**Groovy scripting in soapUI**

* The Groovy scripts inside soapUI have access to the following context-related variables:

• context

• testRunner

* soapUI also provides us with a standard log4j Logger object—**log**—that can be used in scripts at any level in a soapUI project.
* The context object holds information about a particular test run session. It can be used to read and write/update context-specific variables.

Usage of the context object using a simple example:

//Get the name of current TestStep

log.info(context.getCurrentStep().getLabel())

//Get the name of parent TestCase

log.info(context.getTestCase().getLabel())

//Get the name of TestSuite

log.info(context.getTestCase().getTestSuite().getLabel())

//Get the name of the soapUI project

log.info(context.getTestCase().getTestSuite().getProject().getName())

* The context object is useful in situations where you want to read the property values of TestStep.

**prop = context.getProperty('SOAP Request','Endpoint')**

**log.info(prop)**

**Note:** that you cannot read the property value from a TestStep of a different TestCase using this method. You will get null value.

* Find out what properties are available during a single run of a TestStep.

String[] props= context.getPropertyNames()

for (prop in props){

log.info(prop)

}

**Output:**

Wed Mar 11 17:31:38 IST 2018:INFO:ThreadIndex

Wed Mar 11 17:31:38 IST 2018:INFO:log

Wed Mar 11 17:31:38 IST 2018:INFO:RunCount

Double-click on the **TestCase** and run the TestCase and you will see all 19 properties such as

**httpMethod**, **requestUri**, and **postMethod**, which are available at the TestCase context of the run session.

* We can set the properties and retrieve them later during a particular test run.

message = 'hello world'

context.setProperty('msg',message)

log.info(context.getProperty('msg'))

* The context.expand (<String>) method is a useful method, This can be used in multiple situations and the simplest usage is for accessing a custom property at a different level of a test. If we have a custom property at project level (for example, Test), then we can use the expand method to read the property value from a script which runs from the TestStep level:

**log.info(context.expand( '${#Project#Test}'))**

* The context object is very useful if we want to store some value in one TestStep and use it in subsequent script test steps:

1. Add another GrovyScript TestStep in the TestCase.

Let's name it **GroovyTestScript2**:

2. Add the following into GroovyTestScript1 to define a new property,

holder in context:

context.holder="testing"

3. Now, add the following in GroovyTestScript2 to read the property value:

holderValue = context.getProperty("holder")

log.info(holderValue)

**Note: T**he context is visible inside TestCase.

**The testRunner variable**

* The testRunner variables are used to execute tests in a soapUI project.
* The testRunner interfaces provide methods such as start, cancel, and fail to control the test execution.

Follow these steps to see how testRunner can be used to control test execution flow:

Add the following statement in GroovyTestScript1 and run the

TestCase from the TestCase editor:

testRunner.cancel("CANCELLED THE TEST")

You will see that the further executions of TestCase immediately stop when they reach the preceding statement and the CANCELLED THE TEST message is logged at the TestCase log. This is useful if you want to cancel the test run, based on evaluation of certain conditions.

* Let’s Run a Simple TestCase:

import com.eviware.soapui.model.testsuite.TestRunner.Status

//Get hold of the getRoomDetails TestCase which is at a different

// TestSuite than the current Suite

def TestCase2 = testRunner.testCase.testSuite.

project.testSuites["TestSuite 1"].

testCases["TestCase 2"]

//Run the getRoomDetails TestCase synchronously

def testcaserunner = TestCase2.run(null, false)

//Fail if getRoomDetails TestCase fail

log.info(testcaserunner.status)

assert testcaserunner.status == Status.FINISHED

The run(stringToObjectMap properties, boolean async)

**soapUI ModelItems**

* ModelItems are the preliminary building blocks of a soapUI project. The elements such as projects, test suites, test cases, test steps, mock services, mock responses, and assertions are all implemented as ModelItems.
* When you get hold of a ModelItem in your script, you can use the corresponding getters to retrieve values such as id, name, and description of a ModelItem.

TestCase.name

Let's look at some methods in TestCase ModelItem that are frequently used to retrieve TestSteps in a TestCase:

**def testCase = testRunner.testCase.testSuite.project**

**.testSuites["TestSuite 1"]**

**.testCases["TestCase 2"]**

**//To get specific test step**

**testStep = testCase.getTestStepByName("SOAP Request")**

**log.info("Name of Test Step: "+testStep.getLabel())**

**//To get test step count**

**log.info("Number of TestStep in TestCase: "+testCase.getTestStepCount())**

**//To get all test steps**

**for(testStep in testCase.getTestStepList()){**

**log.info("Name of the TestStep in TestCase: "+testStep.getName())**

**}**

• getTestStepByName(String stepName): To retrieve a specific test step inside a TestCase

• getTestStepCount(): To get the TestStep count of a TestCase

• getTestStepList(): To get list of TestSteps included in a TestCase

**Request and response handling using**

**Scripts**

soapUI provides us with a few important APIs to use with request and response messages:

• com.eviware.soapui.support.**GroovyUtils**: The API documentation of the GroovyUtils class (http://www.soapui.org/apidocs/com/eviware/soapui/support/GroovyUtils.html) provides us with all the necessary

information to use this API.

• com.eviware.soapui.support.**XmlHolder**: This is a very useful API to act upon XML request and response messages. More details about the API can be found at the official API documentation (<http://www.soapui.org/apidocs/com/eviware/soapui/support/XmlHolder.html>)

• com.eviware.soapui.model.iface.**MessageExchange**: This interface represents an exchange of request and response messages using various API methods. For more details, visit ( http://www.soapui.org/apidocs/com/eviware/soapui/model/iface/MessageExchange.html ) .

Let's find out the basic usage of each of these classes using our sample project.

1. Add a new Groovy Script TestStep under the TestCase of a **TestSuite**.
2. Add the following script:

def xmlHolder = new com.eviware.soapui.support.XmlHolder(context,"TestStep#Response")

log.info xmlHolder.getXml()

Here, we created a new XmlHolder object which makes use context variable and response of TestStep request through property expansion. Submit the TestStep once so that we will have a valid response in context. Then, run the Groovy script which will execute the script against the last received response message. You will find the response SOAP message in the script log.

1. We can get hold of the response XML message using the GroovyUtils class as follows:

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

def xmlHolder = groovyUtils.getXmlHolder("TestStep#Response")

log.info xmlHolder.getXml()

1. Similarly, the request message can also be accessed:

def xmlHolder = new com.eviware.soapui.support.XmlHolder(context,"TestStep#Request")

log.info(xmlHolder.getXml())

**Using Script Assertions**

1. Validate the content of an HTTP Header in the response

assert messageExchange.responseHeaders["Content-Length"] != 0

1. Validate a certain response time

assert messageExchange.timeTaken < 1000

1. Validate the existence of a specific XML element in the response

import com.eviware.soapui.support.XmlHolder

// check for RequestId element in response

def holder = new XmlHolder( messageExchange.responseContentAsXml )

assert holder["//ns1:RequestId"] != null

* As the testRunner object is not available in the Script assertion, you must use another approach to get test case, test suite or project properties:

// Get a test case property

def testCaseProperty = messageExchange.modelItem.testStep.testCase.getPropertyValue( "MyProp" )

// Get a test suite property

def testSuiteProperty = messageExchange.modelItem.testStep.testCase.testSuite.getPropertyValue( "MyProp" )

// Get a project property

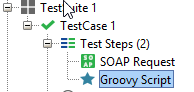
def projectProperty = messageExchange.modelItem.testStep.testCase.testSuite.project.getPropertyValue( "MyProp" )

// Get a global property

def globalProperty = com.eviware.soapui.SoapUI.globalProperties.getPropertyValue( "MyProp" )

**Parsing XML**

Created a Add SOAP Request and then added a Groovy Script



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

**def holder = groovyUtils.getXmlHolder("SOAP Request#Response")**

**def value = holder.getNodeValue("//ns1:AddResult") //giving the xPath**

**log.info(value)**

**Parsing JSON**

**Script Assertion to Read JSON Response**

**import groovy.json.JsonSlurper**

**def response = messageExchange.response.responseContent  
def slurper = new JsonSlurper()  
def json = slurper.parseText(response)**