Advanced Strategies

This lesson covers advanced strategies that will help you develop more robust, reusable test suites.

Sections include:

Creating Reusable (Modular) Test Suites

In many cases, you may want to create a test suite that can be reused by other test suites. A common example is a test suite that logs into a website. Once such a test suite is created, it can be used by other test suites in various scenarios that require a login.

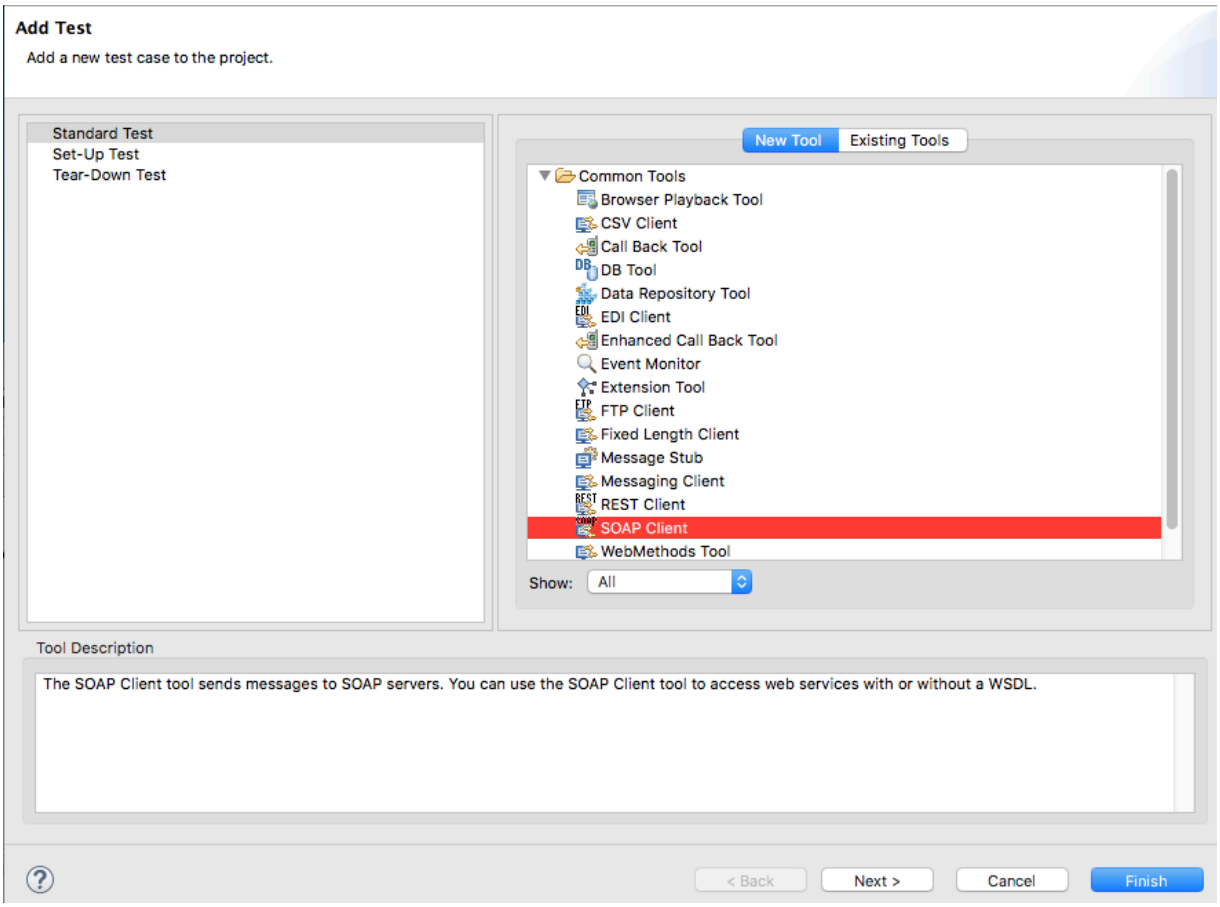
Two SOAtest features are especially helpful for the creation and use of reusable test suites:

- **Referenced test suites:** Once a reusable module or test suite has been created, it can be referenced by another test suite.
- **Variables:** You can parameterize tests with variables, which can be set to specific values from a central location, extracted from a tests (e.g., through a data bank tool), or set from data sources.

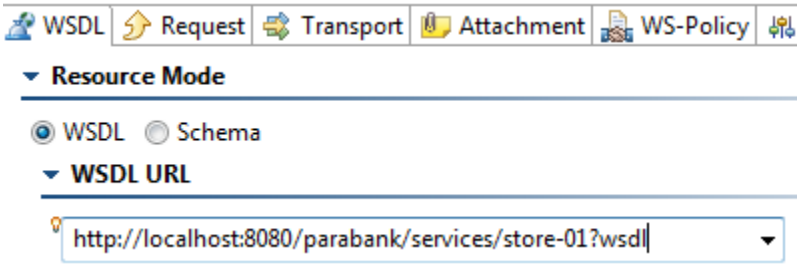
This lesson demonstrates how those two features can be used to create and use reusable test suites. For simplicity, we will use the store web service; however, the principles and steps below can then be applied to any scenario that you create.

To create a reusable test suite:

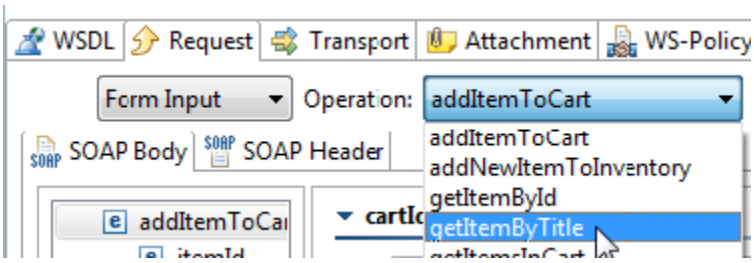
1. Create an empty test suite (.tst) file called `ReusableModule.tst` as follows:
    1. Right-click the **Examples** project node in the Test Case Explorer, then choose **Add New> Test (.tst) file**.
    2. Under **File name**, enter `ReusableModule`.
    3. Click **Next**.
    4. Select **Empty**, then click **Finish**.
2. Create a SOAP Client test as follows.
    1. Expand the **ReusableModule.tst** Test Case Explorer node.
    2. Right-click the **Test Suite: Test Suite** node, then choose **Add New> Test**.
    3. In the dialog that opens, select **SOAP Client**, then click **Finish**.



3. Configure the SOAP Client test as follows:
    1. In the test configuration panel's **WSDL** tab, enter `http://localhost:8080/parabank/services/store-01?wsdl` for the WSDL URL.

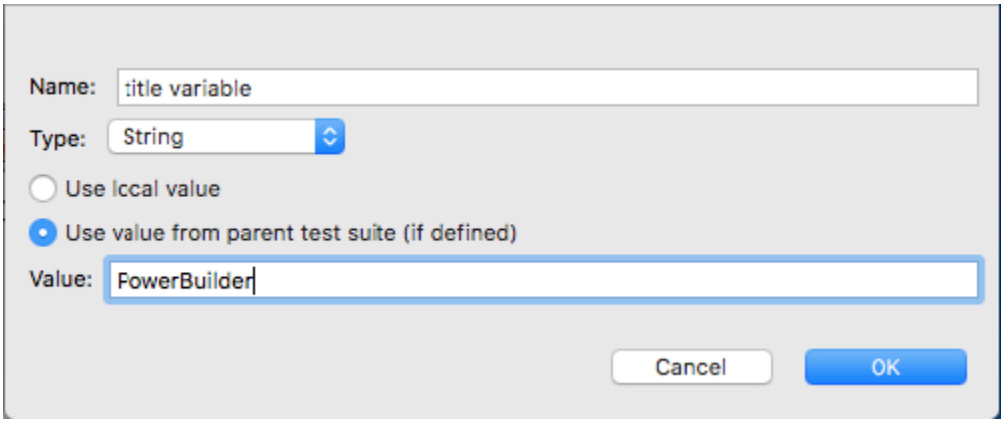


    2. In the **Request** tab, set **Operation** to **getItemByTitle**.



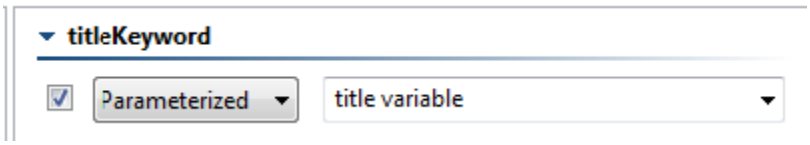
    3. Save the changes to the SOAP Client tool.
4. Define a variable as follows:
    1. Double click the **Test Suite: Test Suite** node to open the test suite configuration panel.
    2. In the **Variables** tab, click the **Add** button.
    3. In the dialog's **Name** field, enter `title variable`.

4. Change **Type** to **String**.
5. Keep the selection at **Use value from parent test suite (if defined)**. This will allow the variable to use values set in the test suite that references this test suite. If the selection is changed to **Use local value**, the value of the variable will always be the value specified in the **Value** field.
6. Enter `PowerBuilder` for **Value**. This is the default value that will be used if SOAtest does not find a data source named `store` with a column named `title`.

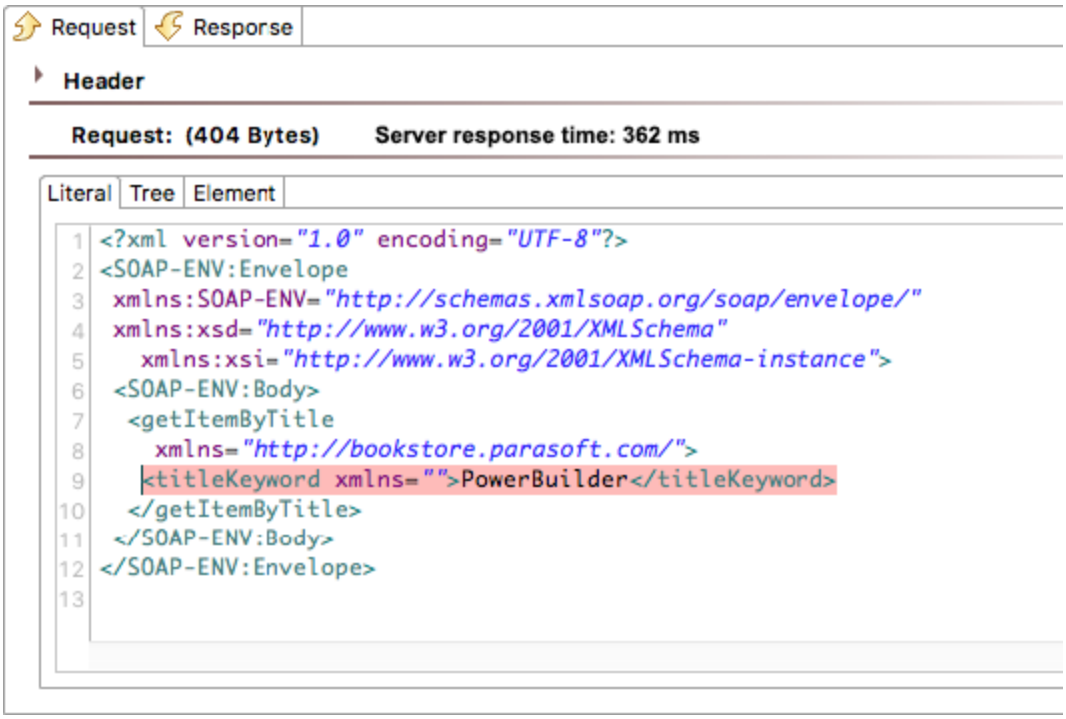


7. Click **OK**.
8. Save the test suite configuration changes.

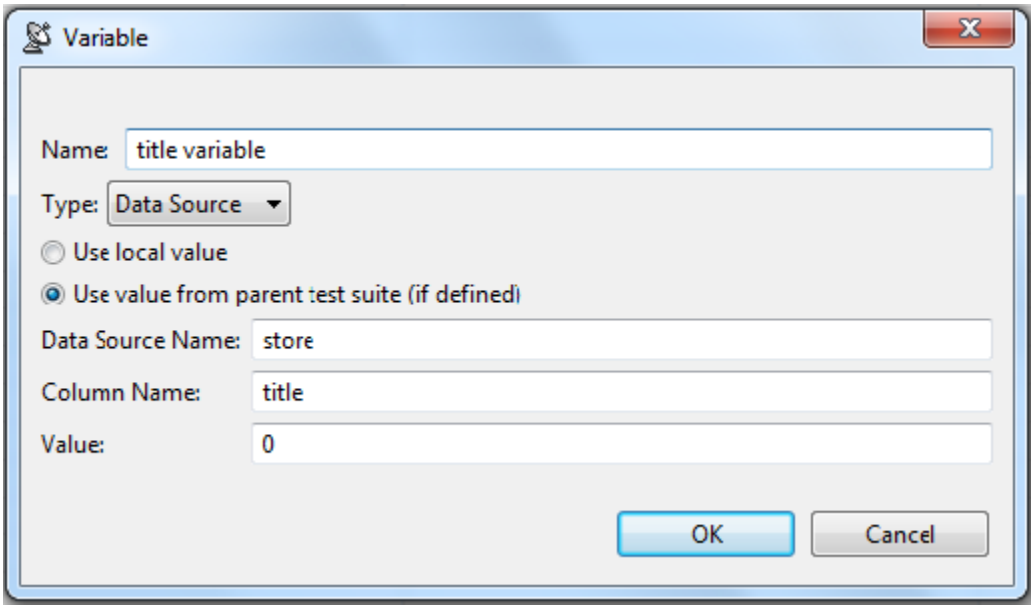
5. Configure the SOAP Client test to use the specified data source values, if available, as follows:
   1. In the SOAP Client's test configuration panel, go to the **Request** tab, check the box for **titleKeyword**, then change **Fixed** to **Parameterized**.
   2. Select **title variable** in the combo box.



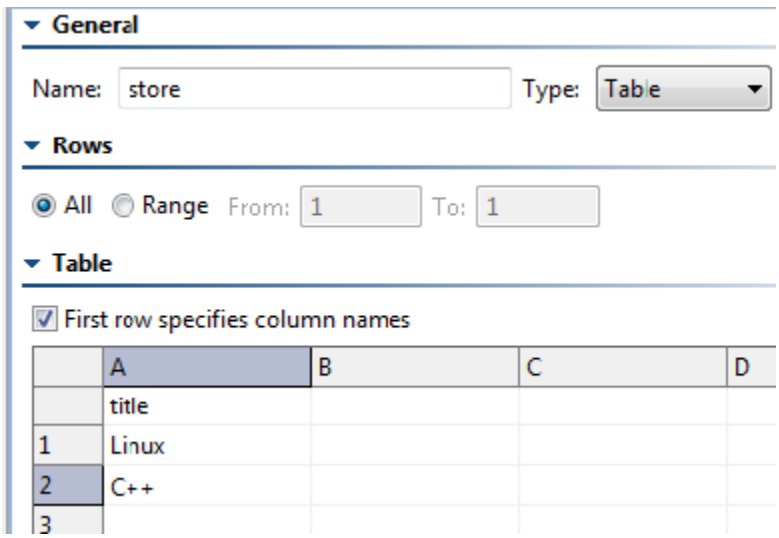
   3. Save the changes to the SOAP Client tool.
6. Run the test suite by selecting **ReusableModule.tst**, then clicking the **Test** toolbar button.
7. Double-click the SOAP Client test's **Traffic Viewer** node to see the traffic. Note that the titleKeyword used was **PowerBuilder**. SOAtest used the default variable value because it did not find the specified data source (since we did not create it yet).



8. Create an empty test suite (.tst) file called `TestStoreTitles.tst` as follows:
    1. Right-click the **Examples** project node in the Test Case Explorer, then choose **Add New> Test (.tst) file**.
    2. Under **File name**, enter `TestStoreTitles`.
    3. Click **Next.**
    4. Select **Empty**, then click **Finish**.
9. Define a variable as follows:
    1. Double click the **TestStoreTitles.tst> Test Suite: Test Suite** node to open the test suite configuration panel.
    2. In the **Variables** tab, click the **Add** button.
    3. In the dialog's **Name** field, enter `title variable`.
    4. Change **Type** to **Data Source**.
    5. Enter `store` for **Data SourceName** and enter `title` for **Column Name**.



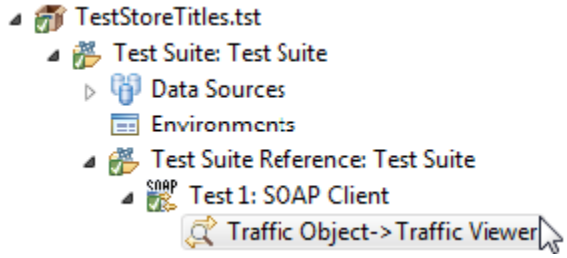
    Note that this test variable will override the string test variable that is defined in the reference .tst file.
10. Click **OK**.
11. Add a data source to that test suite as follows:
    1. Right-click the **TestStoreTitles.tst> Test Suite: Test Suite** node, then choose **Add New> Data Source**.
    2. Select **Table**, then click **Finish**.
    3. In the data source configuration panel, change the data source name to `store`.
    4. Check **First row specifies column names**.
    5. Add a column named `title`.
    6. Add two values to that column: `Linux` and `C++`.



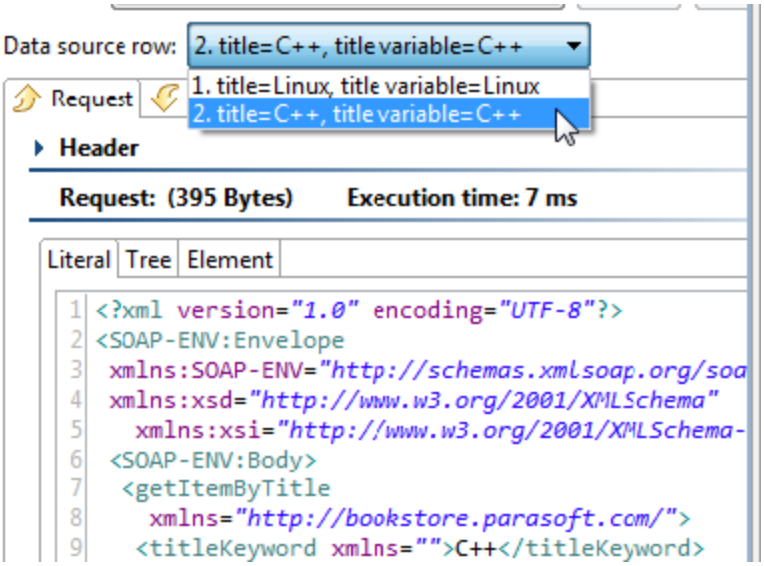
    7. Save the data source changes.
12. Configure this test suite to reference the first test suite we created in this exercise as follows:
    1. Right click **TestStoreTitles.tst> Test Suite: Test Suite** and choose **Add New> Test Suite**.
    2. Select **Reference Test (.tst) File**.

3. Click **Next**.
4. Select the ReusableModule.tst file from your SOAtest workspace.
5. Click **Finish**.

13. Run the current test suite by selecting **TestStoreTitles.tst**, then clicking the **Test** toolbar button.
14. Double-click the **Traffic Viewer** node to see the traffic.



15. Verify that Linux, then C++ were used as the titleKeyword.



Looping Until a Test Succeeds or Fails - Using Test Flow Logic

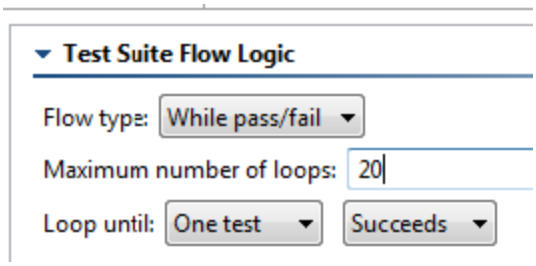
In many cases, you may want to have SOAtest repeatedly perform a certain action until a certain condition is met. Test suite flow logic allows you to configure this.

SOAtest allows you to choose between two main test flow types:

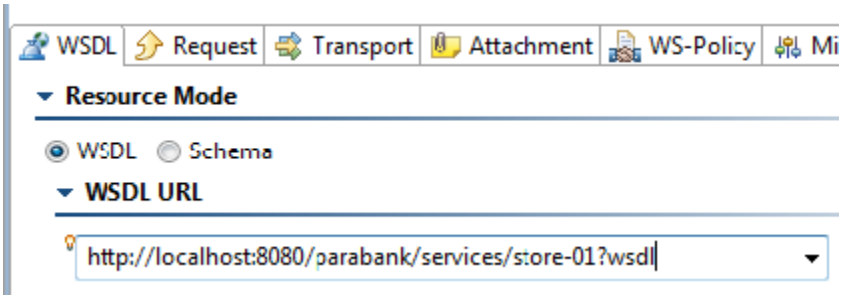
- **While variable:** Repeatedly perform a certain action until a variable condition is met.
- **While pass/fail:** Repeatedly perform a certain action until one test (or every test) in the test suite either passes or fails (depending on what you select in the "loop until" settings). Note that if you choose this option (for instance, with it set to loop until one of the tests succeeds) and the overall loop condition is met (e.g., one test succeeds), then tests that had failures will be marked as succeeding. If the overall loop condition is not met (e.g., no tests succeed), then the individual tests that failed will be marked as failed. The console will show which tests passed and which tests failed— regardless of whether or not the loop condition is met.

To see how while pass/fail logic allows you to have a test suite loop until a specified price value is reached.

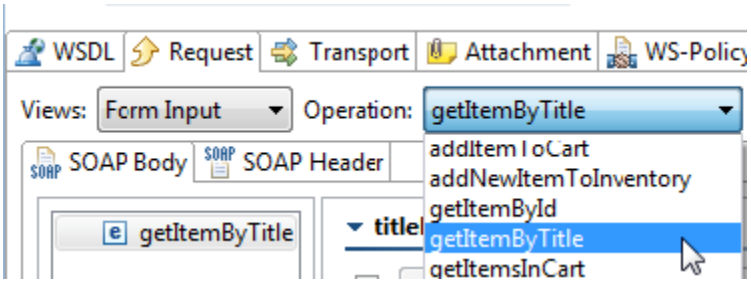
1. Create an empty test suite (.tst) file called `TestFlowLogic.tst` as follows:
    1. Right-click the **Examples** project node in the Test Case Explorer, then choose **Add New> Test (.tst) file**.
    2. Under **File name**, enter `TestFlowLogic`.
    3. Click **Next**.
    4. Select **Empty**, then click **Finish**.
2. Open the test suite configuration panel by expanding the test suite, then double-clicking the **Test Suite: Test Suite** node.
3. Open the **Execution Options> Test Flow Logic** tab.
4. Set the **Flow type** to **While pass/fail**.
5. Set the **Maximum number of loops** to `20`.
6. Change the **Loop until** setting to **One test** and **Succeeds**.



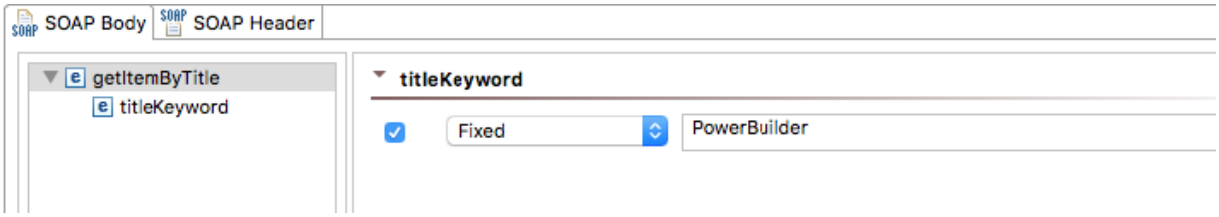
7. Save the test suite configuration settings.
8. Create a SOAP Client test as follows.
    1. Right-click the **TestFlowLogic.tst> Test Suite: Test Suite** node, then choose **Add New> Test**.
    2. In the dialog that opens, select **SOAP Client**, then click **Finish**.
9. Configure the SOAP Client test as follows:
    1. In the test configuration panel's **WSDL** tab, enter `http://localhost:8080/parabank/services/store-01?wsdl` for the WSDL URL.



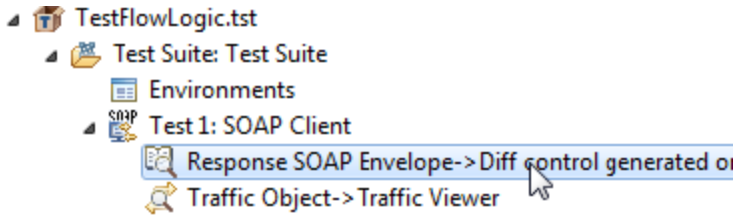
    2. In the **Request** tab, set **Operation** to **getItemByTitle**.



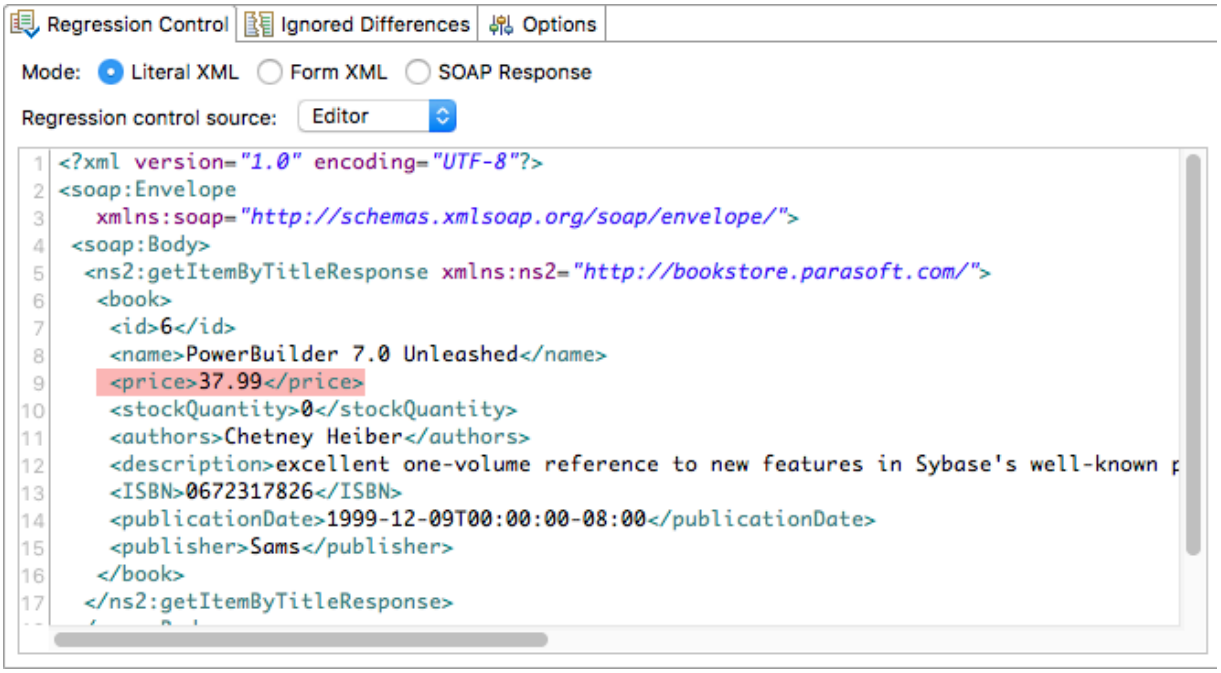
    3. For **titleKeyword**, check the box, then enter `PowerBuilder` as the value.



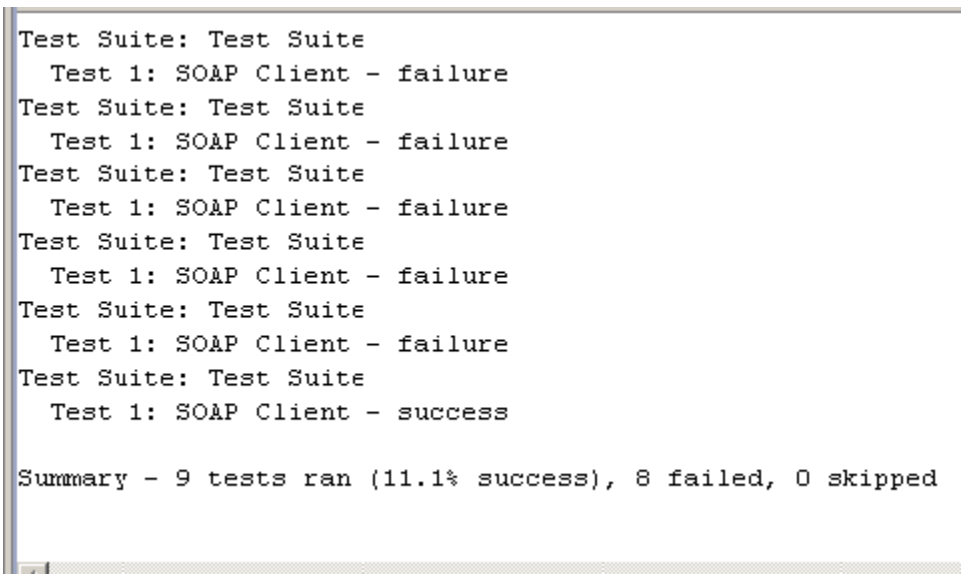
    4. Save the changes to the SOAP Client test.
10. Create a regression control for that test as follows:
    1. Right-click the **SOAP Client** test node and select **Create/Update Regression Control**.
    2. Select **Create Regression Control**.
    3. Click **Finish**.
11. Modify the expected price in the regression control as follows:
    1. Double-click the newly-created **Diff control** node to open the Diff tool editor.



2. Modify the value in the price element. The store service increases the price of the book by $1.00 after several calls to getItemByTitle, so depending on how many times you've ran tests in this tutorial, the current price may be different than shown.



12. Run the current test suite by selecting the test suite node, then clicking the **Test** toolbar button. Even though specific tests failed, the test suite will succeed because the price entered in the expected response will be reached before looping 20 times. You can see this in the Console tab:



Higher prices than expected?

The more times you've called getItemByTitle, the more the prices will have increased. To bring prices back down, restart the Parabank server and rerun the SOAP Client that calls getItemByTitle.

13. Double click the previously-created **Diff Control** node to re-open the Diff tool editor.
14. Modify the price so that the test will loop the full 20 times before the price is reached.
15. Run the Test Suite again. The Test Suite will fail because the price is never reached after looping 20 times.

```
Test Suite: Test Suite
  Test 1: SOAP Client - failure
Test Suite: Test Suite
  Test 1: SOAP Client - failure
Test Suite: Test Suite
  Test 1: SOAP Client - failure
Test Suite: Test Suite
  Test 1: SOAP Client - failure
Test Suite: Test Suite
  Test 1: SOAP Client - failure
Test Suite: Test Suite
  Test 1: SOAP Client - failure

Summary - 20 tests ran (0.0% success), 20 failed, 0 skipped
```

Extending SOAtest with Scripting

If you have a testing requirement which requires you to add custom functionality or logic to your tests cases, SOAtest allows you to easily integrate custom scripts into your testing environment.

Using SOAtest's XML Assertor, you can integrate custom scripts written in into SOAtest. This means that almost any testing situation can be handled with ease, even if the situation is not directly supported by SOAtest's current tool set.

In this example you will create a Scenario Test using the book store service used in previous examples. In this Scenario you will search for a book by its title, then validate that the price of the book is an even integer.

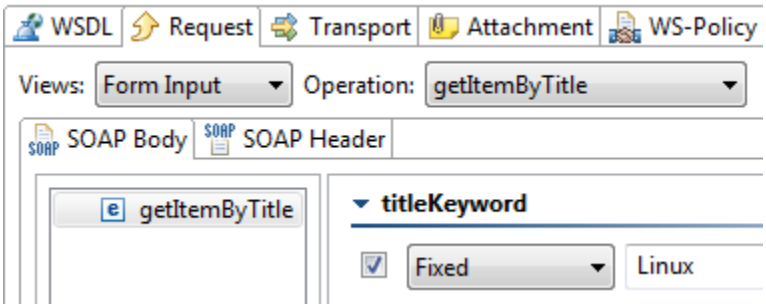
When you complete this section of the tutorial, your test suite should resemble the test suite entitled "Custom Scripting" in the `SOAtestTutorial.tst` file.

1. Select the **Test Suite: Functional Tests** node (created in Creating Test Suites for Unit Tests) and click the **Add Test Suite** toolbar button.

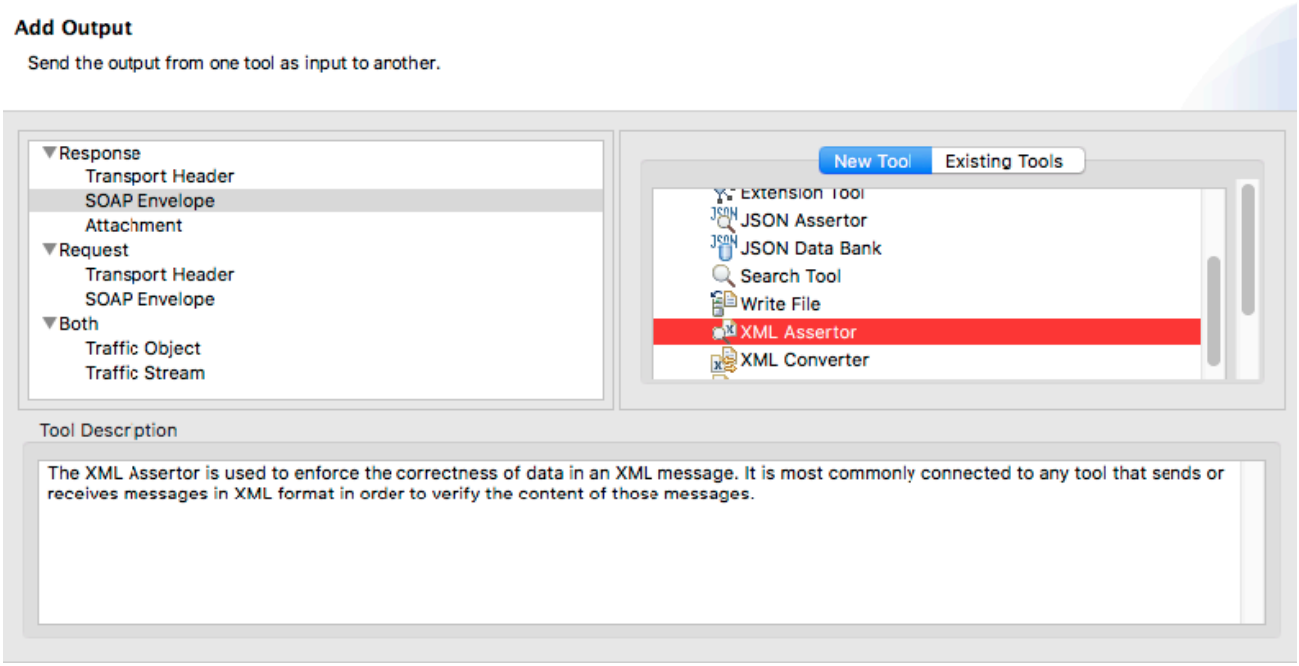
2. In the Add Test Suite wizard, click **Empty**, then click **Finish.**
3. Double-click the new **Test Suite: Test Suite** node added to the test suite tree, enter `Custom Scripting` in the **Name** field in the test configuration panel, then click **Save**.
4. Select the **Test Suite: Custom Scripting** node and click the **Add test or output** button.

5. In the Add Test Wizard, select **SOAP Client** in the right, then click **Finish**. A SOAP Client tool is added to the test suite.
6. Double-click the **Test 1: SOAP Client** node underneath the **Test Suite: Custom Scripting** node and enter `Validate Price Value` in the **Name** field in the right GUI Panel.
7. In the **WSDL** tab of the test configuration panel, enter `http://localhost:8080/parabank/services/store-01?wsdl` in the **WSDL URL** field.
8. Open the **Request** tab, then select **getItemByTitle** from the **Operation** drop down box.
9. In the **SOAP body** tab, click **getItemByTitle** and enable the **titleKeyword** option.

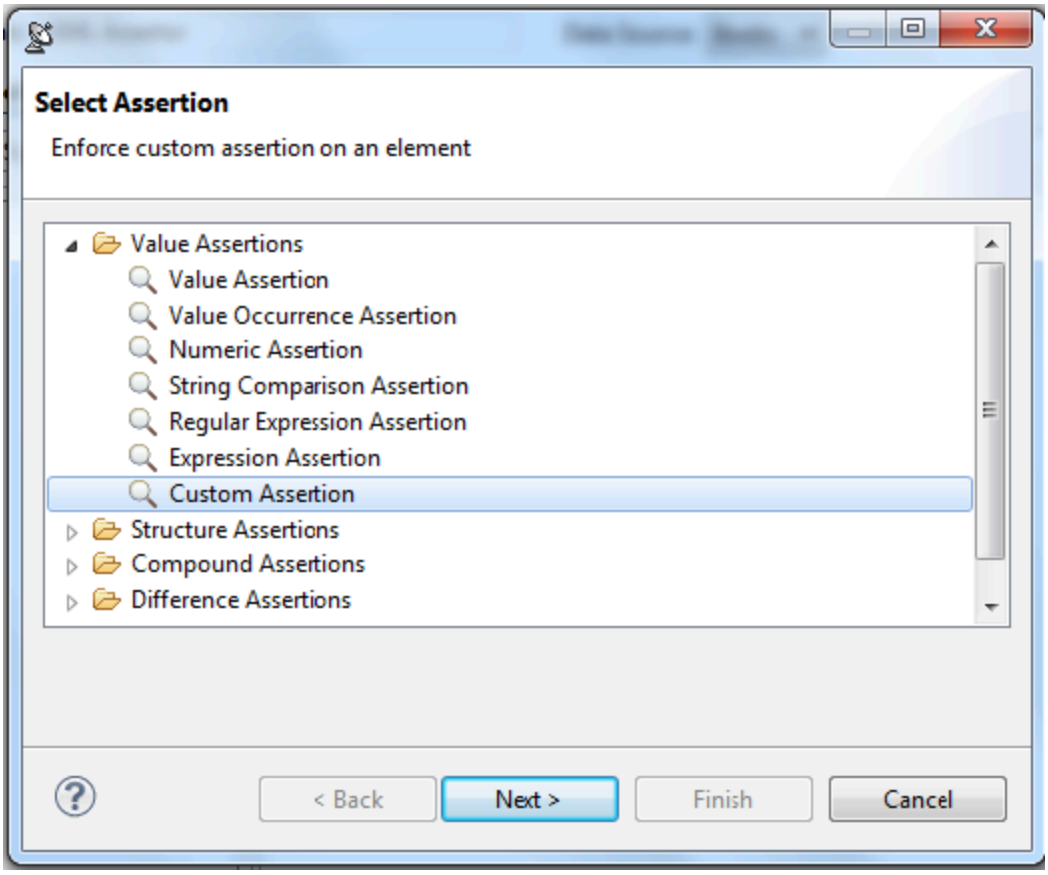
10. Choose **Fixed** from the drop-down menu and enter `Linux` as the value.
11. Click the **Save** toolbar button.



12. Right-click the **Test 1: Validate Price Value** node and select **Add Output**.
13. In the **Add Output** wizard, select **Response> SOAP Envelope** on the left, and **XML Assertor** on the right, and click **Finish**. This tells SOAtest to chain an XML Assertor to the XML Response output of the SOAP Client.

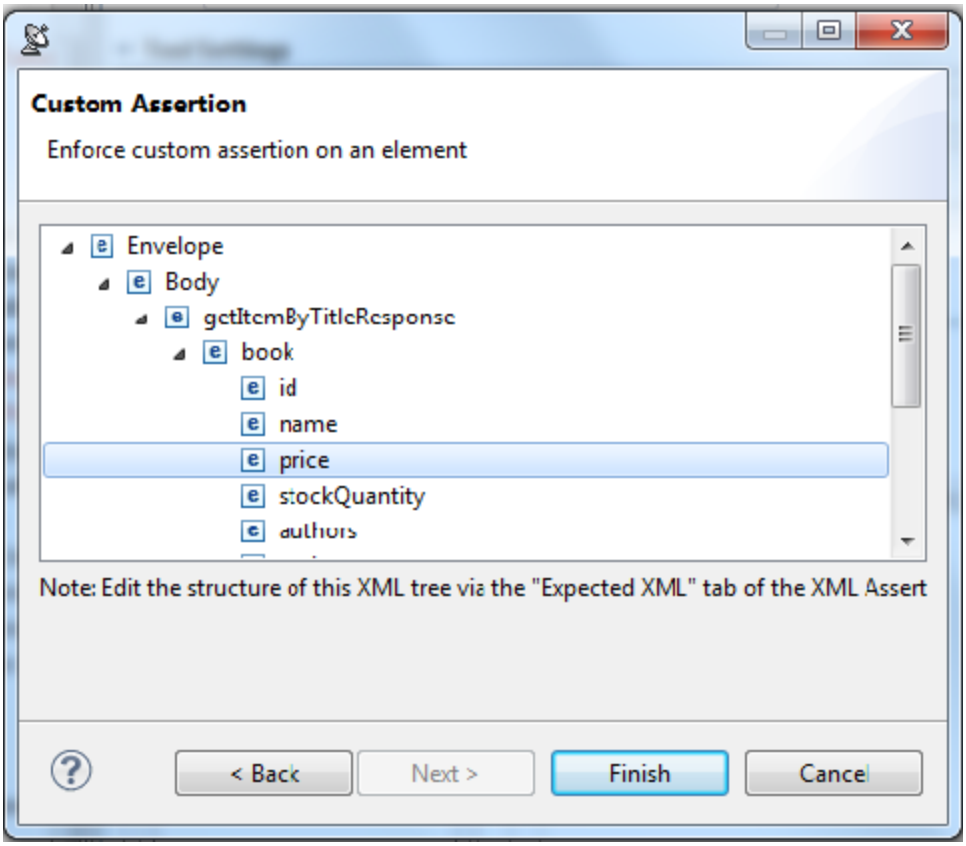


14. Open the **Configuration** tab within the XML Assertor test configuration panel, then click the **Add** button.
15. In the **Select Assertion** dialog, expand the **Value Assertion** node, select **Custom Assertion**, and click the **Next** button.



The Custom Assertion dialog then displays a tree view of the XML message from which you can select a single value to enforce.

16. Select the **price** element in the XML tree view and click the **Finish** button.

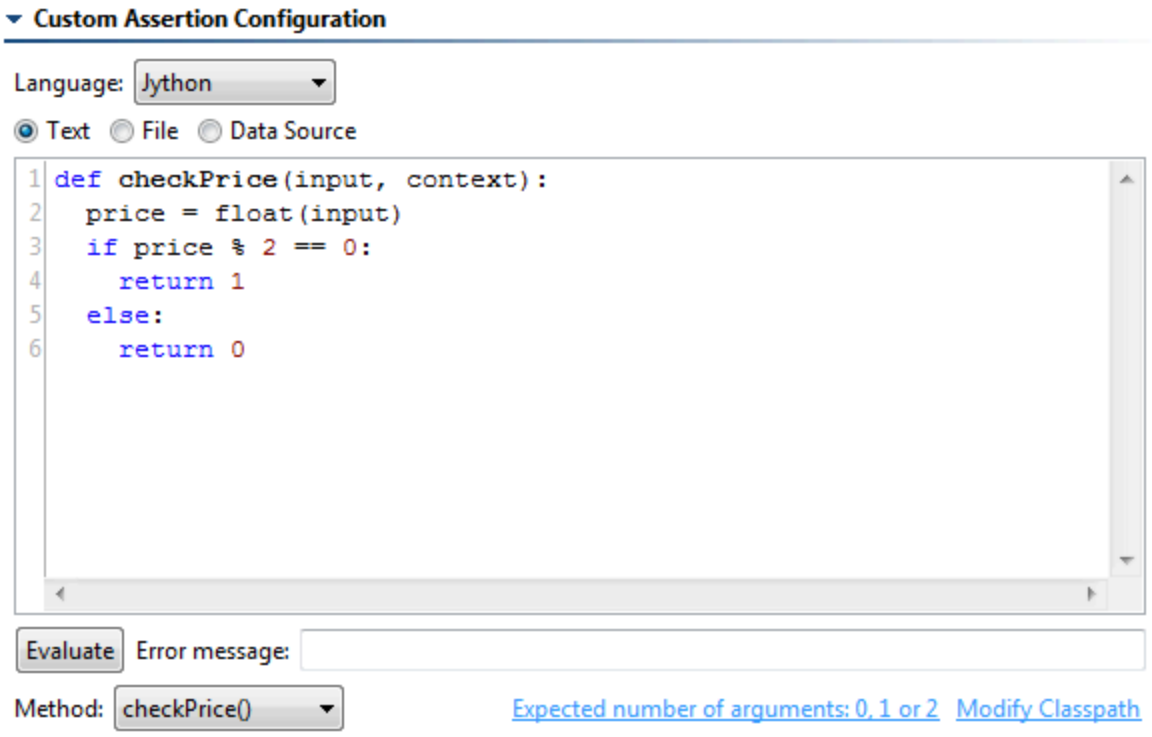


The test configuration tab will now be populated with a Custom Assertion.

17. Select **Jython** from the **Language** drop-down menu.
18. Enter the following script, which ensures that the price value is even, in the **Text** field of the test configuration tab:

```
def checkPrice(input, context):
        price = float(input)
        if price % 2 == 0:
                return 1
        else:
                return 0
```

19. Select **checkPrice()** from the **Method** drop-down menu.



20. Click the **Save** toolbar button.
21. Select the **Test 1** node and click the **Test** button. Notice that the test fails. If you double-click the test's **Traffic Viewer** node, you will see that the price of the Linux book is an odd number, causing the test to fail.

Further Extending SOAtest

SOAtest's built-in extensibility framework can be used in your scripts, allowing for the definition and use of custom transport protocols and message implementations within SOAtest. For more information, go to **Parasoft> Help> Parasoft SOAtest Extensibility API**.