

## Class Objectives

- Understand architecture and full capabilities of soapUI Pro
- Put soapUI's features to work to build powerful tests
- Combine unit tests into multi-faceted functional and load tests

## Full Training Agenda

- Introductions
- Methodology
- soapUI Architecture
- Interfaces, Operations, Requests
- TestSteps
- Introduction to Assertions
- Introduction to Functional Tests
- Requirements

## Full Training Agenda

- Introduction to Mock Services
- Data-driven Tests
- Refactoring
- Introduction to Load Testing
- Introduction to Groovy
- Test Coverage
- Web Service Interoperability
- Monitoring
- Introduction to Reporting

## Full Training Agenda

- Advanced Mock Services
- Advanced Groovy
- Introduction to Security
- Event Handlers



# TestSteps

- TestStep Usage
- Types of TestSteps
- Sequencing TestSteps

*Note: To keep the user interface uncluttered, before beginning this unit please close all open windows and projects, and remove any project entries from the Navigator.*

Prior to beginning the exercises in this unit, please follow these steps:

1. Choose the **File -> Import Project** menu option. Navigate to the **Echo-soapUI-project.xml** file (provided to you by your instructor)
2. Once the project is open, right-click on the **EchoSOAPBinding MockService** entry, and choose the **Start Minimized** option.
3. Minimize this project's entry in the Navigator by clicking on the '-' symbol.



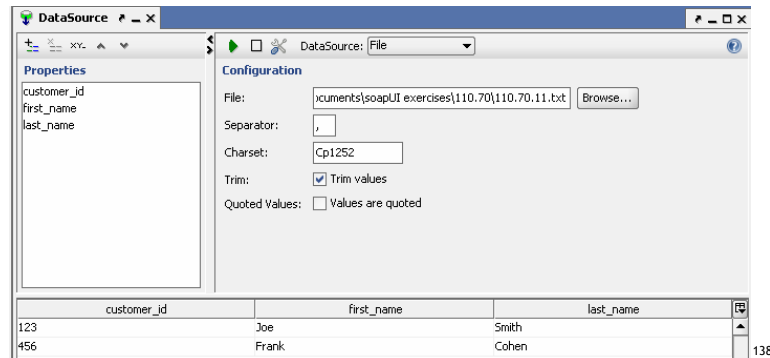
## TestSteps

- Test Request
- Groovy script
- Properties
- Property Transfer
- Conditional Goto
- Delay
- Run TestCase
- REST Test Request
- HTTP Test Request
- Mock Response
- JDBC Request
- JMS Request
- AMF Request
- DataSource
- DataSource Loop
- DataSink
- DataGen



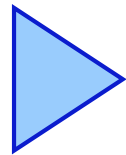
## Exercise 11: Create a File DataSource TestStep

- Purpose: See how to consume external data in support of this TestCase
- Level: Beginner



soapUI Training  
Copyright © 2007 - 2010 Think88 Ventures LLC. All rights reserved.

1. Create a new project using the following WSDL:  
**<http://localhost:8088/mockEchoSOAPBinding?WSDL>**  
Check the **Create Requests** and **Create TestSuite** boxes. Accept all the defaults.
2. Accept the defaults for generating a TestSuite.
3. Using a text editor, create and save a file with the following data  
**123, Joe, Smith**  
**456, Frank, Cohen**
4. Save the file (remember the directory where you saved it!).
5. Right-click on the **Forward** TestCase.
6. Choose the **Add Step -> Data Source** menu option.
7. Click on the **Add a New Property** icon (upper left corner of the window), and then add 3 properties: **customer\_id, first\_name, last\_name**
8. After configuring the properties, configure the data source on the right hand side of the window.
9. Select the **File** data type from the DataSource drop-down, and browse to the file you created in step 4.
10. Click on the **Run** icon at the top of the screen to run this TestStep and see how your data will be retrieved from the file.
11. Save your project; you will use it in the next exercise.



## Introduction to Assertions

- Usage
- Types of Assertion
- Creating Assertions from Results
- Tying Multiple Steps Together

*Note: To keep the user interface uncluttered, before beginning this unit please close all open windows and projects, and remove any project entries from the Navigator.*

Prior to beginning the exercises in this unit, please follow these steps:

1. Choose the **File -> Import Project** menu option. Navigate to the **Complaint3-soapUI-project.xml** file (provided to you by your instructor)
2. Once the project is open, right-click on the **ComplaintSOAPBinding MockService** entry, and choose the **Start Minimized** option.
3. Minimize this project's entry in the Navigator by clicking on the '-' symbol.

## Understanding soapUI Assertions

- Help you determine whether a Web service has met certain criteria, including:
  - Has a SOAP response
  - Contains a value/Doesn't contain a value
  - Has a SOAP fault/Doesn't have a SOAP fault
  - Complies with its XML schema
  - Meets a response SLA
  - XPath/XQuery matches
  - Security status (WS-Security)
  - Addressing status (WS-Address)
  - JMS status & timeout
  - Satisfies a script

Assertion	Purpose
Has SOAP response	Check to see if the service call is a valid SOAP reply
Contains a value	Reply from service contains specified information
Doesn't contain a value	Reply from service does not contain specified information
Has a SOAP fault	Service throws a SOAP exception
Doesn't have a SOAP fault	Service is free from any SOAP errors
Response SLA	Check to see that the service returns information within a specified time threshold
XPath match	Use XPath to check that returned data matches goal
XQuery match	Use XQuery to process results, checking for matches
Security status	Have any security issues occurred?
Addressing status	Are WS-Address headers filled in properly?
JMS status	Did any JMS-related errors occur?
JMS timeout	Does a call to JMS exceed the pre-defined timeout threshold?
Satisfies a script	Apply programmatic validation to results

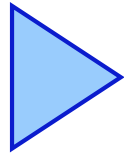


## Exercise 9: Create an XPath Match Assertion

- Purpose: Use XPath syntax to see if a response contains a given value
- Level: Intermediate

The screenshot shows two overlapping dialog boxes in the soapUI application. The primary dialog, titled "XPath Match configuration", has three main sections: "Specify xpath expression and expected result" with a "Declare" button, "XPath Expression" with a text area containing the XPath expression, and "Expected Result" with a text area containing the expected value. The secondary dialog, titled "Select XPath", shows a tree view of an XML document with the "wsdl:ComplaintMessage[1]" node highlighted. Both dialogs have "OK" and "Cancel" buttons at the bottom.

1. Using the current project, remove all assertions by highlighting each assertion and clicking the red **X** found in the assertions area.
2. Add a new assertion, choosing the **XPath Match** value from the drop-down.
3. Click on the **Add assertion** icon, and choose the XPath Match from the drop-down.
4. Click on the **Select node...** icon to identify the information source node.
5. Select the **ComplaintMessage** entry (last item in the XML document) and click **OK**.
6. Notice that soapUI has constructed a full XPath expression and filled in the matching value based on what you selected.
7. Click **Save** to complete preparing the assertion.
8. Enter your desired matching value.
9. Run the test and review the results.
10. Bonus: run the test several times. Change the values to see how the XPath match assertion behaves.



## Introduction to Load Testing

- Functional Load Testing
- Behavioral Load Testing
- Performance Load Testing
- Requirements-Driven Load Testing
- Extending Load Testing

*Note: To keep the user interface uncluttered, before beginning this unit please close all open windows and projects, and remove any project entries from the Navigator.*

Prior to beginning the exercises in this unit, please follow these steps:

1. Choose the **File -> Import Project** menu option. Navigate to the **Complaint1-soapUI-project.xml** file (provided to you by your instructor)
2. Once the project is open, right-click on the **ComplaintSOAPBinding MockService** entry, and choose the **Start Minimized** option.
3. Minimize this project's entry in the Navigator by clicking on the '-' symbol.

## Load Testing

- Four primary styles of load testing:
  - Functional
  - Behavioral
  - Performance
  - Requirements-driven
- Multiple types of load test can be part of one TestCase
  - Lets you measure their effect on each other

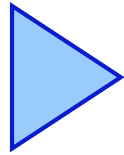


## Exercise 5: Create a Burst Load Test

- Purpose: Run a load test using a burst strategy
- Level: Intermediate

Test Step	min	max	avg	last	cnt	tps	bytes	bps	err	rat
PostComplaint	2	3296	512.31	899	8516	38.61	901180	22594	0	0
TestCase:	2	3296	512.31	899	8516	38.61	901180	22594	0	0

1. Create a new project using the following WSDL:  
**<http://localhost:8088/mockComplaintSOAPBinding?WSDL>**  
Check the **Create Requests** and **Create TestSuite** boxes. Accept the defaults.
2. Double-click on the **PostComplaint** TestStep, and fill in sample values to send to the service.
3. Create a new load test.
4. Configure the settings as follows:
  - **Limit: 300 seconds**
  - **Threads: 20**
  - **Strategy: Burst**
  - **Burst delay: 20**
  - **Burst duration: 60**
5. Click on the **Run** icon to start the test. Observe the burst delay and burst duration counters.
6. Examine the results.



## Advanced Mock Services

- Scripting and Mock Services
- Routing Requests and Responses
- Coverage

*Note: To keep the user interface uncluttered, before beginning this unit please close all open windows and projects, and remove any project entries from the Navigator.*

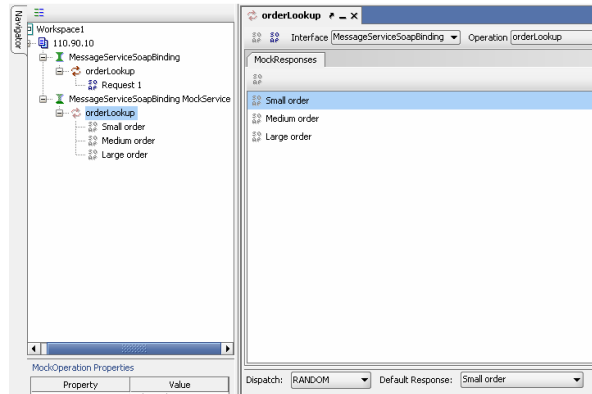
## Mock Services and Responses

- One of the beauties of mock services is that responses are completely configurable
- You can define many different responses for a given operation
- Once responses are defined, you can then route incoming requests in a variety of ways:
  - Script
  - Sequence
  - Query match
  - XPath
  - Random

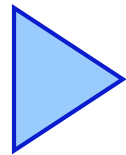


## Exercise 4: Randomly Route Incoming Requests for a Mock Service

- Purpose: Learn how to select a random mock response from a list of available responses
- Level: Intermediate



1. Use the project you created in the last exercise.
2. Expand the Mock Service.
3. Double-click on the **Small order** mock response. Set the **orderAmount** return value to **25**.
4. Double-click on the **Medium order** mock response. Set the **orderAmount** return value to **50**.
5. Double-click on the **Large order** mock response. Set the **orderAmount** return value to **100**.
6. Double-click on the **orderLookup** operation.
7. Explore the two drop-downs at the bottom of the window.
8. Choose the **RANDOM** as the **Dispatch** option.
9. Start the Mock Service.
10. Right-click on the **orderLookup** operation, and choose **Open Request**.
11. Repeatedly run the request (you don't need to fill in any data).
12. Observe the results – they should vary based on random routing.
13. Stop the Mock Service.
14. Save your project.



## Advanced Groovy

- The soapUI object model
- Dynamic management of tests
- Extending soapUI with Groovy

*Note: To keep the user interface uncluttered, before beginning this unit please close all open windows and projects, and remove any project entries from the Navigator.*

Prior to beginning the exercises in this unit, please follow these steps:

1. Choose the **File -> Import Project** menu option. Navigate to the **Echo-soapUI-project.xml** file (provided to you by your instructor)
2. Once the project is open, right-click on the **EchoSOAPBinding MockService** entry, and choose the **Start Minimized** option.
3. Minimize this project's entry in the Navigator by clicking on the '-' symbol.

## Navigating XML in Groovy

- soapUI provides XmlHolder utility object
  - XPath access to reading XML Document and DOM
  - XML construction APIs too
    - DOM Construction APIs
    - Standard Groovy utilities
      - Complex namespace usage is a problem

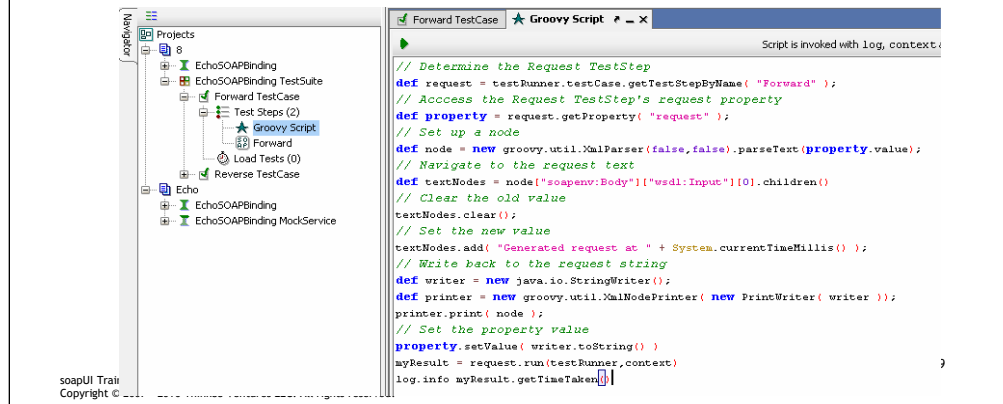
- **GroovyUtils**

```
def groovyUtils =  
new com.eviware.soapui.support.GroovyUtils( context )
```



## Exercise 8: Set an XML Node Value, Issue Request

- Purpose: Learn how to dynamically locate and modify a node, and then launch a request
- Level: Advanced



1. Create a project. Use the following WSDL: <http://localhost:8088/mockEchoSOAPBinding?WSDL>.

Check the **Create Requests** and **Create TestSuite** boxes, and accept all defaults.

2. Open the **EchoSOAPBinding** TestSuite, and **Forward** TestCase.

3. Add a Groovy script TestStep, and enter the following code:

```

// Determine the Request TestStep
def request = testRunner.testCase.getTestStepByName( "Forward" );
// Access the Request TestStep's request property
def property = request.getProperty( "request" );
// Set up a node
def node = new groovy.util.XmlParser(false,false).parseText(property.value);
// Navigate to the request text
def textNodes = node["soapenv:Body"]["wsdl:Input"][0].children()
// Clear the old value
textNodes.clear();
// Set the new value
textNodes.add( "Generated request at " + System.currentTimeMillis() );
// Write back to the request string
def writer = new java.io.StringWriter();
def printer = new groovy.util.XmlNodePrinter( new PrintWriter( writer ) );
printer.print( node );
// Set the property value
property.setValue( writer.toString() )
myResult = request.run(testRunner,context)
log.info myResult.getTimeTaken()

```

4. Run your script, and examine the request/response, and the log