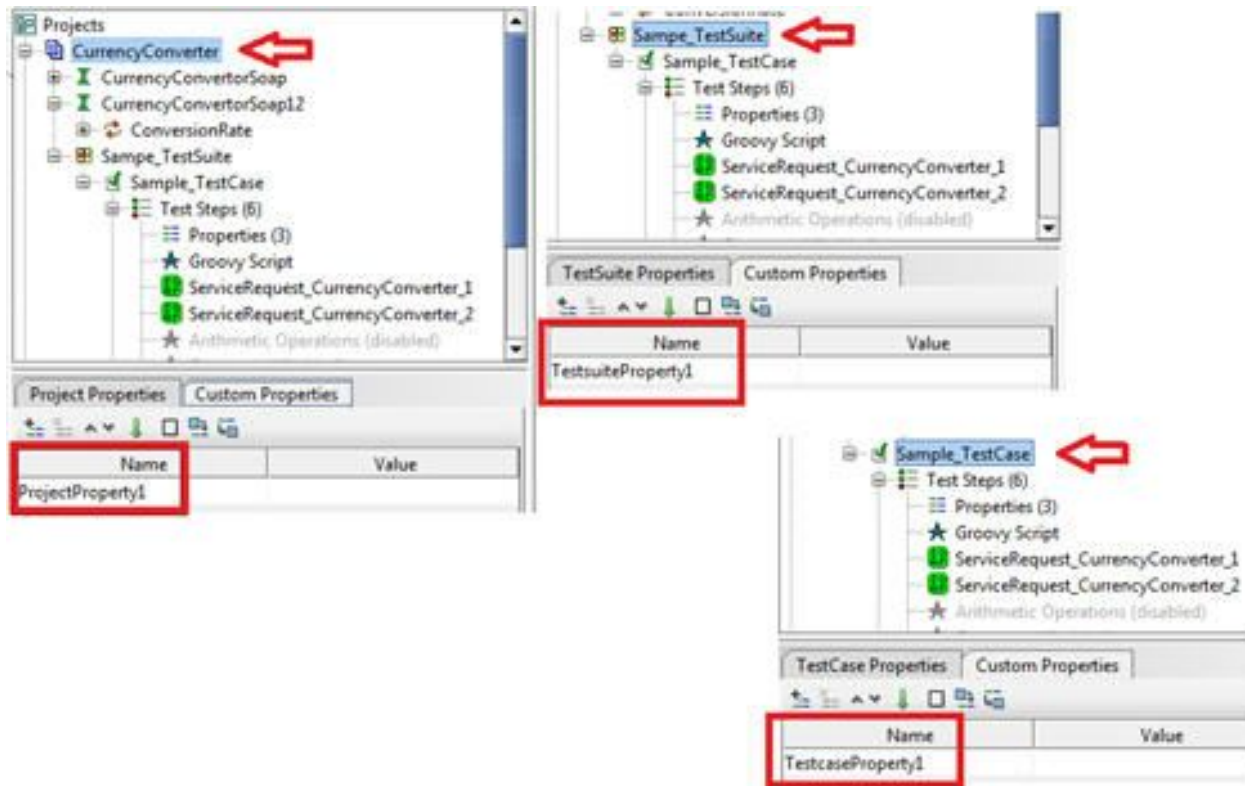
This can be adapted for each level as below:

```
testRunner.testCase.testSuite.project.addProperty('ProjectProperty1')
```

```
testRunner.testCase.testSuite.addProperty('TestsuiteProperty1')
```

```
testRunner.testCase.addProperty('TestcaseProperty1')
```

After executing the above scripts, click on the project/test suite/test case name. Check the custom properties tab in property panel and the created property appears here. See below for reference:

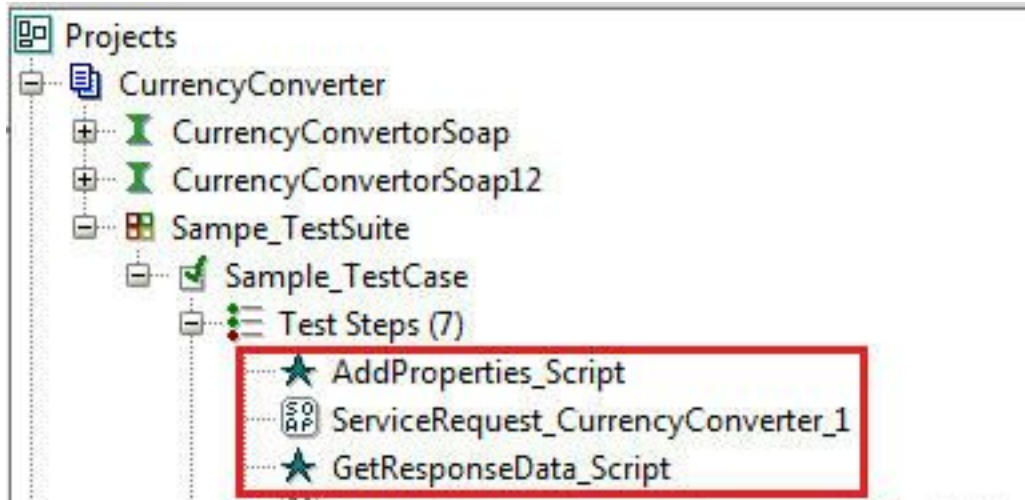


Using properties in services

In this section, we will learn how we can use the properties in services and we are going to use the above scripts for adding, assigning, retrieving property data with currency converter web service.

Integrating Properties in Service:

Let us start adding test steps as shown in the below screenshot.



In the above screenshot, *AddProperties_Script* test step contains the following script which adds two properties such as **Property_FromCurrency** and **Property_ToCurrency**.

// Add Properties

```
testRunner.testCase.addProperty('Property_FromCurrency')
```

```
testRunner.testCase.addProperty('Property_ToCurrency')
```

// Assign values to the Properties

```
testRunner.testCase.setPropertyValue('Property_FromCurrency','USD')
```

```
testRunner.testCase.setPropertyValue('Property_ToCurrency','INR')
```

In the **ServiceRequest_CurrencyConverter_1** contains the request with input parameters as seen below:



Assigned values in the properties will be transferred to these parameters during the execution. Following this test step, **GetResponseData_Script** test step has the script that will get the response value and show the result in the log. Here's the script.

```
// Get Response data from the service
```

```
def response = context.expand( '${ServiceRequest_Currency
```

```
Converter_1#Response}' )
```

```
def parsedResponse = new XmlSlurper().parseText(response)
```

```
String convertedValue = parsedResponse.Body.ConversionRateResponse.
```

```
ConversionRateResult.text()
```

```
log.info(convertedValue)
```

Once all the steps are ready, double click on the test suite name and run the test suite. Then, double click on the **ServiceRequest_CurrencyConverter_1** and see the response section.

This is what we would find:

- Response will be received

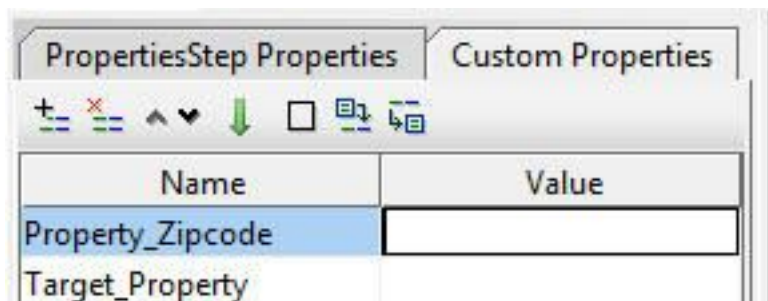
- Open the script log to see the resultant data that is converted based on input parameters

This is how we can pass the parameters to the input request and get the response through the script using properties. Going further we can also pass the response value to another service as input.

Property Transfer

The property transfer test step transfers the property data from one property to another during the execution. Let us see in brief how we can create property transfer test step and how the property value is transferred between two properties.

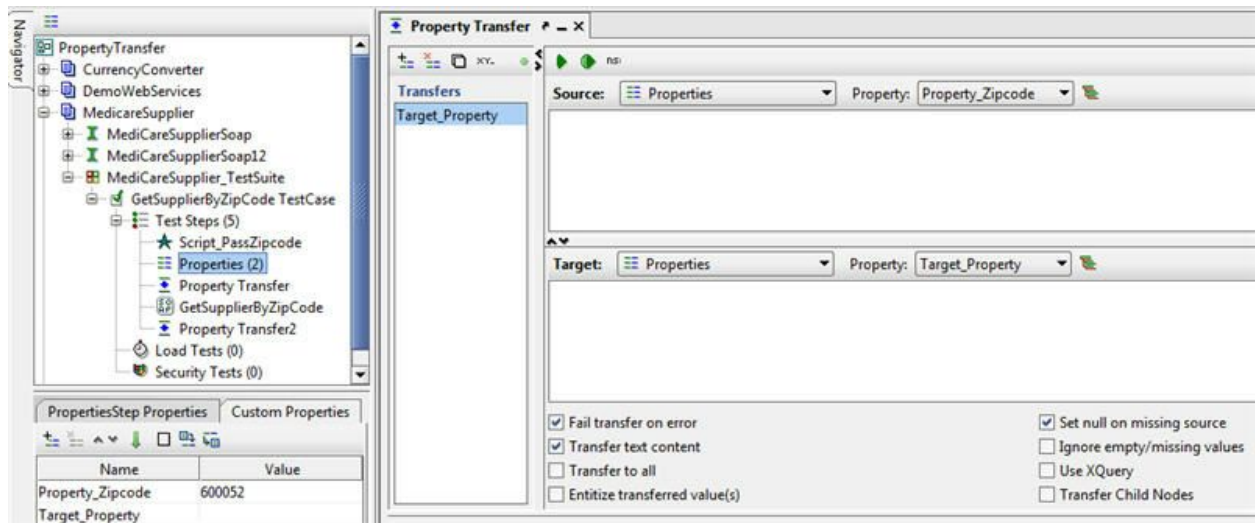
- Right click on the test case name under the test suite
- Click **Add Step** and then click **Properties** option from context menu
- Repeat the above steps to create the second property. See the below screenshot:



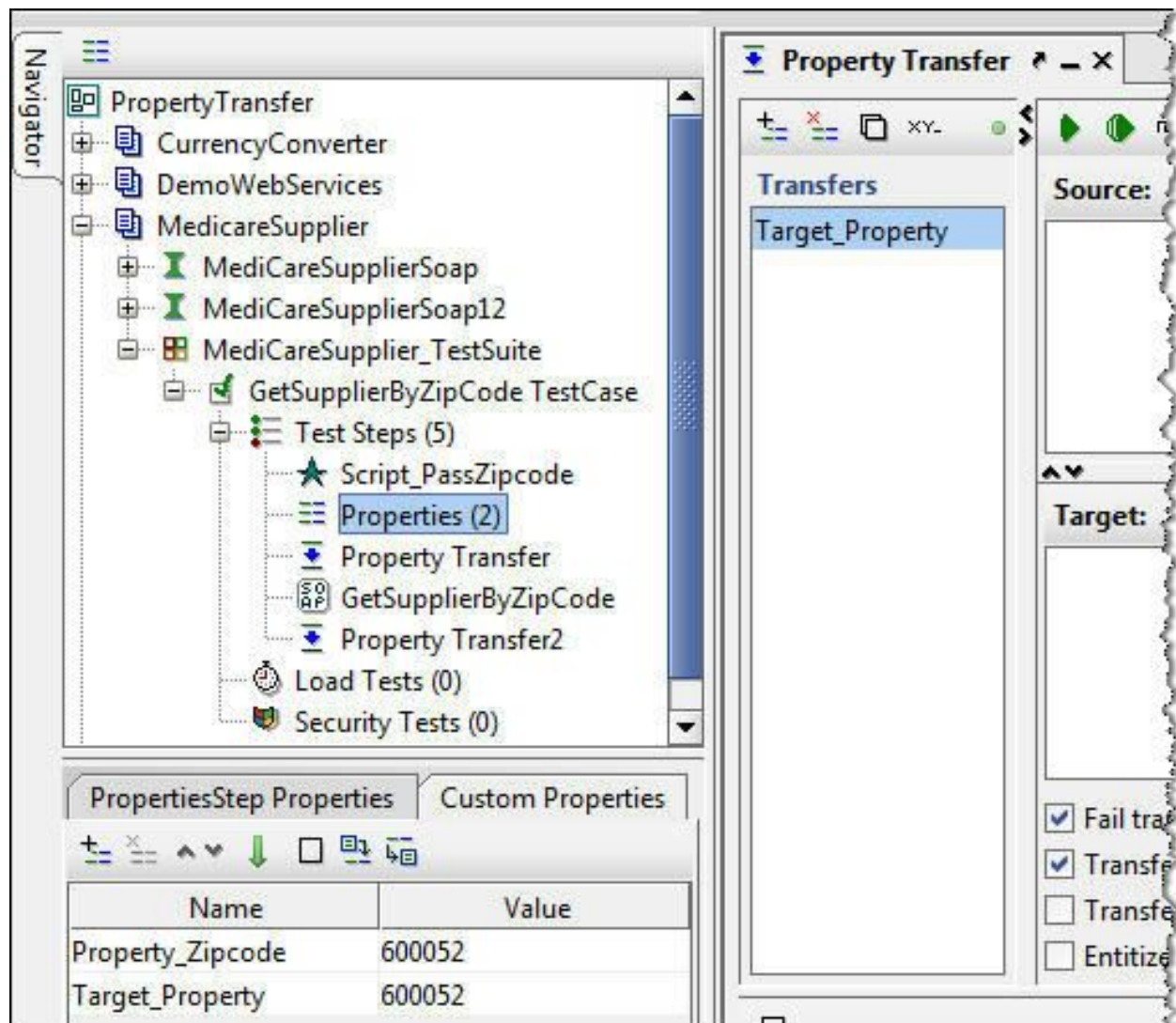
- Now we have to add property transfer test step.
- Right click on the test case name and click property transfer option from the context menu
- Enter your desired property transfer name and then click OK
- Click Add i.e. plus sign in the property transfer tool bar
- Specify the transfer name and then click OK button
- In the right side panel, there are two sections available: Source and Target.

Choose the source as **Properties** and property as **Property_Zipcode**. Do the same in the target section. Choose **Target_Property** from the property drop down. When run icon, the property value will be transferred from **Property_Zipcode** to **Target_Property**.

(Click on image for enlarged view)



See the transferred value as shown in the below screenshot.



Note: Source property should contain the default value.


In addition to this, there are many options available in the property transfer screen.

- Fail Transfer on Error
- Transfer Text Content
- Transfer to All
- Entitize Transferred Value(s)
- Set Null on Missing Source
- Ignore Empty/Missing Values
- Use XQuery

- *Transfer Child Nodes*

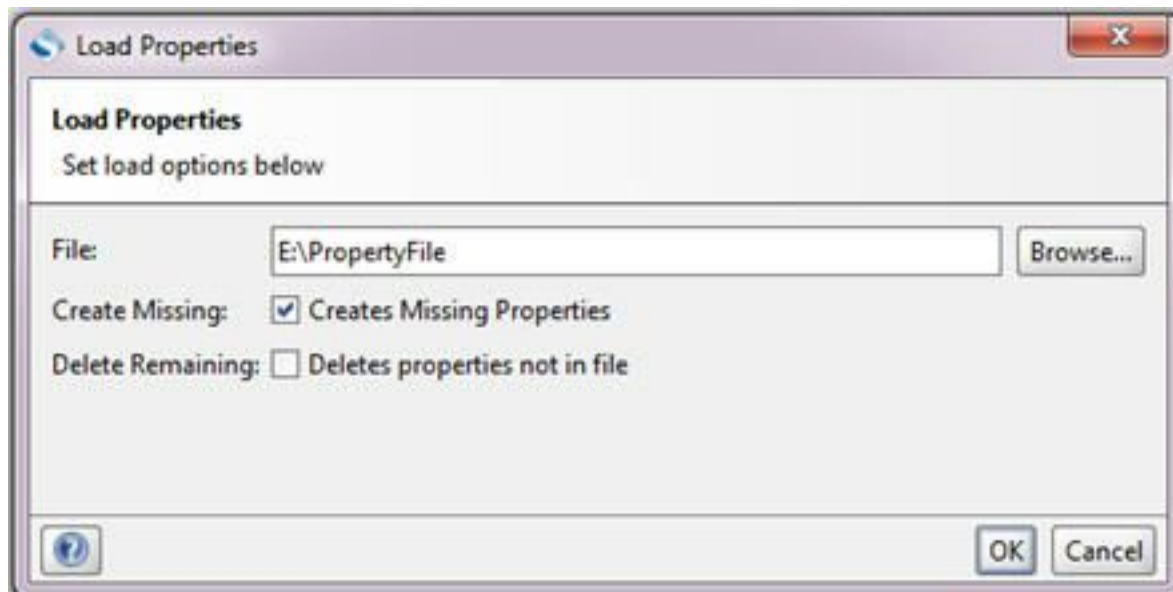
Load Properties from External Source:

To load properties from an external source, follow these steps.

- *Add Properties test step under the test case*
- *Enter the property step name and then click OK*
- *In the property panel under the navigation panel, click Custom Properties tab*
- Click  icon to load the properties from the external property file

Note: *Property file should be saved or present on your computer. To save the properties click icon.*

Then, go to the respective drive and pick the property as shown below:



On OK, we can see the loaded properties and their values in the Custom Properties tab.

9. **Conditional Statements in Groovy**

Conditional statements are used to control the flow of the execution. Here are different types of conditional statements in Groovy.



#1. Control or logical statements:

These statements result in true or false based on the logical conditions. They are AND, OR and NOT. The symbols used are '&& (and)', '||' and '!' (Exclamation)' respectively. The behavior is as mentioned in the truth table:

"And" Operator truth table:

A	B	A && B
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

"OR" operator truth table:

A	B	A B
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

"NOT" operator for negation purpose

Conditional statement: programming languages support the following conditional statements:

- If...else statement
- If...else if...else statement
- Ternary operator
- Switch statement

A) if...else block syntax:

if <condition>

{

<Statements to be executed>

```
}
```

Else

```
{
```

<default statements>

```
}
```

In this syntax, when <condition> meets true the given statement will get executed. We need to enclose the statements by curly braces if the block contains more than one line. This instructs the entire block should execute based on the condition.

Look at the example code snippet.

```
1      int
      a=100

2      int
      b=200

3      if
      (a>b)

4      {

5          log.info('B is greater than
      A');

6          <To
      Do>

7      }

8      else
```

```

9      {
10     log.info('A is greater or both are
      equal');
11     <To Do>
12  }
```

An **if...else...** statement should contain only one ELSE block.

B) Multiple conditional blocks: if...else if...else syntax

if <condition1>

```
{
```

<Statements to be executed>

```
}
```

Else if <condition2....n>

```
{
```

<Statements to be executed>

```
}
```

Else

```
{
```

<default statements>

}

If...else if...else statement is little different than **if...else...** statement- in terms that it has **else if** block. This block is used for adding multiple conditions. Look at the following example.

```
1      int
      a=100

2      int
      b=200

3      int
      c=300

4      if (a>b &&
      a>c)

5      {

6          log.info('A is greater than B
      and C');

7      }

8      else if (b>a &&
      b>c)

9      {

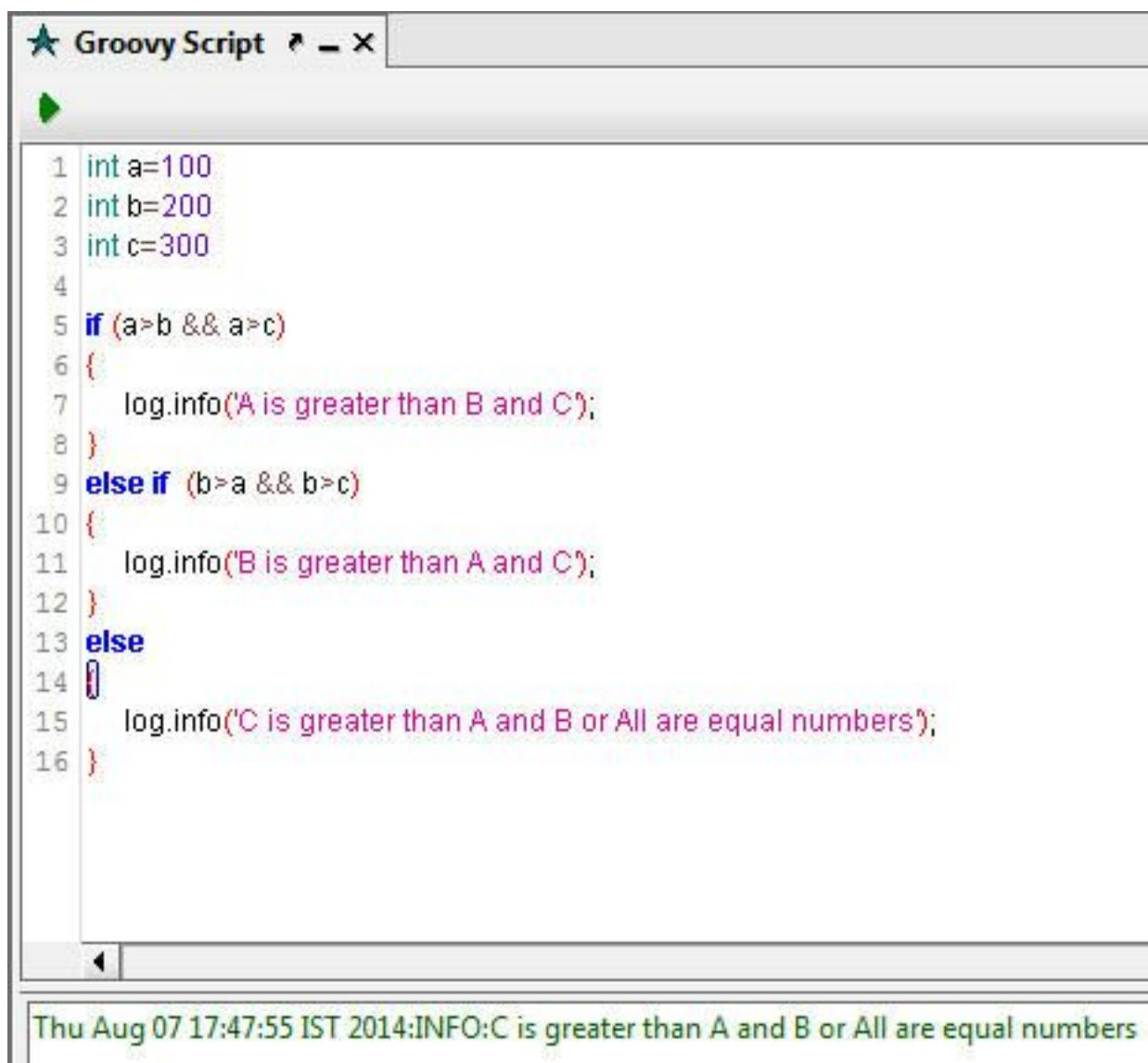
10         log.info('B is greater than A
      and C');

11     }

12     else
```

```
13 {  
14     log.info('C is greater than A and B  
    or All are equal numbers');  
15 }
```

And look at the output screenshot for the above script: Also, please use this as a reference for the boolean operator example too:



The screenshot shows a 'Groovy Script' editor window. The script contains the following code:

```
1 int a=100  
2 int b=200  
3 int c=300  
4  
5 if (a>b && a>c)  
6 {  
7     log.info('A is greater than B and C');  
8 }  
9 else if (b>a && b>c)  
10 {  
11     log.info('B is greater than A and C');  
12 }  
13 else  
14 {  
15     log.info('C is greater than A and B or All are equal numbers');  
16 }
```

Below the script editor, the output is displayed in a green font:

```
Thu Aug 07 17:47:55 IST 2014:INFO:C is greater than A and B or All are equal numbers
```


C) Ternary operator:

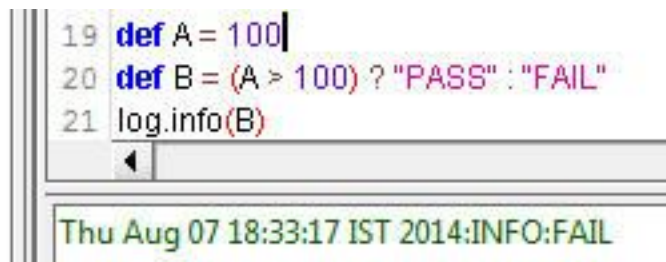
This operator works similar to **if...else** statement. It has two operands such as question mark (TRUE) and colon for FALSE / default statements.

```
def A = 100
```

```
def B = (A > 100) ? "PASS" : "FAIL"
```

```
log.info(B)
```

Here's the screenshot for the above script.



```
19 def A = 100
20 def B = (A > 100) ? "PASS" : "FAIL"
21 log.info(B)
```

Thu Aug 07 18:33:17 IST 2014:INFO:FAIL

D) Switch case: This statement allows multiple conditional branches with different values. Also it supports comparison of class object references, regular expressions, collections such as arrays, lists etc.

Switch <value>

{

case <match1>:

 <execute block>;

 break;

case <match1...match (n)>:

`<execute block>;`

`break;`

`default:`

`<execute default block>;`

`}`

The **case** statements compare the values with the **switch** case value. If it's a match, then the corresponding case block gets executed. Once the block is executed then it should be stopped by the "break" keyword. If we missed "break" keyword at end of the case statement, execution will be moved to next case statement- and that might not be necessary. In case none of the cases are true, **default** block will be executed. Please note that Groovy supports all the major operators and keywords as it is incorporated with the java libraries.

```
1      def
      country="India"

2      switch(country
      )

3      {

4          case
          "Japan":

5              log.info('Country matched with 1st case
          statement');

6          break;

7          case
```

```
    "Australia":  
8      log.info('Country matched with 2st case  
statement');  
9  
    break;  
10     case  
    "India":  
11         log.info('Country matched with 3st case  
statement');  
12  
    break;  
13  
    default:  
14         log.info('None of the matches available  
:(');  
15 }  

```

Here's the output screenshot for the above script.

```

24 def country="India"
25 switch(country)
26 {
27     case "Japan":
28         log.info('Country matched with 1st case statement');
29         break;
30     case "Australia":
31         log.info('Country matched with 2st case statement');
32         break;
33     case "India":
34         log.info('Country matched with 3st case statement');
35         break;
36     default:
37         log.info('None of the matches available :()');
38 }

```

Thu Aug 07 18:56:48 IST 2014:INFO:Country matched with 3st case statement

#2. Looping or Iterative Statements:

These enable repetition whenever we need and are especially useful for data driven testing.

*For instance, let us assume we have a few zip codes in an excel file. To retrieve all the zip codes one by one from the excel file, and pass it to service i.e. **GetSuppliersZipcode**, we can use iterative statements. SoapUI also provides an alternative feature called data source and data source loop test steps.(Available only in SoapUI Pro trial and licensed versions.)*

We can use these following iterative statements in the groovy script:

- For loop
- While loop

For loop:

for (<initialization>; <condition>; <increment /decrement>)

```
{
```

```
    <Execute Statements>;
```

```
}
```

In the above syntax, initialization denotes starting point of the loop and based on the condition loop will continue or stop the execution.

See the below script

```
for(int i=1; i<=10; i++)
```

```
{
```

```
    log.info('Loop Iterated ' + i + ' times');
```

```
}
```

Above script will produce the results as shown in the following screenshot.

```
41 for(int i=1; i<=10; i++)
```

```
42 {
```

```
43     log.info("Loop Iterated " + i + "times");
```

```
44 }
```

```
45
```

```
Thu Aug 07 20:33:04 IST 2014:INFO:Loop Iterated 1 times
```

```
Thu Aug 07 20:33:04 IST 2014:INFO:Loop Iterated 2 times
```

```
Thu Aug 07 20:33:04 IST 2014:INFO:Loop Iterated 3 times
```

```
Thu Aug 07 20:33:04 IST 2014:INFO:Loop Iterated 4 times
```

```
Thu Aug 07 20:33:04 IST 2014:INFO:Loop Iterated 5 times
```

```
Thu Aug 07 20:33:04 IST 2014:INFO:Loop Iterated 6 times
```

```
Thu Aug 07 20:33:04 IST 2014:INFO:Loop Iterated 7 times
```

```
Thu Aug 07 20:33:04 IST 2014:INFO:Loop Iterated 8 times
```

```
Thu Aug 07 20:33:04 IST 2014:INFO:Loop Iterated 9 times
```

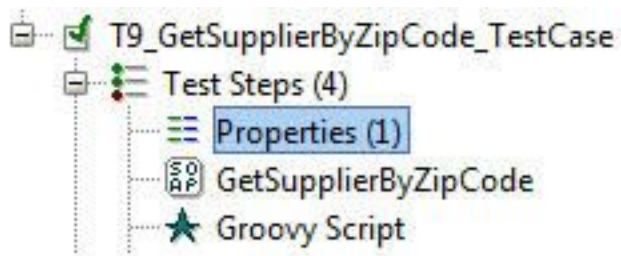
```
Thu Aug 07 20:33:04 IST 2014:INFO:Loop Iterated 10 times
```

Now let us use control statement and iterative statement in our real testing world. Follow the below steps:

- *Add Testcase with your desired name. I have created as "T9_GetSupplierByZipCode_TestCase".*
- *Then add a property named "Property_Zipcode"*
- *Add Test Request step for "GetSupplierByZipCode" service and put the property name in the request as shown in the screenshot.*



Add groovy script test step as shown in the following screenshot.



In the groovy script editor, write the following script.

```
1      for (int zipCode=1; zipCode<10;
2          zipCode++)
3      {
4          testRunner.testCase.testSteps['Properties']
5              .
6              setPropertyValue('Property_Zipcode', '3000' +
7                  zipCode )
8
9          def testStep =
10             testRunner.testCase.testSteps['GetSupplierByZipCode']
```

```
    | ];  
6  |  
7  |     if  
    |     (zipCode>5)  
8  |     {  
9  |         log.info('*****Stopped  
    | Execution*****');  
10 |  
    | break;  
11 |  
    | }  
12 |  
    | testStep.run(testRunner,context);  
13 |  
    | log.info('Executed ' + zipCode + '  
    | times')  
14 | }  
    |
```

The following result is received in the log as well as in the service response screen.

(Click image for enlarged view)


```

Fri Aug 08 11:17:36 IST 2014:INFO:Executed 1 times
Fri Aug 08 11:17:37 IST 2014:INFO:Executed 2 times
Fri Aug 08 11:17:38 IST 2014:INFO:Executed 3 times
Fri Aug 08 11:17:39 IST 2014:INFO:Executed 4 times
Fri Aug 08 11:17:40 IST 2014:INFO:Executed 5 times
Fri Aug 08 11:17:40 IST 2014:INFO:*****Stopped Execution*****

```

```

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:tns="http://www.webservices.com"
  xsi:schemaLocation="http://www.w3.org/2003/05/soap-envelope
    http://www.w3.org/2003/05/soap-envelope.xsd"
  >
  <soap:Body>
    <GetSupplierByZipCodeResponse xmlns="http://www.webservices.com">
      <GetSupplierByZipCodeResult>true</GetSupplierByZipCodeResult>
      <SupplierDataLists>
        <SupplierData>
          <SupplierNumber>0250370001</SupplierNumber>
          <CompanyName>AMERICARE HEALTH SERVICES CORP</CompanyName>
          <Address1>6025 SHILOH RD</Address1>
          <Address2></Address2>
          <City>ALPHARETTA</City>
          <Zip>GA</Zip>
          <ZipPlus4>30005</ZipPlus4>
          <Telephone>1706</Telephone>
          <Description>(770)886-2604</Description>
          <IsSupplierParticipating>01</IsSupplierParticipating>
        </SupplierData>
      </SupplierDataLists>
    </GetSupplierByZipCodeResponse>
  </soap:Body>
</soap:Envelope>

```

"while" loop:

Syntax:

while <condition>

{

<Execute Statements>;

}

The above logic can be implemented using the "while" loop too.

Hence, the gist is that iterative statement can be used to:

1. Execute the test cases or test steps repeatedly
2. Control the flow of the execution through the control statements.

#3. Arrays Collection:

Array collection helps store multiple values in a single variable or object. Array index starts at zero by default and we need to use that index with array name to access the corresponding value stored in the array. Syntax to declare arrays in groovy script:

`arrayName = new Object[5] or,`

`arrayName = new Object[10][] or,`

`arrayName = new Object[10][10][]`

Note: While declaring arrays we must specify the initial size otherwise it will throw a compile error. Here's the simple example for the single dimensional array.

`ArrayObj = new Object [5];`

`ArrayObj[0] = "Testcase1";`

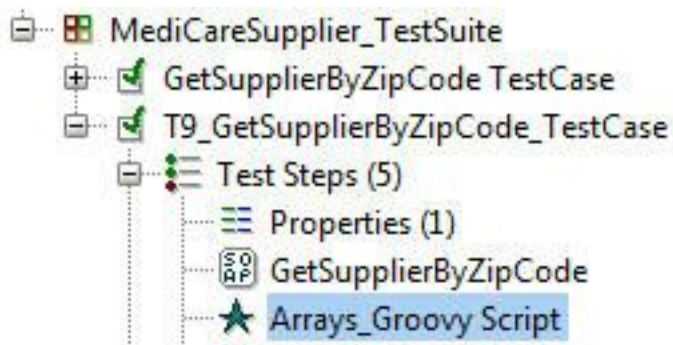
`ArrayObj[1] = "Testcase2";`

`ArrayObj[2] = "Testcase3";`

`ArrayObj[3] = "Testcase4";`

`ArrayObj[4] = "Testcase5";`

Now let us implement this in our regular test steps. So add property, test request and script test step under the project test suite as shown in the following screenshot.



And then double click on the script test step and write the following script.

```
1      def MAX_LIMIT =  
      5  
  
2      def zipCodes = new  
      Object[MAX_LIMIT]  
  
3  
  
4      zipCodes[0] =  
      "92704"  
  
5      zipCodes[1] =  
      "99362"  
  
6      zipCodes[2] =  
      "31401"  
  
7      zipCodes[3] =  
      "90247"  
  
8      zipCodes[4] =  
      "87102"  
  
9  
  
10     int  
      i=0;  
  
11     while  
      (i<5)  
  
12     {  
  
13         if  
      (i<5)  
  
14     {
```

```

15 | testRunner.testCase.testSteps['Properties'].
16 | setPropertyValue('Property_Zipcode',zipCodes
    | [i]);
17 |
    |         def testStep =
testRunner.testCase.testSteps['GetSupplierByZipCode'
    | ];
18 |
    | testStep.run(testRunner,context);
19 |
    |         log.info('Loop executed ' + i + '
    | times');
20 |     }
21 |
    |     i++;
22 | }
23 |
    | log.info("Testcase execution is
    | completed....");

```

*In this script, we initialized array object as 5 and assigned five zip codes in each array location respectively. Next part of the script is iterative block. Here we iterate the loop up to 5 times. Each time array value will be assigned to the property and then move to **GetSupplierByZipCode** web service to pass this array value in the request. After that, service step will be executed with the zip code.*

Finally we will get the following message in the log as shown in the screenshot.

```
Fri Aug 08 13:02:49 IST 2014:INFO:Loop executed 0 times  
Fri Aug 08 13:02:50 IST 2014:INFO:Loop executed 1 times  
Fri Aug 08 13:02:51 IST 2014:INFO:Loop executed 2 times  
Fri Aug 08 13:02:52 IST 2014:INFO:Loop executed 3 times  
Fri Aug 08 13:02:53 IST 2014:INFO:Loop executed 4 times  
Fri Aug 08 13:02:53 IST 2014:INFO:Testcase execution is completed....
```

So arrays are very useful to handle multiple elements with different types. More practice will foster better understanding and ease of use.

10. **Object Oriented Concepts**

Object-Oriented Groovy Scripting in SoapUI:

In last SoapUI tutorial we learned conditional statements in Groovy scripts. Now this tutorial is all about- object oriented programming in SoapUI Groovy scripts- a very interesting and important programming concept. It includes classes and objects, inheritance, encapsulation and polymorphism.

This tutorial #10 in our comprehensive SoapUI tutorial series.

All the major programming languages such as C++, Java, C#, Visual Basic 6.0 etc., support object oriented concepts. All the parts of computer devices have object oriented concepts implemented internally.

So let's get started.

What is object oriented programming?

Object oriented programming is a programming language specification that deals with "objects". Using this concept, complex desktop/web/mobile and business applications are often developed. A brief introduction to the components of object oriented programming concept is below:

- **Classes and Objects**
 - *Consists of methods and properties. Methods are set of instructions that perform specific action. It helps us to reuse the code that is already defined. Properties are the variables to store the values temporarily.*
- **Inheritance**
 - *This is very useful component in OOP. It helps us create a new class from an existing class so the properties and methods can be reused or accessed in the derived class directly. Technically, the main class is called as Base / super class and the inherited class is known as derived class.*
- **Encapsulation**

- Using this concept, we can hide the class information or protect properties of the class by accessing from outside.
- **Polymorphism**
 - **Poly** refers "many" and **morphism** means "forms" that means an object can be defined in many forms.

What are objects?

An object is a container that consists of properties and methods that perform certain action(s).

Objects are considered as real world materials like smart phone, tablet, monitor etc. it has characteristics and activities.

Let's take human being as an example for object. As a person, we have different characters and behavior among us like face, skin color, living style etc. we are doing regular activities such breathing, speaking, walking and so forth. So an object can include characteristics and actions implicitly.

In the object oriented programming paradigm, an object is an instance of a class which consists of properties and methods also it is represented as blue print of the class. A class is generally defined as following format.

```
class <class name>

{

    [access modifiers:]

    <properties declaration>

    <methods>

}
```

As we discussed in our previous tutorials, groovy script supports java code directly without having to import built-in libraries explicitly. Groovy script editor does not allow defining

user-defined classes directly but defining methods in groovy is straightforward. Once done these methods can be invoked anywhere in the script. The methods in groovy scripts can be defined as below:

```
<return type> <method name> [ parameters ]
```

```
{
```

```
    <statements>
```

```
[return]
```

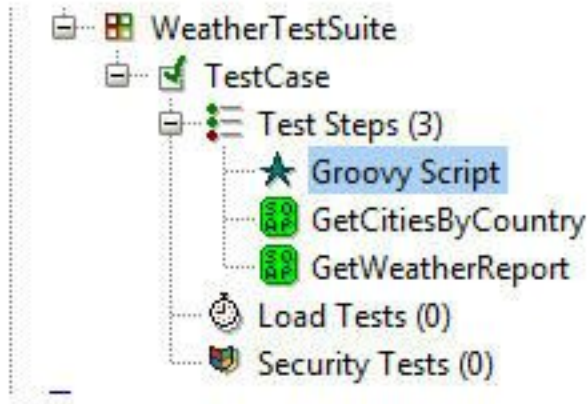
```
}
```

Example:

In the following example, we are going to define a method with some input parameters. Later the parameter values will be assigned to the global properties. Then, we will pass the property values to the service request as an input.

Here are steps:

- *Create a project with <http://www.webservices.net/globalweather.asmx?WSDL> URL*
- *And then add test suite and test cases as shown in the below screenshot*



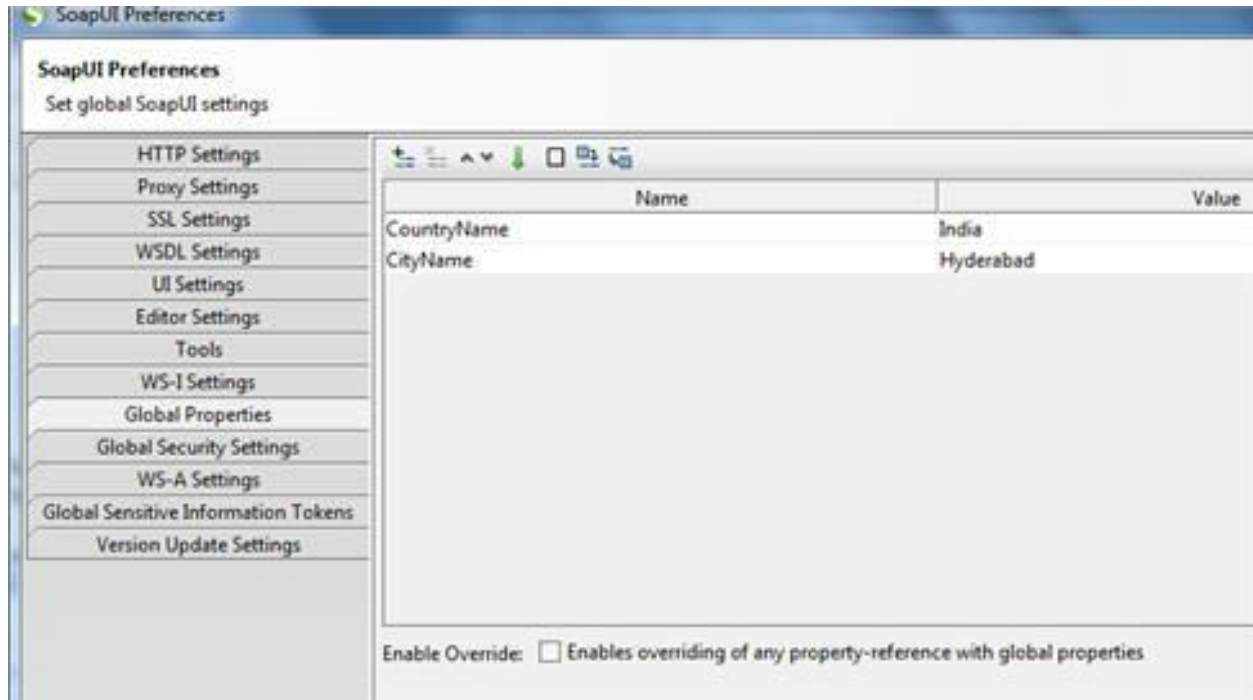
Double click on the groovy script test step and Copy and paste the following script.

```
1      def countryName =
2          "India"
3
4      def cityName =
5          "Hyderabad"
6
7      void CreateAndPassProperties(String countryName,
8          String cityName)
9
10     {
11
12         // Add Global
13         Properties
14
15         com.eviware.soapui.SoapUI.globalProperties.addProperty("CountryName",
16             countryName)
17
18         com.eviware.soapui.SoapUI.globalProperties.addProperty("CityName",
19             cityName)
```

```
9
10 // Assign values to the global
   properties
11
   com.eviware.soapui.SoopUI.globalProperties
   .
12 setValue( "CountryName",
   countryName )
13
   com.eviware.soapui.SoopUI.globalProperties
   .
14 setValue( "CityName",
   cityName )
15 }
16
17 // Method
   Invoking
18 CreateAndPassProperties(countryName,
   cityName)
19
   log.info("Testcase execution is completed
   successfully.")
```

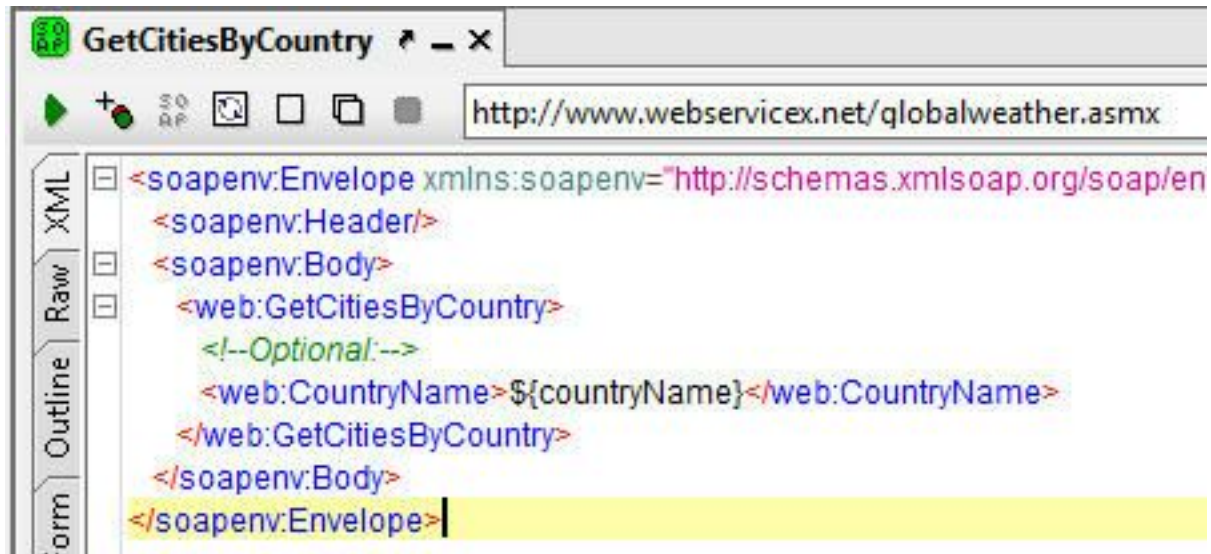
In the above script, we are defining local variables and assigning string data to each variable. Following that, we define a method called **CreateAndPassProperties** with two parameters –**countryName** and **cityName**.

Within the method, the first two lines are used to define the global properties. These can be seen by using the "File->Preferences->Global Properties" as below. **Initially before the test gets executed this will be empty.**

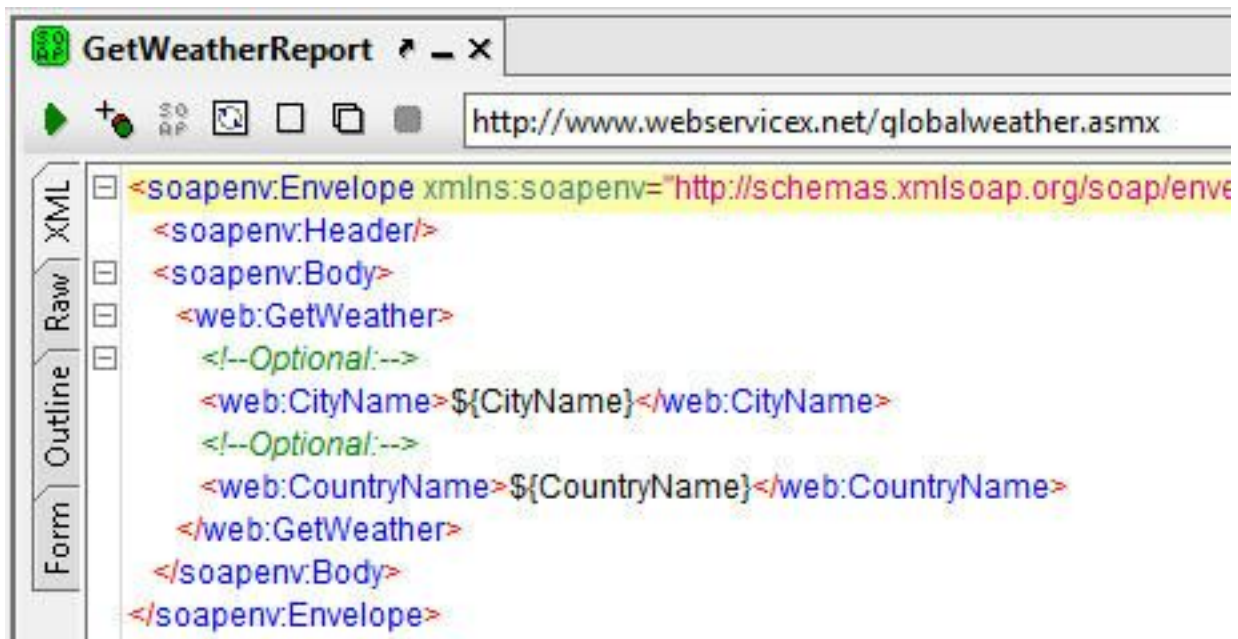


Once done go to the request test steps like **GetCitiesByCountry** and **GetWeatherReport** and make necessary changes in the input request as shown in the below screen shots.

GetCitiesByCountry Web Service:

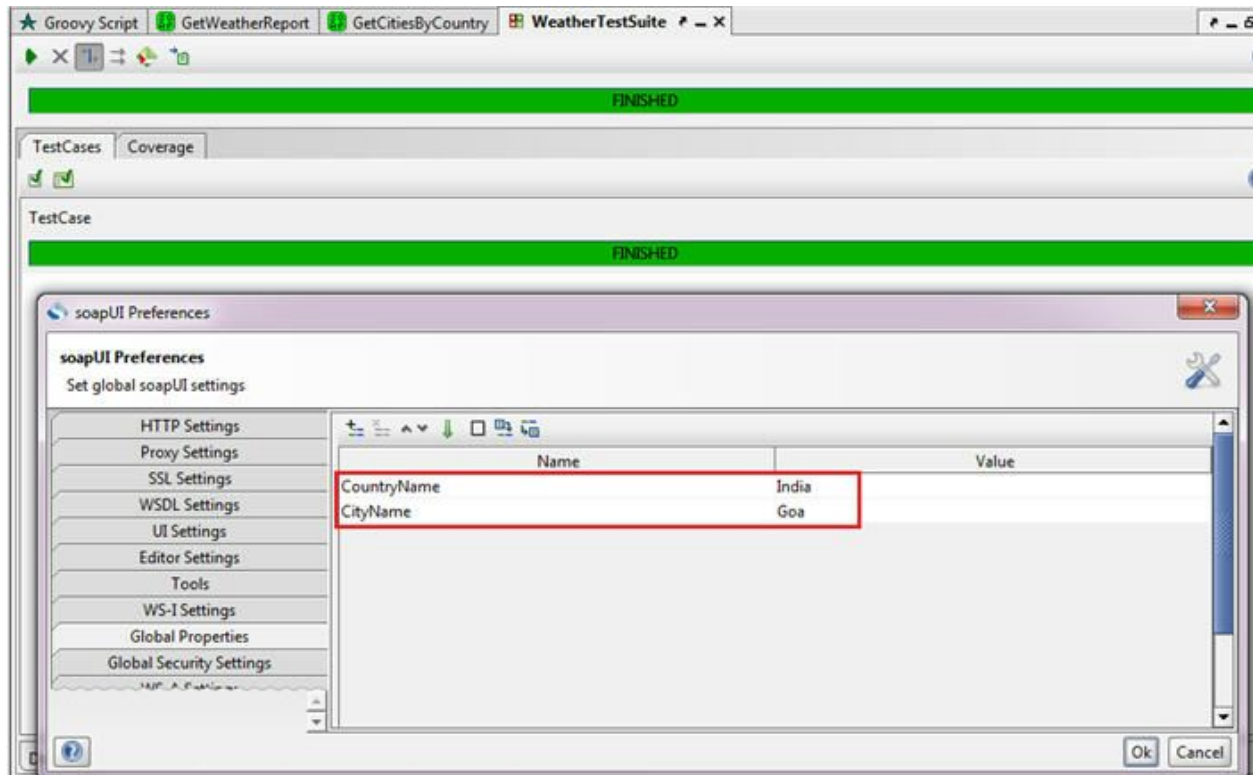


GetWeatherReport Webservice:



Now we can execute the test suite and verify the results with the following screenshot.

(Click image for enlarged view)

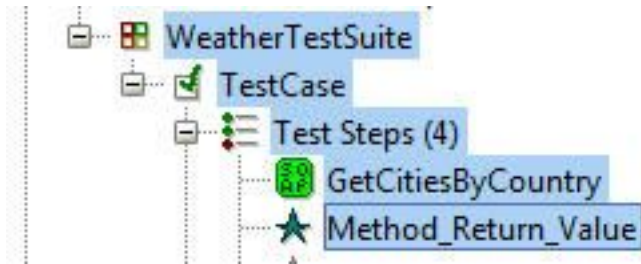


This way we can define multiple methods for various purposes in groovy script test step which can help us execute bulk test steps at once. Methods also allow us to pass arrays to handle bulk data.

Methods and Arrays:

We will now to assign array values to a global property and then pass it to the service. During test suite execution, the script will pass input data to the service and the service will then process the data. Finally it will send the respective response in SoapUI.

Let us create a test suite and test steps as shown in this screenshot.



- Double click on the **GetCitiesByCountry** service and make the change in the request as shown in the following screenshot.



- Double click on the **Method_Return_Value** test step and write the following script:

```
1  def MAX_LIMIT =  
   5  
  
2  def countries = new  
   Object[MAX_LIMIT]  
  
3  
  
4  countries[0] =  
   "India"  
  
5  countries[1] =  
   "US"
```

```
6      countries[2] =  
      "Mauritius"  
  
7      countries[3] =  
      "Cyprus"  
  
8      countries[4] =  
      "Austria"  
  
9  
  
10     // Invoke  
      Method  
  
11     GetCountries (countries  
      );  
  
12  
  
13     // Method  
      Definition  
  
14     void GetCountries (Object[]  
      countries)  
  
15     {  
  
16         for (int i=0; i<5;  
            i++)  
  
17             {  
  
18                 // Assign values to the global properties and  
                call the service  
  
19                 com.eviware.soapui.SoapUI.globalProperties.setProperty
```

```

Value
20     ("CountryName",
    countries[i] )
21
22     // Call GetCitiesByCountry service to
    run
23
    def testStep =
testRunner.testCase.testSteps['GetCitiesByCountry'];
24
testStep.run(testRunner,context);
25
    log.info('Method is called ' + i +
'time(s) ');
26
    }
27
    }

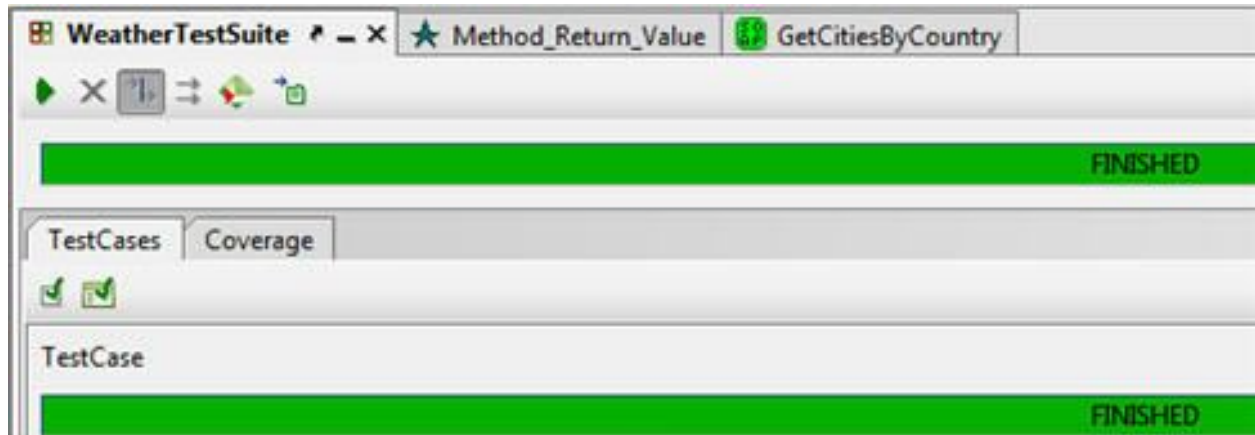
```

In the above script, we have defined an array called **countries** which holds 5 country names.

Following that, we are invoking **GetCountries** method by passing **countries** array. Now the execution flow moves to the method definition where we iterate the loop five times. Each time array element that is country name will be assigned to the global property (i.e. CountryName). Then, **GetCitiesByCountry** test step gets called.

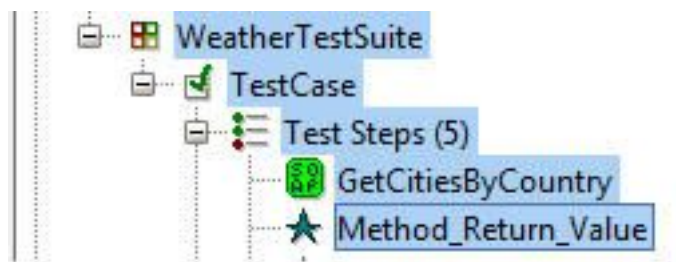
Since we already mentioned global property name in the service request, actual country name will be taken as input and processed accordingly.

Once this test suite is executed successfully, verify the response to determine the service execution status. Let us have a look at the following screenshot.



Now you know the purpose of the methods there is another feature that we need to discuss in this topic and that is Methods can be return values to the caller through output values.

To do so, we employ the "**return**" keyword. Let us take **GetCitiesByCountry** service for this example. Add one more groovy script test step under **WeatherTestSuite** as shown in the screenshot.



Double click on the service name and make the changes if needed. Then write the following script in the groovy script editor.

```

1      // Method
      definition

2      String ReturnCountryName (String
      CName)

3      {

```

```
4      CName =  
      'Mauritius';  
  
5      return  
      CName;  
  
6  }  
  
7  
  
8      // Invoke  
      method  
  
9      String getMethodValue =  
      ReturnCountryName ('US');  
  
10  
  
11      // Assign value to the global  
      property  
  
12      com.eviware.soapui.SoapUI.globalProperties.setPropertyValue  
        
13      ( "CountryName",  
      getMethodValue )  
  
14  
  
15      // Call GetCitiesByCountry service to  
      run  
  
16      def testStep =  
      testRunner.testCase.testSteps['GetCitiesByCountry'];  
  
17      testStep.run(testRunner, context);
```

```

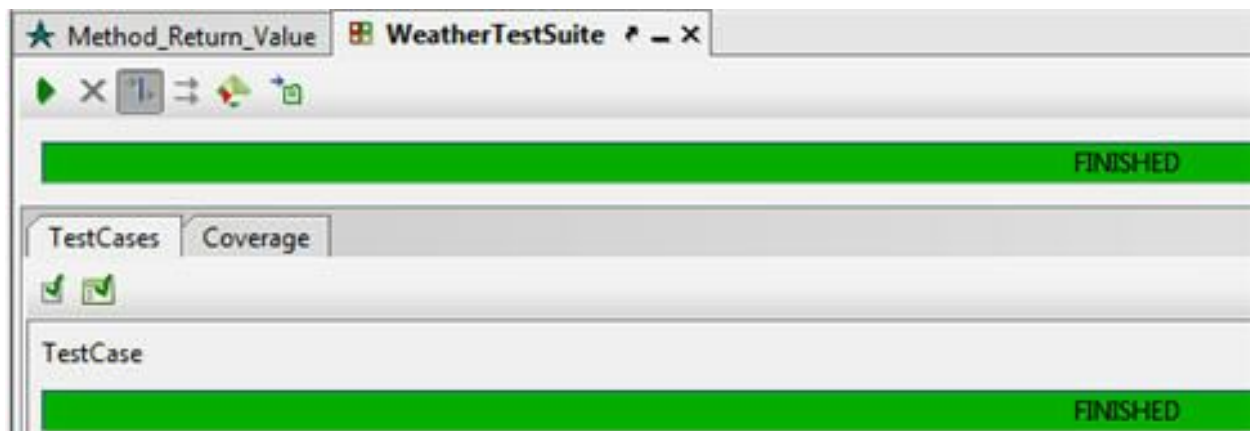
18
19 //
    Message
20 log.info('Testsuite executed
    successfully');

```

The above is a slight modification of the previous script. In the method definition, we are passing a country name as string type and then we assign new country name to the method variable. So the "US" string will be changed to "Mauritius" after the execution.

This will be sent to the **getMethodValue** variable. Remaining code is same as we discussed earlier. It will assign property value to the global property and then service test step will be called by **testStep.run(...)** method.

To execute the test suite, double click on the test suite name and then click Run icon. The result would be as below:



Conclusion

Here is a quick recap of all the concepts we discussed so far:

- Object oriented programming concept plays a main role in all the major programming languages.

- *Objects are the main components that contain properties and methods.*
- *Objects are the blueprint of a class.*
- *Without defining class, it is not possible to instantiate an object.*
- *Methods are one of the parts in the class. These are used to avoid repeated code.*

A few samples have been provided in this article to execute test cases. Implement these samples to your real time web services and test suites to make you more comfortable and to become an expert in the script writing.

11. Exception Handling in Groovy

In this SoapUI tutorial we will look at exception handling using Groovy scripting. Handling runtime exceptions in groovy is similar to Java as the Java libraries are integrated. However we will discuss basic concepts in SoapUI and go into the depth of exception handling in Java.

This is tutorial #11 in SoapUI tutorials series. This is the last tutorial for free version of SoapUI. There are couple of more topics remaining in this series which are on SoapUI pro features, REST and SOAP services, and data driven testing in SoapUI.

Let's begin with a brief introduction to exception:



How to Handle Exception in SoapUI Groovy Scripts

SoapUI Tutorial #11

© www.SoftwareTestingHelp.com

What is an exception?

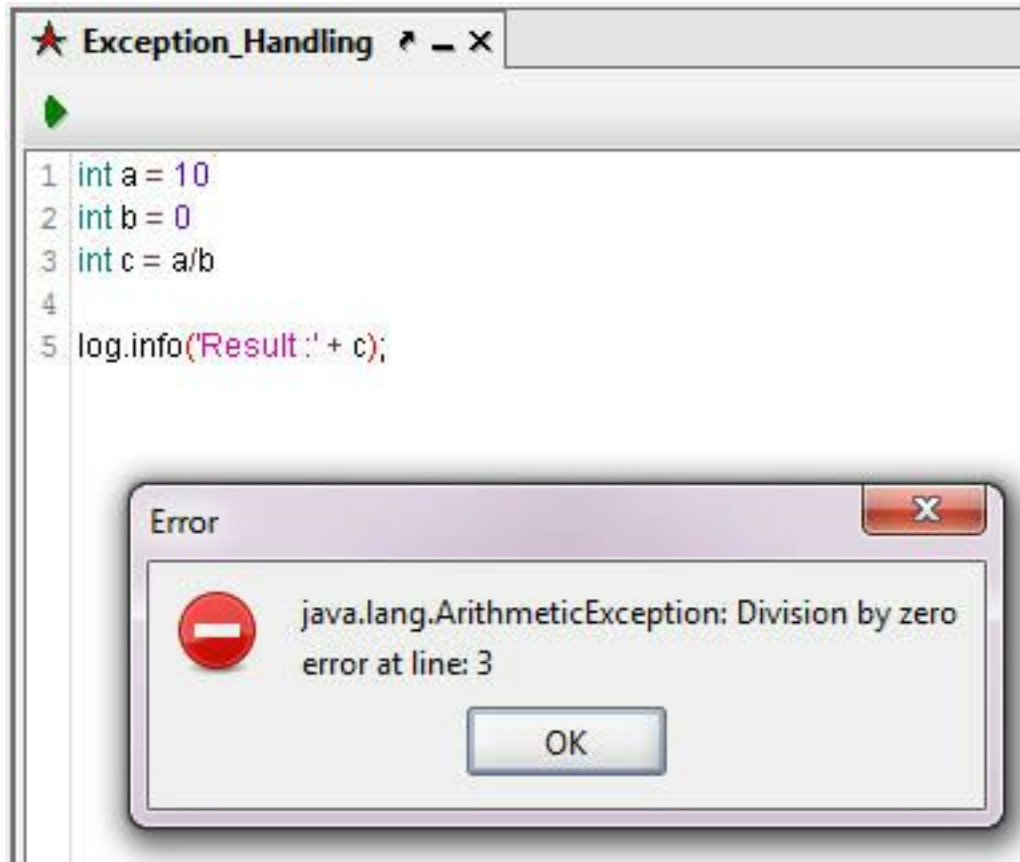
An exception is an error that is encountered during the execution of a program. It can happen due to many reasons such as invalid data, network connection loss, trying open files that are unavailable, accessing invalid class, memory leakage i.e. forcing the system to work with huge amount of data, database server being unresponsive, etc. These errors may be because of users, developers or hardware resources.

Internally, when an exception is encountered during the execution, SoapUI will try to find the handler. Handler is the block that contains the code to catch the exception.

Exceptions are categorized two types:

1. *Runtime exception*
2. *Compile time exception – not applicable to SoapUI as it does not have an explicit compiler*

*Look at the following screenshot which shows us a runtime exception for invalid code. In the below script we tried to divide an integer by 0. In the error dialog it can be seen that the exception is raised from java libraries and the error message is **Division by zero**.*



We can catch this exception during the execution and handle it programmatically. Before that we will see some of the important keywords that are used in the java exception concepts. Some of the keywords can be used in groovy script also. They are:

- **Throw** – This keyword help us to throw an exception manually i.e. to throw user defined exceptions
- **Throws** – It is used to call the pre-defined exceptions from the method definition. So it will catch the exception if there is any runtime error found during the execution.
- **Try and Catch** – "try" keyword is used with "catch" keyword. If we can predict the portion of the program where the exception can arise during the execution, we can use "try" block in that place. At end of the "try" block, "catch" block should start to catch exception. Inside the catch block, we have to write the handler to handle the exception.

- **Finally** – This is the default and optional block in the exception structure. If we need any statements to be executed at end of the program, like cleaning unused objects, closing connections etc. that can be done inside this block.

The following is the general structure of an exception:

```
try

{

<code to be executed>

}

catch <exception name>

{

<exception handler>

}

finally

{

<default statements>

}
```

Now let us implement the exception handler in the sample code which we have already seen in the screenshot.

Add new test suite under the **GlobalWeather** project. Then add a test case and groovy script test step under the test step. In the script editor, enter the following script.

```
1      // initializing the
      variables

2      int a =
      10;

3      int b =
      0;

4

5      // try, catch
      block

6      try

7      {

8          // Dividing a number by
          zero

9          int c =
          a/b;

10         log.info('Result :' +
            c);

11     }

12     catch(Exception
        expObj)

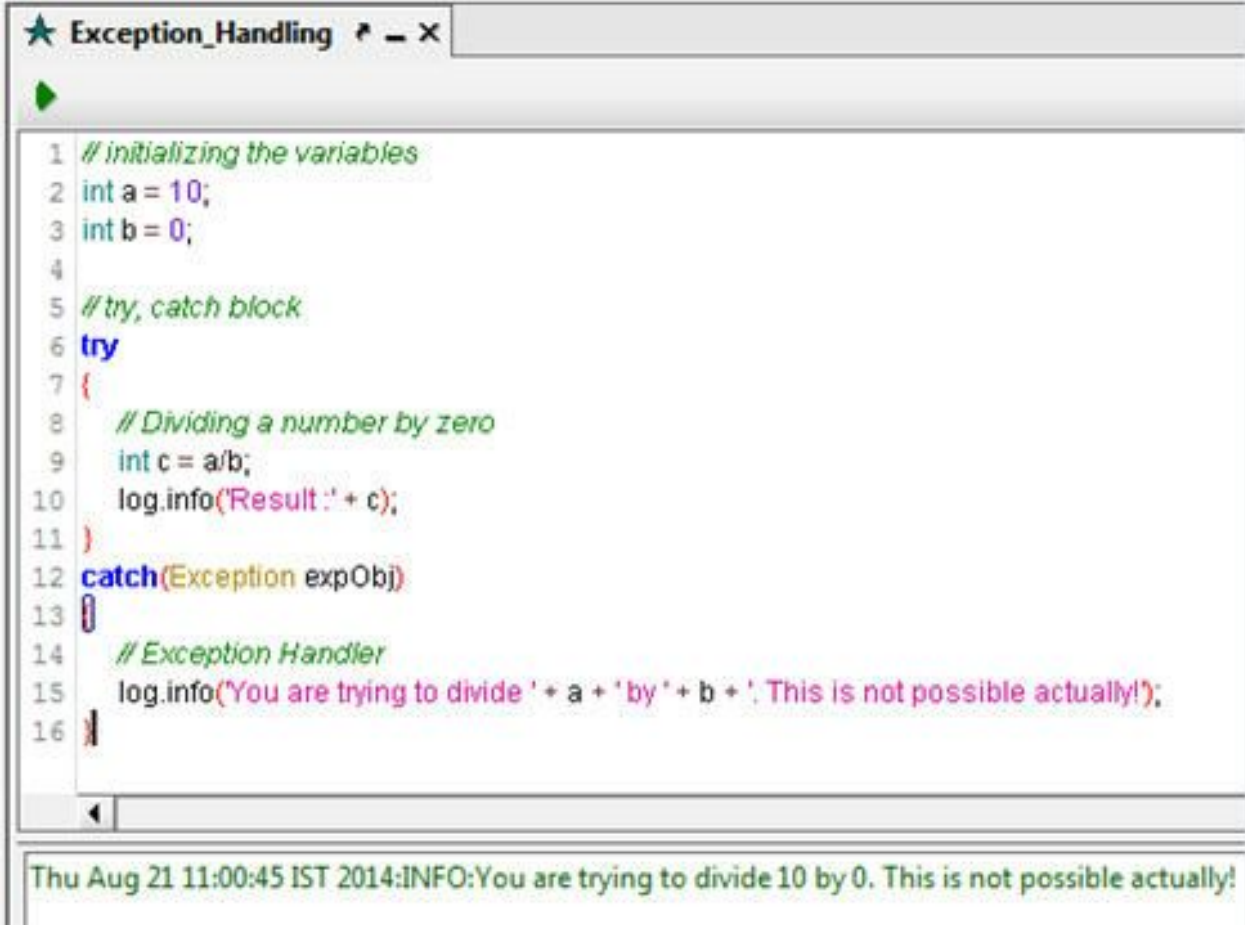
13     {

14         // Exception
        Handler
```



```
15     log.info('You are trying to divide ' + a + ' by  
    ' + b + '. This is not possible actually!');  
  
16 }
```

The above script produces the following result as shown in the screenshot.



The screenshot shows an IDE window titled "Exception_Handling". The code editor contains the following Java code:

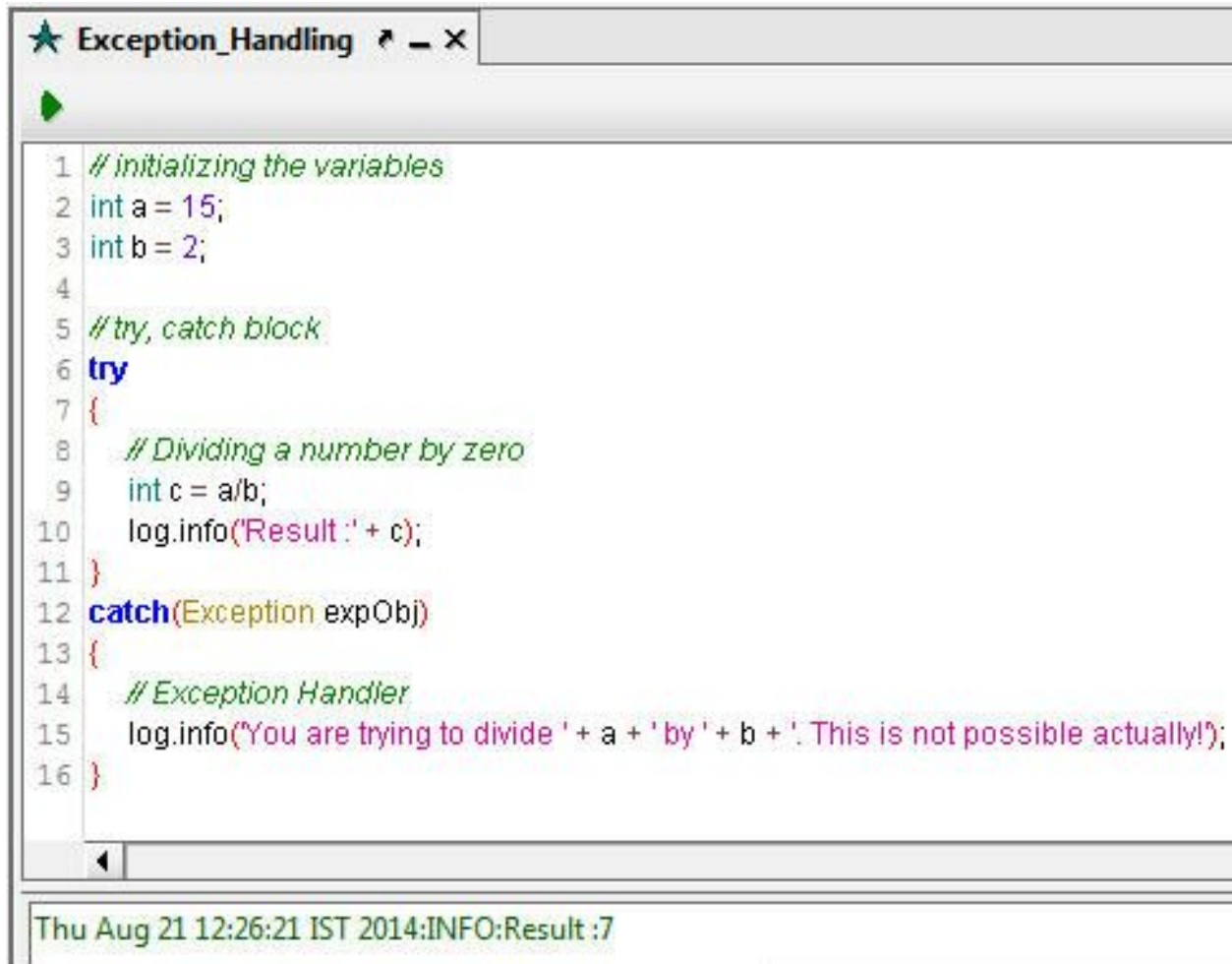
```
1 //initializing the variables  
2 int a = 10;  
3 int b = 0;  
4  
5 //try, catch block  
6 try  
7 {  
8     // Dividing a number by zero  
9     int c = a/b;  
10    log.info('Result : ' + c);  
11 }  
12 catch(Exception expObj)  
13 {  
14     // Exception Handler  
15     log.info('You are trying to divide ' + a + ' by ' + b + '. This is not possible actually!');  
16 }
```

Below the code editor, the output console shows the following log message:

```
Thu Aug 21 11:00:45 IST 2014:INFO:You are trying to divide 10 by 0. This is not possible actually!
```

As we discussed earlier, we tried to divide "A" "B" which is zero. So the 'catch' block gets executed and shows the user defined message in the log. See that in the "catch" statement, we have used **Exception** class which is the super class in Java for all the built-in exceptions. All pre-defined exception classes are inherited from **Exception** class. To handle unpredictable runtime exceptions, we can use **Exception** class in the "catch" block.

Let us now modify the above script to get the required result. See the following screenshot:



```
1 // initializing the variables
2 int a = 15;
3 int b = 2;
4
5 // try, catch block
6 try
7 {
8     // Dividing a number by zero
9     int c = a/b;
10    log.info('Result : ' + c);
11 }
12 catch(Exception expObj)
13 {
14     // Exception Handler
15    log.info("You are trying to divide ' + a + ' by ' + b + '. This is not possible actually!");
16 }
```

Thu Aug 21 12:26:21 IST 2014:INFO:Result :7

Let us now try this in our regular web services testing. In the following script, we did not use try-catch block so we will get runtime exception.

```
1 // Initializing array with 5
   elements
2
3 String[] countryNames = new
  String[5];
4
```

```
4      // Assigning values to the
      array

5      countryNames[0] =
        'India';

6      countryNames[1] =
        'Cyprus';

7      countryNames[2] =
        'Canada';

8      countryNames[3] =
        'Austria';

9      countryNames[4] =
        'Mauritius';

10

11     // Iterate the array elements and assign value to
        the global property

12     for(int idx=0; idx<=5;
        idx++)

13     {

14         com.eviware.soapui.SoapUI.globalProperties.setPropertyValue("CountryName", countryNames[idx]);

15

16     def testStep =
        testRunner.testCase.testSteps['GetCitiesByCountry'];
```

```

17     testStep.run(testRunner, context);

18

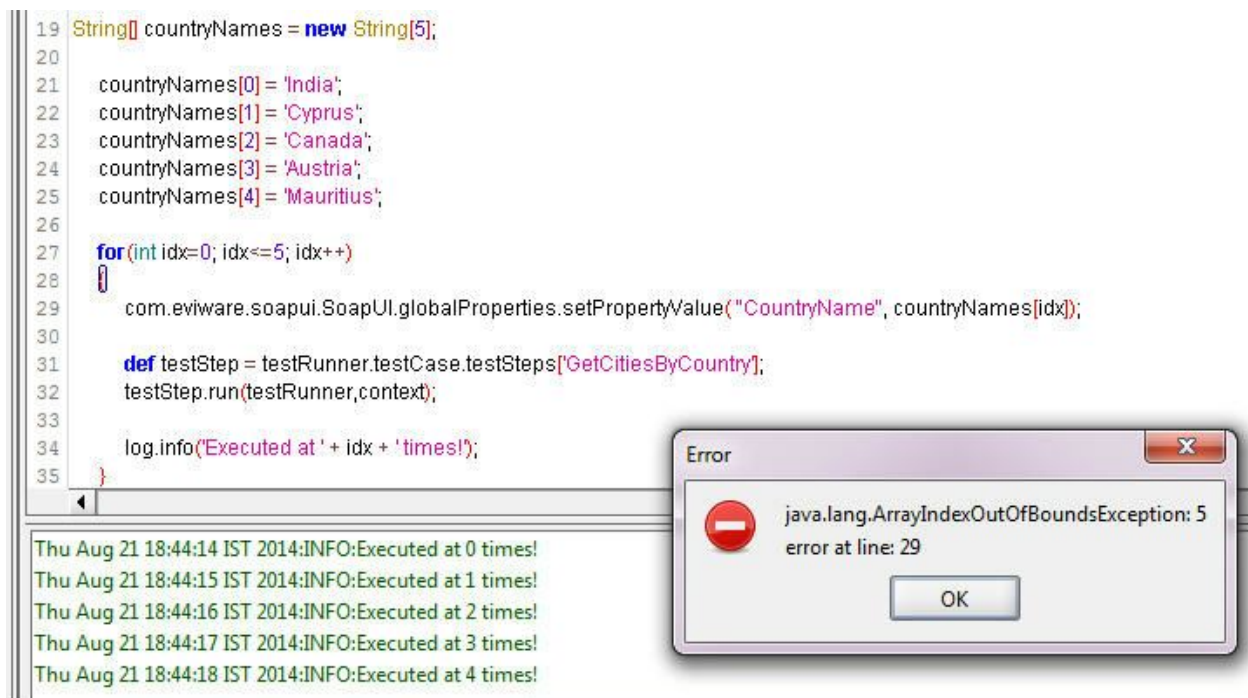
19     log.info('Executed at ' + idx + '
        times!');

20 }

```

The above script will throw an exception called **ArrayIndexOutOfBoundsException** because the script is trying to access invalid array index i.e. 5 which is not available.

(Click image for enlarged view)



As you can see in the above script, we have initialized "countryNames" array with the size of five. It accepts only five string value i.e. country names. Inside the iterative statements, we have checked as **idx <= 5**. So the loop will be iterating up to 6 times and it will try to search 6th element in the array. Since the value will not be there it throws a runtime exception.

To handle this scenario, let us modify the above script as below:

```
1      String[] countryNames = new
      String[5];

2      // Try
      block

3      try

4      {

5          countryNames[0] =
          'India';

6          countryNames[1] =
          'Cyprus';

7          countryNames[2] =
          'Canada';

8          countryNames[3] =
          'Austria';

9          countryNames[4] =
          'Mauritius';

10

11      for(int idx=0; idx<=5;
      idx++)

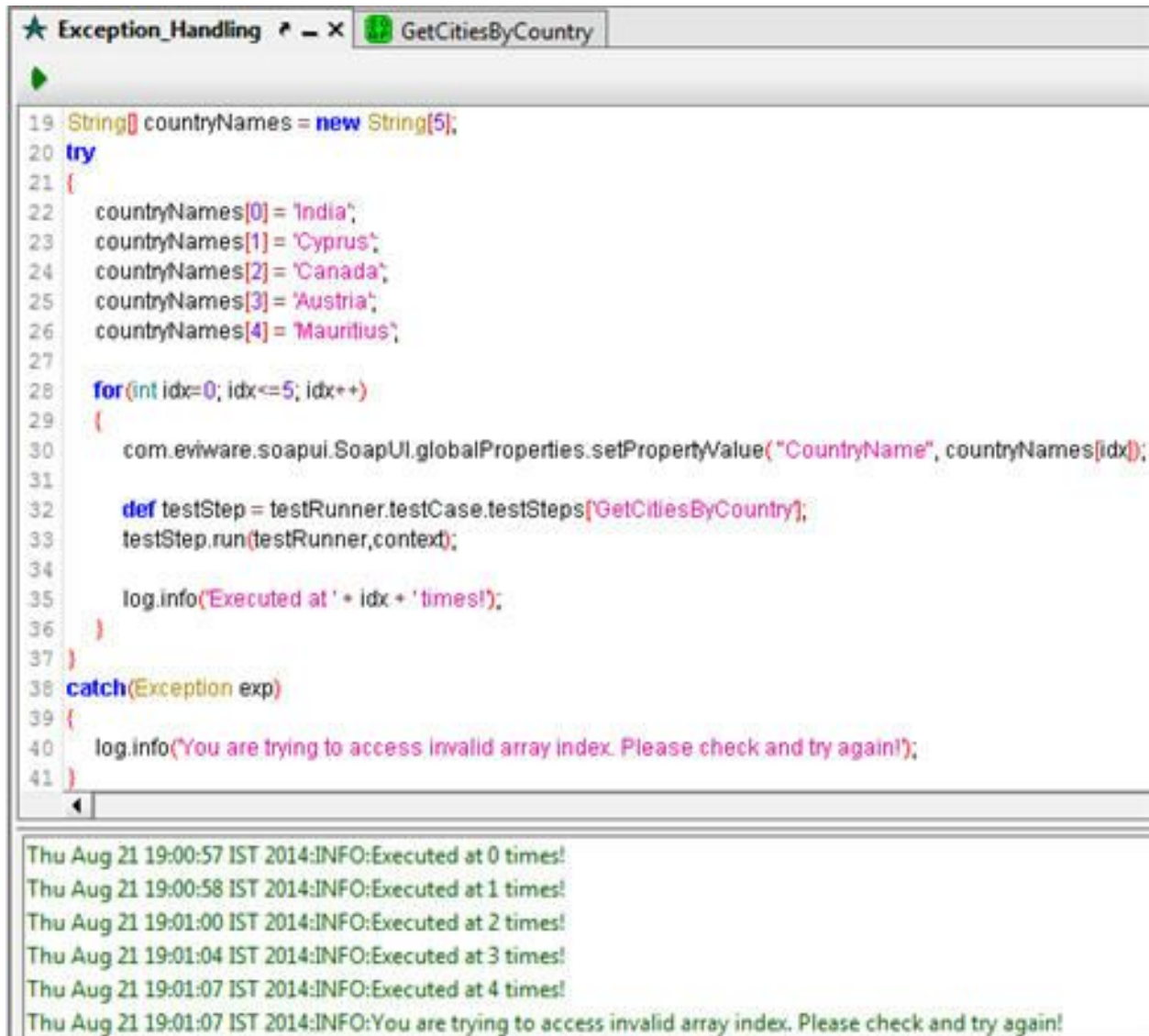
12      {

13

      com.eviware.soapui.SoapUI.globalProperties.setProperty
      Value
```

```
14 | ( "CountryName",  
    | countryNames[idx]);  
  
15 |  
  
16 |     def testStep =  
    | testRunner.testCase.testSteps['GetCitiesByCountry'];  
  
17 |  
    | testStep.run(testRunner,context);  
  
18 |  
  
19 |     log.info('Executed at ' + idx + '  
    | times!');  
  
20 | }  
  
21 | }  
  
22 | catch(Exception exp) // Catch the exception and  
    | displaying the message in the log  
  
23 | {  
  
24 |     log.info('You are trying to access invalid array  
    | index. Please check and try again!');  
  
25 | }
```

Here is the result for the above script.



The screenshot shows an IDE window titled "Exception_Handling" with a sub-tab "GetCitiesByCountry". The code is written in a language that supports try-catch blocks (likely Java or C#). It defines an array of country names and iterates over it, setting a property value and logging each execution. A catch block handles any exception that occurs during the iteration. The logs at the bottom show the program executing successfully for the first five iterations and then throwing an exception on the sixth iteration due to an invalid array index.

```
19 String[] countryNames = new String[5];
20 try
21 {
22     countryNames[0] = 'India';
23     countryNames[1] = 'Cyprus';
24     countryNames[2] = 'Canada';
25     countryNames[3] = 'Austria';
26     countryNames[4] = 'Mauritius';
27
28     for(int idx=0; idx<=5; idx++)
29     {
30         com.eviware.soapui.SoapUI.globalProperties.setPropertyValue("CountryName", countryNames[idx]);
31
32         def testStep = testRunner.testCase.testSteps[GetCitiesByCountry];
33         testStep.run(testRunner,context);
34
35         log.info("Executed at " + idx + " times!");
36     }
37 }
38 catch(Exception exp)
39 {
40     log.info("You are trying to access invalid array index. Please check and try again!");
41 }
```

Thu Aug 21 19:00:57 IST 2014:INFO:Executed at 0 times!
Thu Aug 21 19:00:58 IST 2014:INFO:Executed at 1 times!
Thu Aug 21 19:01:00 IST 2014:INFO:Executed at 2 times!
Thu Aug 21 19:01:04 IST 2014:INFO:Executed at 3 times!
Thu Aug 21 19:01:07 IST 2014:INFO:Executed at 4 times!
Thu Aug 21 19:01:07 IST 2014:INFO:You are trying to access invalid array index. Please check and try again!

This is how we can handle runtime exception during our program execution.

Note: we can use **ArrayIndexOutOfBoundsException** in the "catch" block directly instead of using **Exception** class. If we put the exact exception name in the "catch" block", it will catch only when the particular exception is thrown. If any other pre-defined exceptions thrown, catch block will be failed.

A good automation script should have proper exception handlers. Otherwise it will be difficult to monitor every moment of the execution.

As I have mentioned earlier, groovy script supports "throws" keyword to throw pre-defined exception to the caller.

See the below sample script to understand this concept:

*<return-type> <method name> [arguments / parameters] **throws**<exception name>*

```
{  
  
    <statements to be executed>  
  
}
```

Here's the sample code for the above skeleton.

```
1      // Invoke  
      Method  
  
2      MethodWithThrowKeyword(  
      );  
  
3  
  
4      void MethodWithThrowKeyword()  
      throwsArrayIndexOutOfBoundsException  
  
5      {  
  
6          String[] countryNames = new  
          String[5];  
  
7  
  
8          countryNames[0] =  
          'India';
```



```
9      countryNames[1] =  
      'Cyprus';  
  
10     countryNames[2] =  
      'Canada';  
  
11     countryNames[3] =  
      'Austria';  
  
12     countryNames[4] =  
      'Mauritius';  
  
13  
  
14     for(int idx=0; idx<=5;  
      idx++)  
  
15     {  
  
16         log.info('Country Names: ' +  
      countryNames[idx]);  
  
17     }  
  
18 }
```

In the above script, the **ArrayIndexOutOfBoundsException** will be thrown to the called function. There we need to handle properly with try-catch block. Otherwise, an exception will be thrown by SoapUI.

Conclusion:

Implementing exception handling in our regular web services related testing scripts, will be helpful for us to maintain the code and reduce manual intervention/monitoring by testers. We can use multiple try-catch blocks when required in the script.

