Scripting Examples

Table of Contents

Using Variables

Using Environment, Test Suite, & Data Bank Variables

Language:

Jython

Explanation:

Python script that extracts values from various sources (data bank, test suite variable, and environment variable) for use with string operations such as concatenation using the '+' operator...

```
from soaptest.api import *
from com.parasoft.api import *

# Gets a value from a data bank (line 10) or test suite variable (line 11),
# appends an environment variable value to the front of that value,
# and returns the modified value. Be sure to comment out line 11
# when using line 10 and vice versa.

def getModifiedString(context):
#value = context.getValue("Generated Data Source", "columnName")
value = context.getValue("testSuiteVariableName")

environmentVariableValue =
context.getEnvironmentVariableValue("environmentVariableName")

modifiedString = environmentVariableValue + value

return modifiedString
```

Accessing Data Source

Language:

Jython

Explanation:

To access a data source via scripting within a method tool you can use the following script. Make sure that "python" is selected from the language drop down menu of the method tool. This script defines two methods. The first, "addDataSources" (this method must be defined with this name), returns the name of the data source and allows the method tool to use this data source. The second method, "getDataSourceValue", allows you to retrieve a value from the specified column in the data source.

```
from com.parasoft.api import *
from java.lang import *

def getDataSourceValue(input, context):
 # "value" is the value from the Data source at the specified column
 value = context.getValue("DataSourceName", "columnName")

# this method allows the Method tool to use Data Sources
def addDataSources(context):
 # DataSourceName is name of the data source
 return "DataSourceName"
```

Accessing Multiple Data Sources

Language:

Jython

Explanation:

Here is the addDataSources method for accessing multiple data sources. You still need to define the method where you actually use the data source.

```
from java.util import *

def addDataSources(context):
    ds = ArrayList()
    ds.add("DS1")
    ds.add("DS2")
    return ds
```

Using Environment Variables

Language:

Jython

Explanation:

How to use an Environment variable in your Script. In this script, there is an Environment variable named “BOB”, which has been set equal to the number “4.5”.

```
from com.parasoft.api import *

def foo(input, context):
    return context.getEnvironmentVariableValue("BOB") == "4.5"
```

Use Datasource Value & Return in XML - Jython (UG)

Language:

Jython

Explanation:

One typical usage of an Extension tool is accessing data from a data source, manipulating it, and then storing it dynamically in an XML Data Bank or Writable Data Source. In this scripting example, we define a method called getKeywords in which we are accessing data from a data source titled "Books" with a data source column titled "keywords". We then return an XML representation of this string so that we can send the output from this script to an XML Data Bank.

```
from soaptest.api import *
def getKeywords(input, context):
title = context.getValue("Books", "keywords")
return SOAPUtil.getXMLFromString([title])
```

Use Datasource Value & Return in XML - Javascript (UG)

Language:

Javascript

Explanation:

One typical usage of an Extension tool is accessing data from a data source, manipulating it, and then storing it dynamically in an XML Data Bank or Writable Data Source. In this scripting example, we define a method called getKeywords in which we are accessing data from a data source titled “Books” with a data source column titled “keywords”. We then return an XML representation of this string so that we can send the output from this script to an XML Data Bank.

```
var SOAPUtil = Packages.soaptest.api.SOAPUtil
function getKeywords(input, context) {
title = context.getValue("Books", "keywords")
return SOAPUtil.getXMLFromString([title])
}
```

Access XML Data Bank

Language:

Javascript

Explanation:

You can access values from an XML Data Bank directly with the Method Tool through the context parameter. This is done in a very similar way to accessing Data Source values, but uses a special Data Source name: "Generated Data Source". The column name is the column name in the XML Data Bank.

```
function isNotSaturday(input, context) {
    var theDate = context.getValue("Generated Data Source", "Test 1:
DateField");

    // Some code here to interpret the date string and return Y or N
}

function addDataSources(context) {
    return "Generated Data Source";
}
```

Sharing Data Between Two Extension Tools

Language:

Jython

Explanation:

To share data between Extension tools, you can use the Context.put() and Context.get() methods. For example, suppose you have two Extension Tools with the following scripts:

This works by storing the 'arr' variable in the context, which is an object passed between tools. It is saved under the object name "myArray". The second script can then retrieve it from the context using the get() method and specifying the object name. This also works when tools are chained and in scripts which are part of other tools (such as scripting hooks or Custom Assertions).

```
---[ Extension Tool 1 ]---
from com.parasoft.api import *

def storeValue(input, context):
arr = ['Blue','Cars','House']
context.put("myArray", arr)
---
---[ Extension Tool 2 ]---
from com.parasoft.api import *

def getValue(input, context):
arr = context.get("myArray")
Application.showMessage(arr[0]) # Shows Blue in the console view
---
```

Create Test Suite Variable

Language:

Jython

Explanation:

Do you want to create variables that will persist between test runs? Would you like a variable over which you have full control? Would you like to be able to save your project and have this variable be saved with it? Well, here's how to do it.The following code will create your variable and store it in the test suite context.

```
from soaptest.api import *
from com.parasoft.api import *

SUITE_VAR = "myVar"


#Select this in the Method dropdown to perform the required tasks
#It returns the variable value as XML, which can be stored in a Data Bank
def  doSomething(input, context):
   myVar = getMyVar(context)

   #Do what you like with the variable here

   myVar = str(myVar)
   return SOAPUtil.getXMLFromString([myVar])



#This function retrieves your variable, or creates it
def getMyVar(context):
   suite = getSuite(context)
   myVar = suite.get(SUITE_VAR)
   if myVar == None:
       myVar = "0"
       suite.put(SUITE_VAR, myVar)
   return int(myVar)



#This expects the Method Tool context
#and returns the Suite context
def getSuite(context):
   test = context.getParentContext()
   return test.getParentContext()
```

Test Suite Variables pt. 1

Language:

Jython

Explanation:

Now that you have a Test Suite Variable, this code will allow you to access it (assuming you have named the variable "myVar"):

Once you have accessed a Test Suite Variable, you can manipulate it with getValue() and setValue(). Note that this example is not strictly good coding practice but is intended to demonstrate what is happening.

```
#myVarTSV is a member of the TestSuiteVariable class
myVarTSV = context.getVariable("myVar")


incrementAndSet(myVarTSV):
   #getValue() always returns a string
   myVar = myVarTSV.getValue()

   #Cast to int
   myVarint = int(myVar)

   #Increment by one
   myVarint += 1

   #Cast to str
   myVar = str(myVarint)

   #Store myVar to use later in the test run
   myVarTSV.setValue(myVar)
```

Test Suite Variables pt. 2

Language:

Jython

Explanation:

The above is just basic code assuming that you want an int type. Let's say you want to increment a variable. Then you could call this in your script.

```
#Assumes aVar is an int
def incrementAndSaveVar(aVar, context):
   aVar += 1
   suite = getSuite(context)
   suite.put(SUITE_VAR, aVar)

#If you are using multiple suite variables
#you might want this instead
def incrementAndSaveVar(aVar, aSUITE_VAR, context):
   aVar += 1
   suite = getSuite(context)
   suite.put(aSUITE_VAR, aVar)
```

Test Suite Variables pt. 3

Language:

Jython

Explanation:

Perhaps you want to append a value to the end of a string. Then, call this in your script.

```
#Takes a string and an int, and appends
#the value of aVar
def getModifiedString(myString, aVar):
   modifiedString = myString + str(aVar)
   return modifiedString
```

Test Suite Variables pt. 4

Language:

Jython

Explanation:

Finally, let's say that you have made a few runs, and you would like to reset the variable back to *None*. Then you can select this in the Method dropdown and run the test.

```
#For one suite variable
def resetVar(input, context):
   suite = getSuite(context)
   suite.put(SUITE_VAR, None)



#For many suite variables
SUITE_VARS = ["VarA", "VarB", "VarC"]


def resetAllVars(input, context):
   suite = getSuite(context)
   numSuiteVars = len(SUITE_VARS)
   for i in xrange(numSuiteVars):
       suite.put(SUITE_VARS[i], None)
```

Change Test Suite Variable By Script

Language:

Jython

Explanation:

Sets the value of a test suite variable. does not need to be chained to anything. Test Suite Variable was predefined as “x” with value of 0.

```
def modifyVariable(x, context):
    context.setValue("x", "3");
```

Browser Testing Scripts

Accessing DOM of Browser Testing Tool

Language:

Javascript

Explanation:

These examples are intended for use when chaining an Extension tool to a Browser Testing Tool,used in a Web Functional Test.

getDocument is overloaded as follows:

```
getDocument(); // gets the document for the main window
getDocument(String windowName); // gets the document for the window with the
specified window name
getDocument(String windowName, String frameName); // gets the document for a
frame within a window
```

Accessing DOM of Browser - Retrieve <title> text of Frame

Language:

Javascript

Explanation:

Continued from above. For example, the following code gets the content of a frame titled "mainPane" in the main window. It then uses the document to get the title text for the selected frame.

```
var Application = Packages.com.parasoft.api.Application;
var WebBrowserUtil = Packages.webking.api.browser2.WebBrowserUtil;

function getFrameDocument(input, context) {
   var document = input.getDocument("", "mainPane");
   var titles = document.getElementsByTagName("title");
   var title = titles.item(0);
   var value = WebBrowserUtil.getTrimmedText(title);
   Application.showMessage("title: " + value);
   return value;
}
```

Return Title from Frame in Browser

Language:

Jython

Explanation:

```
The getDocument functional can also take one or two arguments:

getDocument(windowName)
getDocument(windowName, frameName)

For example, the following code gets the frame whose name is "mainPane"
from an unnamed window and returns the title for that frame:
```

```
//---/BEGIN CODE/---
var Application = Packages.com.parasoft.api.Application;
var WebBrowserUtil = Packages.webking.api.browser2.WebBrowserUtil;

function getFrameDocument(input, context) {
var document = input.getDocument("", "mainPane");
var titles = document.getElementsByTagName("title");
var title = titles.item(0);
var value = WebBrowserUtil.getTrimmedText(title);
Application.showMessage("title: " + value);
return value;
}
//---/END CODE/---
```

Get HTML Contents from Frame

Language:

Jython

Explanation:

You may also be interested in the example below, which allows you to get the HTML contents of any of the frames as a string. Please note that, unlike the example above, this example uses undocumented code which is subject to change. SOAtest 6.2 provides a much simpler and documented "getHTML" API to accomplish the same effect.

Modify INDEX_OF_FRAME as necessary to indicate the index of the frame whose HTML you require.

```
//---/BEGIN CODE/---
var BrowserContents = Packages.webtool.browsercontroller.BrowserContents;

// code assumes there are frames in the window
function returnHTML(input, context) {
var contents = input.getBrowserContents();
var window = contents.getTopWindow(0);
var childFrames = window.getChildren();
// the first frame is at index 0
var INDEX_OF_FRAME = 0;
var frame;
for (var i = 0; i <= INDEX_OF_FRAME && childFrames.hasMoreElements(); ++i) {
frame = childFrames.nextElement();
}
var html = BrowserContents.getHTML(frame);
return html;
}
//---/END CODE/---
```

Access HTML Content as String (UG)

Language:

Javascript

Explanation:

If you want to access the HTML content as a string (e.g., if you are working with a text document and you want to avoid having to predict what HTML markup the browser will add), you can add an Extension tool to the browser contents of the browser test. To do this, you would (right-click on your browser test, then choose Add Output> Browser Contents (rendered HTML)> Extension Tool. Next, you configure the Extension tool to use an appropriate script.

You can retrieve the HTML for a browser window or frame using input.getHTML(). See the Javadoc for com.parasoft.api.BrowserContentsInput. The Javadocs can be accessed by choosing **Parasoft> Help> Extensibility API** . For example, here is a JavaScript that searches for an RFC title.

```
// input: com.parasoft.api.BrowserContentsInput.
// context: com.parasoft.api.ExtensionToolContext.

function validateRfcText(input, context) {

  var html = input.getHTML();
  var rfc = context.getValue("ds", "rfc");

  // Extract the numeric part of the RFC.
  // From "RFC5280" extract "5280".
  // From search("\\d”) returns the index of the first digit in rfc.
  // See a reference on JavaScript regular expressions.
  // Alternatively hard-code rfc.substring(3),

  var rfcNumber = rfc.substring(rfc.search("\\d"));
  var title = "Request for Comments: " + rfcNumber;

  if (html.indexOf(title) < 0) {
    context.report("HTML does not contain title: " + title);
  }
}
```

Create Xpath Hook – Browser Testing (UG)

Language:

Javascript

Explanation:

For example, if you want to identify elements by their "class" attribute, you could use the CreateXPath hook to create a custom identifier that checks elements for a "class" attribute and, if it finds one, returns an XPath that identifies the element by the value of the "class" attribute.

When selecting an element to validate, or clicking an element during recording, SOAtest would then use this custom hook. The element would be passed to the custom hook, which would then check the element for a "class" attribute. If it found one, it would return an XPath using the "class" attribute value to identify the element.Here is a a sample script that identifies elements using the "class" attribute and an index (when more than one node has the same "class").

```
app = Packages.com.parasoft.api.Application;
settings = Packages.webtool.browsertest.XPathCreator;
// establish the locator builder priority order
// we use our custom locator after using the id, name and attribute locators
// and before using the position xpath locator
settings.locatorBuildersOrder[0] = settings.ID_LOCATOR;
settings.locatorBuildersOrder[1] = settings.NAME_LOCATOR;
settings.locatorBuildersOrder[2] = settings.ATTRIBUTE_XPATH_LOCATOR;
settings.locatorBuildersOrder[3] = settings.HOOK_LOCATOR;
settings.locatorBuildersOrder[4] = settings.POSITION_XPATH_LOCATOR;
// gets the index of the node out of a list of nodes returned using xpath.
function getIndex(xpath, node, doc) {
index = -1;
if (doc) {
try {
list = Packages.org.apache.xpath.XPathAPI.selectNodeList(doc, xpath);
count = 0;
while (index == -1 && count < list.getLength()) {
candidate = list.item(count);
if (candidate.equals(node)) {
index = count;
} else {
++count;
}
}
} catch (e) {}
}
return index;
}
// the hook function to link to the CreateXPath hook.
// looks for the class attribute and uses it (along with an index) to create
// a locator for the given node.
function createXPathHook(node, doc) {
nodeName = node.getNodeName();
attributes = node.getAttributes();
classValue = attributes.getNamedItem("class");
if (classValue) {
xpath = "//" + nodeName + "[@"+ classValue + "]";
index = getIndex(xpath, node, doc);
if (index != -1) {
return xpath + "[" + index + "]";
}
}
return null;
}
// sets up the hook
function setHook() {
hook = app.getHook("CreateXPath");
hook.set(createXPathHook);
}
```

Using an XPath with a parameterized value (UG)

Language:

Javascript

Explanation:

In SOAtest you can use a parameterized value, which is a value from a data source, test logic variable, or from an extraction. To create an XPath expression that references a parameterized value, you will need to create a scripted element locator. Suppose that you want to click on the "Details" link based on a "number" value extracted in a previous test. For example, maybe the row contains a new number generated by your previous user actions.

You can extract the new number from a previous browser test by using a Browser Data Bank. To create a browser test that clicks on the link specified by that generated number, use configure the browser test to use a scripted "Element Locator" value in the "User Action" tab. Then you can use a JavaScript script such as the following:

```
var Application = Packages.com.parasoft.api.Application;
// input: com.parasoft.api.BrowserContentsInput.
// context: webking.api.BrowserTestContext.
function scriptedXpath(input, context) {
// Assumes that you configured the extraction to use column name
"extractedNumber".
// Use data source name "", null, or "Generated Data Source" because the
// value is from an extraction, not a data source (table, CSV, etc.).
var number = context.getValue("", "extractedNumber");
var xpath = "//tbody[@id='content']/tr[td[1] = '" + number + "']/descendant::
a[text()='Details']";
// Output XPath for debugging purposes.
Application.showMessage("scripted XPath: " + xpath);
return xpath;
}
```

Scripting Element Locators (UG)

Language:

Jython

Explanation:

If you need to use complicated logic to find an element, a single XPath expression may become unwieldy. Note that when using a scripted element locator, you can return either an XPath string or an org.w3c.dom.Node object. As mentioned, IE can be slow to evaluate complex XPath expressions; using a scripted locator that returns a node may be faster. A script to find the "Details" link in the last row:

```
var WebBrowserTableUtil = Packages.webking.api.browser2.WebBrowserTableUtil;
// input: com.parasoft.api.BrowserContentsInput.
// context: webking.api.BrowserTestContext.
function scriptedNode(input, context) {
var document = input.getDocument();
var tbody = document.getElementById("content");
// Assumes that there are no rows within rows: this returns
// all descendants with the given tag, not just children.
var rows = tbody.getElementsByTagName("tr");
var lastRow = rows.item(rows.getLength() - 1);
var detailsColumnIndex = 2; // Zero-based.
var detailsLink = WebBrowserTableUtil.getLinkFromCell(
detailsColumnIndex, lastRow);
return detailsLink;
}
```

How to invoke javascript in the browser context

Language:

Javascript

Explanation:

In general we cannot invoke javascript which is running or being called from the web page in the browser because we don't have an official means of running javascript in the browser context. For example, if your web page hasa javascript method tryThis(), and you wanted to invoke this by trying to use an extension tool or some such means to add javascript which has a line that calls tryThis(), it would not work because SOAtest's JavaScript executes in a self-contained context, not within the context of the browser or the browser's objects. The benefit of this is that we can invoke the java based extensibility api, the downside is as has been mentioned, that it cannot invoke methods in the browser context like tryThis().

However, there is one case where we can actually invoke methods in the browser context, and that's in the element locator of the user action. If you look at the file i included, test 2 is a dummy click user action, while we perform this dummy click action, we wish to also invoke the tryThis()method.

To invoke the tryThis() method, under the "User Action" tab, the "Element Locator" must be set to "Use XPath" and we can use the following values:

(in Firefox)
```
document.defaultView.tryThis();document.getElementById("b2")
```

(in Internet Explorer)
```
document.parentWindow.tryThis();document.getElementById("b2")
```

SOAtest will evaluate the JavaScript on the current document. The locator must begin with "document." as that string notifies SOAtest that the user would like to evaluate a JavaScript statement on the document object. To use the same locator for both browsers, the user must use the value:

```
document.defaultView.tryThis();document.getElementById("b2")
```

AND modify UserCustomizableOptions.js in
<SOAtest Installation Directory>/eclipse/plugins/com.parasoft.xtest.libs.web_<version>/root/browsers/ie
to include the line:

```
document.defaultView = document.parentWindow;
```

This defines the "defaultView" variable in IE, allowing it to be used as part of the locator for either browser. The second statement: document.getElementById("b2") must be a DOM handle to a real element on the page, otherwise the test will fail saying that it cannot find the element, so the user will have to change this to an element on their page. On my sample age that i provided, there is a "Button 2" with id "b2", which does nothing when it is clicked, ie its a dummy click. Let me know if you have any further questions.

Browser Validation Scripts

Complex Browser Testing Validations (UG)

Language:

Javascript

Explanation:

If you want to perform complex validations that cannot be properly representing using the GUI controls, you can express them using scripting. For example, suppose that you want to validate all of the rows in a table. That table could be of variable length. You can attach an Extension tool to a browser functional test and pull values from the document provided by input.getDocument(). Here's a sample JavaScript script that accomplishes that

```
var Application = Packages.com.parasoft.api.Application;
var WebBrowserTableUtil = Packages.webking.api.browser2.WebBrowserTableUtil;
var WebKingUtil = Packages.webking.api.WebKingUtil;

// Verify that all values in a table column are equal to a previously
// extracted value. For example, we searched for all places in which
// widget 11 is sold, and we want to make sure that all results are
// for widget 11.
// input: com.parasoft.api.BrowserContentsInput.
// context: com.parasoft.api.ExtensionToolContext.

function validateTable(input, context) {

  // For the column we want to validate.
  var widgetColumnIndex = 0;
  // We extracted through a Browser Data Bank in a previous test
  // the expected value to data bank column "widgetName".
  // The value was extracted, not from a data source, so use "" or
  // null (None in Python) as the name of the data source.

  var expectedWidgetName = context.getValue("", "widgetName");
  var document = input.getDocument();
  // Table should have some unique identifying attribute (e.g., id).
  var table = WebBrowserTableUtil.getTable("id", "mytable", document);
  // If the first row of the table contained column headers, we could
  // use getCellValuesForColumn(String, Element). For example if the
  // widget name column was named "Widget Name", then we could use
  // getCellValuesForColumn("Widget Name", table). In either case,
  // getCellValuesForColumn returns an array of String objects. See
  // the JavaDocs for more information.

  var values = WebBrowserTableUtil.getCellValuesForColumn(
                                            widgetColumnIndex,   table);
  if (values.length == 0) {

    context.report("No rows found!");
  }

  for (var i = 0; i < values.length; ++i) {

    if (values[i] != expectedWidgetName) {

      var errorMessage = "Widget name (column " + widgetColumnIndex + "): "
       + "Expected value '" + expectedValue
       + "', but found '" + values[i] + "'.";
       context.report(errorMessage);
     }
  }
}
```

Validating a “Checked” property of HTML check box

Language:

Jython

Explanation:

jons note: replace WebKingUtil.getDocument with input.getDocument, the other is deprecated

The checked property is separate from the value attribute in that it exists primarily as a DOM property of the checkbox. As such it usually isn't specified in HTML code except to set default values for checkboxes in which case the HTML usually will show 'checked' in the checkbox definition. For example, it may be something like:

<INPUT class="" id="radio1" name="MyRadioButton" type="checkbox" value="RandomValue" checked>

The value attribute is the value that will be sent to the server if the checkbox is checked. Also, it is possible to parse the HTML and read the value via a script. Note that you will need WebKing 6.0.5 to do this from a Method Tool outside of a Browser Functional Test. With WebKing 6.0.5 you can get access to the DOM from any Method Tool with a call to WebKingUtil.getDocument() (please refer to Help->Scripting API->webking.api->WebKingUtil for more information):

```
ex.
function processDocument(input,context) {
var document = Packages.webking.api.WebKingUtil.getDocument(context);
// parse document to get checkbox
// get the value of the checked property of the checkbox
}
```

For given HTML, such as:

```
<HTML>
   <HEAD>
   </HEAD>
   <BODY>
     <INPUT checked="checked" id="radio1" name="radio" type="radio" value="FirstRadio"/>
     <INPUT checked="false" id="radio2" name="radio" type="radio" value="SecondRadio"/>
   </BODY>
```

```
</HTML>
```

When calling this getDocument() method you are returned an org.apache.html.dom.HTMLDocumentImpl object. While similar to the javascript DOM library used when developing web applications, it is not the same. You will need to use the methods that are available through this DOM to find your checkbox and validate its checked property. You can try using this template script to validate check/uncheck property.

```
var webKingUtil = Packages.webking.api.WebKingUtil;
var app = Packages.com.parasoft.api.Application;

function radiosChecked(input, context)
{

  var doc = webKingUtil.getDocument(context);
  var radios = doc.getElementsByName("radio");

  // This will return (checked="checked")
  app.showMessage("first radio checked: " +
                 radios.item(0).getAttributes().getNamedItem("checked"));

  // This will return (checked="false")
  app.showMessage("second radio checked: " +
                 radios.item(1).getAttributes(). getNamedItem("checked"));

}
```

Return Actual Value for Use in Scripted Browser Validation

Language:

Jython

Explanation:

I had a streaming session with user. she explained me her test scenario and wanted to verify whether data on page is either "KRK,DFW" or "DFW,KRK". We added validation tool and wrote a small script to do so.

note on String getActualValue() :
This function is specifically and only for use within the scripted option of a validation in extractions. This returns the actual value of the current extraction. This provides the ability to test the actual value of an element against the expected value within a script option of a validation.

```
from com.parasoft.api import Application

def validate(input, context):

  value = context.getActualValue();
  Application.showMessage(str(value))

  if (value == "KRK,DFW" or value == "DFW,KRK"):
     return value;
  else:
     return "";
```

Scanning Tool

Scanning Hook - Alert (UG)

Language:

Jython

Explanation:

When SOAtest encounters a JavaScript alert() method, its default behavior is to click the **OK** button and print JavaScript Alert: "message" to the SOAtest Console view. You can modify this behavior using the Alert hook. By default, the Alert hook is called whenever SOAtest encounters a JavaScript alert. By adding methods to this hook, you determine how SOAtest behaves when it encounters a JavaScript alert. When the Alert hook is invoked, SOAtest will print the message JavaScript Alert: "message" to the location specified in the script, or to the SOAtest Console view (if no alternate location is specified).

This hook is commonly used to print the results of alert messages to a special SOAtest Message window. For example, if you wanted SOAtest to report all alert messages in a SOAtest Message window named "Alert Messages," you could create the following Jython script and add it to your <soatest_install_dir>/plugins/com.parasoft.xtest.libs.web_<soatest_version>/root/ startup directory: The argument passed into the method myAlertHook will be the alert message. It is a string that is the message the alert contains.

```
from com.parasoft.api import Application
def myAlertHook(msg):
# Print the alert message to SOAtest Messages
Application.showMessage(" JavaScript Alert: %s" % str(msg))
# Add the contents of the alert message to a Result Window named
# "Alert Messages". This ResultWindow can later be used
# as part of a test suite in a regression test, to make sure the
# contents of the window, which contains the alert message,
# are what you expect them to be.
Application.report("Alert Messages", str(msg))
# Add your method to the Alert Hook
Application.getHook("Alert").set(myAlertHook)
```

Scanning Hook – Alert 2 (UG)

Language:

Jython

Explanation:

You could also use the Alert hook to verify that when a user (or SOAtest) submits certain invalid form values, the application opens an alert box warning that invalid data was entered. To implement this test, you would first create and invoke a script (like the one above) that records alert messages into a SOAtest Message window. Then, you would implement a test suite that checks the contents of the Message window using the following test cases:

1. An Extension tool that clears the Message window.
2. A Test Path tool that executes the click path that should cause the alert message.
3. An Extension tool with a script (like the one shown below) that returns the text of the Message window, with an attached regression control that verifies that the correct alert message appears in the Message window.

```
#Script to return text of result window named "Alert Messages"
from soaptest.api import *
def return():
return SOAPUtil.getResultWindowText("Alert Messages")
#Script to clear text of result window named "Alert Messages"
from soaptest.api import *
def clear():
SOAPUtil.clearResultWindow("Alert Messages")
```

Scanning Hook – Confirm and Prompt (UG)

Language:

Jython

Explanation:

By customizing the Confirm hook, you determine how SOAtest behaves when it encounters the confirm() method. For example, one way to prompt SOAtest to return a customized (non-default) response whenever it encounters the confirm() method is to perform the following steps:

First, create a Jython file (named startup.py) that defines a method which takes a single argument. This argument will have the same value that gets passed to the confirm() method in your JavaScript to this argument. Based on whatever logic you need, determine whether you want confirm() to return "true" or "false", and then return that value from the method you defined. To have SOAtest use the method you just defined, add it to the Confirm hook in the following way, where the argument you pass to set is the name of the method that you defined. In the example, the method defined is named myConfirmHook.

```
from com.parasoft.api import Application
# msg will have the same value that gets passed to confirm()
def myConfirmHook(msg):
if msg == "Yes or no?":
return 1
return 0
Application.getHook("Confirm").set(myConfirmHook)
```

Scanning Hook – Realm Password (UG)

Language:

Jython

Explanation:

The Realm Password hook is called when *all* of the following conditions are satisfied:

- SOAtest makes a request of a server.
- The server responds with a challenge, asking for either a realm password or an NTLM password (both of which SOAtest handles in the same way).
- You have not yet entered a realm or NTLM password.

If you have defined a custom Realm Password hook before the above conditions are satisfied, SOAtest will execute the custom hook method before prompting you to manually enter a password. If that method adds a password (either to a site in the project, or to a load test virtual user), SOAtest will use that password, and will not prompt you for a password. If you or another SOAtest user previously entered a password, this hook will not get executed. If you want to ensure that SOAtest uses the password you defined for the hook, you should clear the site passwords before you want SOAtest to add new passwords from the hook.

This hook is particularly useful in cases where you want to vary usernames/passwords depending on the particular context you are in (e.g., based on the particular path you are using, or you would like to use different usernames/passwords in the context of a virtual user load test). The following sample Java file is a very basic sample implementation of the Realm Password hook. This script adds the same password regardless of the context.

```
import com.parasoft.api.*;
import java.lang.reflect.*;
import java.net.*;
import soaptest.api.*;
public class RealmPasswordHook {
static public void returnPassword(String realm, String url, boolean ntlm,
Context context)
throws MalformedURLException {
SOAPUtil.addSitePassword(realm, "user1", "password1", url, ntlm);
}
public void setHook() throws UserMethodException, NoSuchMethodException
{
Hook hook = Application.getHook("RealmPassword");
Method userMethod = getClass().getMethod("returnPassword", new Class[]
{String.class,
String.class, boolean.class, Context.class});
hook.set(userMethod);
}
}
```

Scanning Hook – URL Logging (UG)

Language:

Jython

Explanation:

The URL Logging hook is called every time SOAtest visits a URL. It can be used to prompt SOAtest to print each URL visited. The methods you attach to this hook need to have either 3 or 4 arguments. The first argument should be a string specifying the URL visited. The second argument should be any POST data submitted with the URL (this mat be NULL). The third argument is a java.util.Hashtable containing other HTTP properties sent with the request, (e.g., referer). The optional fourth argument, if added to the method signature, is a com.parasoft.api.Context.

In addition, you need to set SiteUtil.enableLogging to true and pass UrlLogging to Application. getHook() For example, if you wanted SOAtest to print each URL visited to the Console view, you could create the following Jython script and add it to your <soatest_install_dir>/plugins/com.parasoft. xtest.libs.web_<soatest_version>/root/startup directory:

```
from com.parasoft.api import *
from webtool.site import SiteUtil
count = 1
def setHook():
SiteUtil.enableLogging = 1
hook = Application.getHook("UrlLogging")
hook.set(loggingHook)
def loggingHook(url, post, props):
global count
if post:
Application.showMessage(str(count) + ". " + url + " [POST=" + post + "]")
else:
Application.showMessage(str(count) + ". " + url)
count = count + 1
setHook()
```

Reading/Writing to Files

Writing Elements to a CSV/Excel File W/ a Timestamp

Language:

Jython

Explanation:

Writing specific values to a csv/excel format. This example uses the add operation, where we want to write the two operands, as well as the result of the add operation to a csv file format for easy comparison. This example uses a writable datasource(which uses a databank), the test structure should look like this:

Data source

- table - addition operands (for data driving add operation)

- writable datasource

SOAP Client - Add operation

- Request - databank(capture x and y operands)

- Response - databank (caputre response)

Extension Tool – With the following script:

```
from com.parasoft.api import *
from sys import *
from java.util import *
from java.text import *

def appendToFile(input, context):
#Create a timesatmp on the filename
 today = Date()
 cal = GregorianCalendar()
 todayPlus3 = cal.getTime()
 parser = SimpleDateFormat("HH_mm")
 dateField = parser.format(todayPlus3)
 Application.showMessage(str(dateField))
 #retrieve values from the datasource
 input_one = context.getValue("Writable","A")
 input_two = context.getValue("Writable","B")
 input_three = context.getValue("Writable","C")

 #opens files, creates one if it doesn't exists
 f = open('C:\Users\Jon\Calculator_'+str(dateField)+'.csv', 'a')
 f.write(str(input_one) + ',' + str(input_two) + ',' + str(input_three))
 f.write('\n')
 f.close()
 return

def addDataSources(context):
  return "Writable"
```

Read Contents of File into String

Language:

Jython

Explanation:

To read the contents of a file into a String using a Jython script you can use the following method:

```
from java.lang import *
from java.io import *
from soaptest.api import *

def getContents(input, context):
  contents = StringBuffer()
  reader = BufferedReader(FileReader(File("c:\Documents and 
Settings\jhendrick\Desktop\test.txt")))
  line = String()
  while line != None:
    line = reader.readLine()
    if line != None:
      contents.append(line)
    contents.append(System.getProperty("line.separator"))
  reader.close()
  return contents.toString()
```

Log/Write Info (xml data bank) to file(old)

Language:

Jython

Explanation:

To log information to a file you can use the following script. This example logs a value from a Generated Data Source, such as the XML Data Bank. In this example we are retreiving a value that represents a price and are choosing to write it to a file if the price is greater than 50.

```
from com.parasoft.api import *
from java.util import *
from java.io import *

def logPrice(input, context):
   price = context.getValue("Generated Data Source", "Test 1: price")
   if output != None:
       if price > 50:
           home = "c:\\home\\"
           file = File(home, "priceLog.txt")
           # 0 means over-write, 1 means to append
           writer = FileWriter(file, 0)
           writer.write(Date().toString() + " $" + str(price) + "\n\r")
           writer.close()

def addDataSources(input, context):
   return "Generated Data Source"
```

Writing information to a file

Language:

Javascript

Explanation:

Writing information to a file

```
from com.parasoft.api import *
from java.util import *
from java.io import *

def logPrice(input, context):
   price = context.getValue("Generated Data Source", "Test 1: price")
   if output != None:
       if price > 50:
           home = "c:\\home\\"
           file = File(home, "priceLog.txt")
           # 0 means over-write, 1 means to append
           writer = FileWriter(file, 0)
           writer.write(Date().toString() + " $" + str(price) + "\n\r")
           writer.close()

def addDataSources(input, context):
   return "
```

Customize File names with datasource values

Language:

Jython

Explanation:

To customize the name of a file with values from a Datasource via a Method Tool, the following script can be used. Make sure that "python" is selected from the language drop down menu of the method tool. "DatasourceName" is the name of the datasource and "DatasourceColumnName" is the name of the datasource column from which you want to get values. "C:\dir\fileName" + str(value) + ".txt" should be replaced by your path, file name, and extension - "+ str(value) +" adds the datasource value to the file name.

```
from com.parasoft.api import *
from java.util import *
from java.io import *
from java.lang import *

def writeFile(input,context):
   # "value" is the value from the Data source at the specified column
   value = context.getValue("DatasourceName", "DatasourceColumnName")
   # adds the Data source value to the file name, casts value to a string
   ostr = FileOutputStream("C:\\dir\\fileName" + str(value) + ".txt")
   # adds file output stream to a buffer
   bstr = BufferedOutputStream( ostr )
   # writes the file
   bstr.write(input, 0, String(input).length())
   # closes BufferedOutputStream
   bstr.close()
```

How to read the contents of a file into a string

Language:

Jython

Explanation:

How to read the contents of a file into a String

```
from java.lang import *
from java.io import *
from soaptest.api import *

def getContents(input, context):
  contents = StringBuffer()
  reader = BufferedReader(FileReader(File("c:\Documents and
Settings\jhendrick\Desktop\test.txt")))
  line = String()
  while line != None:
    line = reader.readLine()
    if line != None:
      contents.append(line)
    contents.append(System.getProperty("line.separator"))
  reader.close()
  return contents.toString()
```

Reading Text using IOUtil.readTextFile(File)

Language:

Jython

Explanation:

This script reads the text_unix.data file and does, saves it, reads it, then verifies the contents. The 3 definitions represent the contents of 3 separate extension tools within a test suite. The contents of text_unix.data is:

```
c 1
c 2
```

```
from com.parasoft.api import Application

def setInputFile(input, context):
    file = context.getAbsolutePathFile("text_unix.data")
    if not file.exists():
        context.report("File does not exist: " + file.getAbsolutePath())
    context.put("inputFile", file)


#This script reads it using IOUtil.readTextFile

from com.parasoft.api import IOUtil

def readTextFile(input, context):
    inputFile = context.get("inputFile")
    contents = IOUtil.readTextFile(inputFile)
    # Store in context instead of return value because return value
    # is transformed into TextUsable.
    context.put("textFileContents", contents)

#this script validates text contents:

def validateTextFile(input, context):
    textFileContents = context.get("textFileContents")
    expectedContents = "c 1\nc 2\n\n"
    if textFileContents != expectedContents:
        context.report("Wrong text file contents: "
            + "expected '" + expectedContents
            + "' but got '" + textFileContents + "'")
```

Reading Binary file using IOUtil.readBinaryFile(File)

Language:

Jython

Explanation:

This script reads the text_unix.data file and does, saves it, reads it, then verifies the contents. The 3 definitions represent the contents of 3 separate extension tools within a test suite. The contents of text_unix.data is:

```
c 1
c 2
```

```
from com.parasoft.api import Application

def setInputFile(input, context):
    file = context.getAbsolutePathFile("text_unix.data")
    if not file.exists():
        context.report("File does not exist: " + file.getAbsolutePath())
    context.put("inputFile", file)

#This script reads it using IOUtil.readBinaryFile

from com.parasoft.api import IOUtil

def readBinaryFile(input, context):
    inputFile = context.get("inputFile")
    contents = IOUtil.readBinaryFile(inputFile)
    # Store in context instead of return value because return value
    # is transformed into TextUsable.
    context.put("binaryFileContents", contents)

#this script validates binary contents:

import jarray

def validateReadBinaryFile(input, context):
    binaryFileContents = context.get("binaryFileContents")
    expectedContents = jarray.array([0x63, 0x20, 0x31, 0x0a, 0x63, 0x20,
0x32, 0x0a, 0x0a], "b")
    if expectedContents != binaryFileContents:
        context.report("Wrong binary file contents: "
            + "expected " + str(expectedContents)
            + " but got " + str(binaryFileContents))
```

Load Test Scripts

Load Test Stop Script – Print to Console Merged data

Language:

Jython

Explanation:

Load Tester stop actions have been introduced. These stop actions allow you to stop the load tester from executing if a particular condition or state is reached. Some sample scripts that can be used are shown below:

```
from java.lang import *
from java.util import *
from com.parasoft.api.loadtest import *
from com.parasoft.api.loadtest.output import *

#
# Print to console available merged data
#
def printMergedData(args):
 print '************************'
 resultSet = args.getOutput().getMergedOutput()
 resultSet.resetRowIterator()
 while resultSet.hasNextRow():
   row = resultSet.nextRow()
   entries = row.entrySet()
   iter = entries.iterator()
   print '    -------------'
   while iter.hasNext():
     entry = iter.next()
     print '    ', entry.getKey(), entry.getValue()
```

Load Test Stop Script – Print to console current data

Language:

Jython

Explanation:

Load Tester stop actions have been introduced. These stop actions allow you to stop the load tester from executing if a particular condition or state is reached. Some sample scripts that can be used are shown below:

```
#
# Print to console available current data
#
def printCurrData(args):
 print '************************'
 iter = args.getOutput().getCurrentOutput().entrySet().iterator()
 while iter.hasNext():
   entry = iter.next()
   print '    ', entry.getKey(), entry.getValue()
```

Load Test Stop Script – Stop if # of errors exceed N

Language:

Jython

Explanation:

Load Tester stop actions have been introduced. These stop actions allow you to stop the load tester from executing if a particular condition or state is reached. Some sample scripts that can be used are shown below:

```
#
# Stop if total number of errors exceeds N
#
def stopAfterNErrors(args):
 ERROR_THRESHOLD = 200
 STOP_DESC = 'Number of errors exceeded 200.'
 output = args.getOutput()
 resultSet = output.getMergedOutput()
 resultSet.resetRowIterator()
 failures = 0
 while resultSet.hasNextRow():
   row = resultSet.nextRow()
   failures += row.get(MergedOutputConstants.FAILURE_COUNT)
   if failures > ERROR_THRESHOLD:
     return LoadTestScriptAction(LoadTestScriptAction.ACTION_STOP,STOP_DESC)
```

Load Test Stop Script – Stop if N is above threshold for M measurements

Language:

Jython

Explanation:

Load Tester stop actions have been introduced. These stop actions allow you to stop the load tester from executing if a particular condition or state is reached. Some sample scripts that can be used are shown below:

```
#
# Stop if parameter N is above threshold T for M consecutive measurements
#
def stopWhenCPUHigh(args,context):
 #-- Stop if CPU utilization is greater than 10% for 5 consecutive
invocations
 #-- Modify theese numbers according to your needs:
 PARAM_THRESHOLD = 10
 CONSECUTIVE_LIMIT = 5
 PARAM_NAME = 'cheetah/CPU'
 STOP_DESC = 'CPU utilization greater than 10% for 5 consecutive invocations'
 #-- Store the consecutive count in context between invocations
 contextValueID = 'COUNT'
 count = context.get(contextValueID)
 if count is None:
   count = 0
   context.put(contextValueID,count)
 paramValue = args.getOutput().getCurrentOutput().get(PARAM_NAME)
 print PARAM_NAME,'=',paramValue
 if paramValue > PARAM_THRESHOLD:
   count = count + 1
 else:
   count = 0
 context.put(contextValueID,count)
 if count >= CONSECUTIVE_LIMIT:
   return LoadTestScriptAction(LoadTestScriptAction.ACTION_STOP,STOP_DESC)

#
# Do nothing
#
def doNothing(allData):
 return 0
```

Simple Monitor which Returns Memory Metrics in Current JVM Environment

Language:

Java

Explanation:

In this example, the method makes a call to the java.lang.Runtime object to get the current memory status of the JVM. The idea is to create methods which return a numerical type (int, long, etc) which the Load Tester will then call and plot in a graph. If you use a Java class, the easiest way to deploy it is to wrap it into a jar file and put the jar into your SOAPtest installation directory. If you use Python, you can input the code directly into the custom monitor. To add the monitor, go to the Load Test configuration panel, right-click on Monitors, and select New Monitor > Custom. Give the monitor a name, click New and either input the Python code or type the name of the Java class. You will then be able to select one of the defined methods as an input to the monitor.

```java
public class JVMMonitor {
  /**
   * @return the total amount of memory in the JVM (in MB)
   */
  public static long getTotalMemory() {
      return Runtime.getRuntime().totalMemory() / (1024 * 1024);
  }
  /**
   * @return the amount of free memory in the JVM (in MB)
   */
  public static long getFreeMemory() {
      return Runtime.getRuntime().freeMemory() / (1024 * 1024);
  }
  /**
   * @return the amount of memory current used by the JVM (in MB)
   */
  public static long getUsedMemory() {
      return getTotalMemory() - getFreeMemory();
  }
}
```

Returning memory metrics in the current JVM

Language:

Jython

Explanation:

In this example, the method makes a call to the java.lang.Runtime object to get the current memory status of the JVM. The idea is to create methods which return a numerical type (int, long, etc) which the Load Tester will then call and plot in a graph. If you use a Java class, the easiest way to deploy it is to wrap it into a jar file and put the jar into your SOAPtest installation directory. If you use Python, you can input the code directly into the custom monitor. To add the monitor, go to the Load Test configuration panel, right-click on Monitors, and select New Monitor > Custom. Give the monitor a name, click New and either input the Python code or type the name of the Java class. You will then be able to select one of the defined methods as an input to the monitor.

```
from java.lang import Runtime

def getTotalMemory():
  return Runtime.getRuntime().totalMemory() / (1024 * 1024)

def getFreeMemory():
  return Runtime.getRuntime().freeMemory() / (1024 * 1024)

def getUsedMemory():
  return getTotalMemory() - getFreeMemory()
```

Date-Time Scripts

Date/Time Scripting Tricks

Language:

Jython

Explanation:

The following two scripting examples demonstrate a variety of tricks that can be used when working with dates and/or times.

Script #1: Adds one day to the current date and returns a timestamp in the specified format. *Note - The day number is "self-aware", meaning, it is automatically wrapped and the month is incremented (or decremented) when the number of days in the current month are exceeded.

```
from java.util import *
from java.text import *

def getTime():
  # Get an instance of the calendar
  calendar = Calendar.getInstance()

  # Set the day in the calendar
  day = calendar.get(Calendar.DAY_OF_WEEK)
  calendar.set(Calendar.DAY_OF_WEEK, day+1)

  # Retreive the date from the calendar
  date = calendar.getTime()

  # Create the date formatter
  myFormat = "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'"
  formatter = SimpleDateFormat(myFormat)

  # Create, print, and return the timestamp
  timestamp = formatter.format(date)
  print timestamp
  return timestamp
```

Date/Time Scripting Tricks

Language:

Jython

Explanation:

The following two scripting examples demonstrate a variety of tricks that can be used when working with dates and/or times.

Script #2: Returns XML of the date which is three business days from the current date. Example: If the current day were Thursday, this method would return the date for the following Tuesday.

```
from com.parasoft.api import *
from soaptest.api import *
from java.util import *
from java.text import *

def todayDate():
   today = Date()
   cal = GregorianCalendar()
   dayCount = 0;
   while dayCount < 3:
       cal.add(Calendar.DATE, 1)
       if (cal.get(Calendar.DAY_OF_WEEK) != Calendar.SATURDAY and
cal.get(Calendar.DAY_OF_WEEK) != Calendar.SUNDAY):
           dayCount = dayCount + 1
   todayPlus3 = cal.getTime()
   parser = SimpleDateFormat("yyyyMMdd")
   dateField = parser.format(todayPlus3)
   return SOAPUtil.getXMLFromString([str(dateField)])
```

Generates XML Schema dateTime string

Language:

Jython

Explanation:

Sample script to generate XML schema dateTime string using timezone offset format

```
###  Name:    getCurrentXMLDateTimeString
###  Author:  Mark Carlson
###  Date:    09/15/2006
###
###  Description:  Returns a string containing an XML schema dateTime with
the offset from GMT
###

from com.parasoft.api import *
from soaptest.api import *
from java.util import *
from java.text import *

def getCurrentXMLDateTimeString():
  cal = GregorianCalendar().getInstance()
  today = cal.getTime()

  #Format the offset from GMT in XML schema dateTime format
  offsetParser = SimpleDateFormat("Z")  #build a SimpleDateFormat that will
return the timezone portion of the time
  offsetStr = offsetParser.format(today) #gets the timezone offset in java
format (without a colon between the offset hours and offset minutes
  xmlOffset = offsetStr[0:3] + ":" + offsetStr[3:5] #insert a colon between
the timezone hours and minutes position
  #Application.showMessage("xmlOffset = " + xmlOffset)

  #Format the current time without the timezone
  parser = SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS")
  dateField = parser.format(today)

  #append the xmlOffset
  formattedDate = dateField + xmlOffset
  #Application.showMessage("formattedDate = " + formattedDate)

  #return the current time formatted in XML schema dateTime offset format
  return formattedDate
```

Get a XML Representation of a date

Language:

Jython

Explanation:

This example returns XML of the date which is three business days from the current date. For instance, If the current day were Thursday, this method would return the date for the following Tuesday.

```
from com.parasoft.api import *
from soaptest.api import *
from java.util import *
from java.text import *

def todayDate():
   today = Date()
   cal = GregorianCalendar()
   dayCount = 0;
   while dayCount < 3:
       cal.add(Calendar.DATE, 1)
       if (cal.get(Calendar.DAY_OF_WEEK) != Calendar.SATURDAY and
cal.get(Calendar.DAY_OF_WEEK) != Calendar.SUNDAY):
           dayCount = dayCount + 1
   todayPlus3 = cal.getTime()
   parser = SimpleDateFormat("yyyyMMdd")
   dateField = parser.format(todayPlus3)
   return SOAPUtil.getXMLFromString([str(dateField)])
```

Return String containing XML schema dateTime

Language:

Jython

Explanation:

Adds one day to the current date and returns a timestamp in the specified format. *Note - The day number is "self-aware", meaning, it is automatically wrapped and the month is incremented (or decremented) when the number of days in the current month are exceeded.

```
from com.parasoft.api import *
from soaptest.api import *
from java.util import *
from java.text import *

def getCurrentXMLDateTimeString():
  cal = GregorianCalendar().getInstance()
  today = cal.getTime()

  #Format the offset from GMT in XML schema dateTime format
  offsetParser = SimpleDateFormat("Z")  #build a SimpleDateFormat that will
return the timezone portion of the time
  offsetStr = offsetParser.format(today) #gets the timezone offset in java
format (without a colon between the offset hours and offset minutes
  xmlOffset = offsetStr[0:3] + ":" + offsetStr[3:5] #insert a colon between
the timezone hours and minutes position
  #Application.showMessage("xmlOffset = " + xmlOffset)

  #Format the current time without the timezone
  parser = SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS")
  dateField = parser.format(today)

  #append the xmlOffset
  formattedDate = dateField + xmlOffset
  #Application.showMessage("formattedDate = " + formattedDate)

  #return the current time formatted in XML schema dateTime offset format
  return formattedDate
```

JDBC Scripts

Calling Methods on JDBC Objects – DB tool (UG)

Language:

Jython

Explanation:

You can chain tools to the traffic objects that result from DB tool execution. For instance, if you want to call methods on JDBC objects directly (instead of just using the XML output), you can attach an Extension tool to the traffic object and get the JDBC ResultSet, Connection, or Statement. For example, you could use the following for the Extension tool: You can then have your own code calling methods on rs, c, and s.

```
def checkJdbcObjects(input, context):
rs = input.get("ResultSet")
c = input.get("Connection")
s = input.get("Statement")
# your own code calling methods on rs, c, s
```

How to Query a Database using Jython

Language:

Jython

Explanation:

Here is a template (which can be expanded upon) written in Jython which allows you to connect to a database and perform a query. Note that you will need to add in your own data base values to make this script work correctly.

```
from java.sql import DriverManager
from java.lang import Class
from soaptest.api import *
from com.parasoft.api import *

def getValues(input, context):
   connection = getConnection()
   statement = connection.createStatement()
   QUERY = "REPLACE_WITH_SQL_QUERY"
   resultSet = statement.executeQuery(QUERY)
   while resultSet.next():
      # perform some action with returned data
      # i.e resultSet.getString("DB_COLUMN_NAME")
      #      resultSet.getInt("DB_COLUMN_NAME2")

def getConnection():
   # load driver
   driverName = "REPLACE_WITH_DRIVER_NAME"
   Class.forName(driverName)
   DB_URL = "REPLACE_WITH_URL"
   DB_USER = "REPLACE_WITH_USER"
   DB_PASSWD = "REPLACE_WITH_PASSWORD"
   # get connection to database
```

JDBC Programming pt.1 – Get Connection

Language:

Jython

Explanation:

Get a connection and save it using context.put

```
from com.parasoft.api import *

def getConnection(input, context):
    myConnection = SQLUtil.getConnection("com.mysql.jdbc.Driver",
"jdbc:mysql://boa:3306/test", "devtest", "dev%pass")
    context.put("PR83674_connection", myConnection)
```

JDBC Programming pt.2 - Create Table

Language:

Jython

Explanation:

Create a database table, based on connection in pt1.

```
from com.parasoft.api import *

def test(input, context):
    c = context.get("PR83674_connection")
    stmt = c.createStatement()
    rs = stmt.executeUpdate("CREATE TABLE PR83674 (name VARCHAR(255))")
```

JDBC Programming pt.3 - Populate Table

Language:

Jython

Explanation:

```
Populate the table created in pt2
```

```
from com.parasoft.api import *

def test(input, context):
    c = context.get("PR83674_connection")
    stmt = c.createStatement()
    rs = stmt.executeUpdate("INSERT INTO PR83674 VALUES ('abc'), ('123')")
```

JDBC Programming pt.4 - Select Statement

Language:

Jython

Explanation:

```
Do Select Statement from table from pt.3
```

```
from com.parasoft.api import *

def test(input, context):
    c = context.get("PR83674_connection")
    stmt = c.createStatement()
    rs = stmt.executeQuery("SELECT name from PR83674")
    return SQLUtil.toXML(rs);
```

JDBC Programming pt.5 - Test GetDriver

Language:

Jython

Explanation:

Test to make sure that jdbc driver is available

```
from com.parasoft.api import *

def test(input, context):
    SQLUtil.getDriver("com.mysql.jdbc.Driver");
```

JDBC Programming pt.6 - Test GetDriver

Language:

Jython

Explanation:

`Test to make sure that jdbc driver is available.`

```
from com.parasoft.api import *

def test(input, context):
    try:
       SQLUtil.getDriver("com.mysql.jdbc.Driverr");
    except:
        return 1
    return 0
```

JDBC Programming pt.7 - Drop Table

Language:

Jython

Explanation:

`Deletes the db table that was created`

```
from com.parasoft.api import *

def test(input, context):
    c = context.get("PR83674_connection")
    stmt = c.createStatement()
    rs = stmt.executeUpdate("DROP TABLE PR83674")
```

JDBC Programming pt.8 - Close Connection

Language:

Jython

Explanation:

`Close the db connection.`

```
from com.parasoft.api import *

def getConnection(input, context):
    myConnection = context.get("PR83674_connection")
    myConnection.close()
```

Changing Protocols

Switch all Tests' Protocols to HTTP 1.0

Language:

Jython

Explanation:

The first script changes all tests in the suite to HTTP 1.0 with close-connection.

```
from java.lang import *
from com.parasoft.api import *
from com.parasoft.util import Util
from webtool.messaging import *

def setTestToolSettingsA(input,context):
    toolTest = context.getParentContext()
    testSuite = toolTest.getParentTestSuite()
    updateTestSuite(testSuite, context)

def updateTestSuite(testSuite, context):
    Application.showMessage("Processing Test Suite: " + testSuite.getName())
    tests = testSuite.getTests()
    itr = tests.iterator()
    while itr.hasNext():
        test = itr.next()
        if Util.shortClassName(test.getClass()) == "TestSuite":
            #recur
            updateTestSuite(test, context)
        elif Util.shortClassName(test.getClass()) == "SOAPRPCToolTest":
            #SOAP test
            updateSOAPTest(test, context)

def updateSOAPTest(test, context):
    Application.showMessage("Test: " + test.getName())
    transportProp = test.getTool().getTransportProperties()
    transportProp.setTransport(0); #HTTP 1.0
    curHTTPProp = transportProp.getHTTPProperties()
    # new HTTP Properties args
    branch = curHTTPProp.getParentBranch()
    headers = curHTTPProp.getHTTPHeaders()
    auth = curHTTPProp.getAuthentication()
    keyName = curHTTPProp.getSSLProperties().getKeyStoreDataName()
    newHTTPProp = HTTPProperties(branch, 0, headers, auth, keyName)
    newHTTPProp.setUseChunking(0) #no chunking
    transportProp.setHTTPProperties(newHTTPProp)
```

Switch all Tests’ Protocols to HTTP 1.1

Language:

Jython

Explanation:

The second script changes all tests in the suite to HTTP 1.1 with keep-alive and chunking.

```
from java.lang import *
from com.parasoft.api import *
from com.parasoft.util import Util
from webtool.messaging import *

def setTestToolSettingsB(input,context):
    toolTest = context.getParentContext()
    testSuite = toolTest.getParentTestSuite()
    updateTestSuite(testSuite, context)

def updateTestSuite(testSuite, context):
    Application.showMessage("Processing Test Suite: " + testSuite.getName())
    tests = testSuite.getTests()
    itr = tests.iterator()
    while itr.hasNext():
        test = itr.next()
        if Util.shortClassName(test.getClass()) == "TestSuite":
            #recur
            updateTestSuite(test, context)
        elif Util.shortClassName(test.getClass()) == "SOAPRPCToolTest":
            #SOAP test
            updateSOAPTest(test, context)

def updateSOAPTest(test, context):
    Application.showMessage("Test: " + test.getName())
    transportProp = test.getTool().getTransportProperties()
    transportProp.setTransport(1); #HTTP 1.0
    curHTTPProp = transportProp.getHTTPProperties()
    # new HTTP Properties args
    branch = curHTTPProp.getParentBranch()
    headers = curHTTPProp.getHTTPHeaders()
    auth = curHTTPProp.getAuthentication()
    keyName = curHTTPProp.getSSLProperties().getKeyStoreDataName()
    newHTTPProp = HTTPProperties(branch, 1, headers, auth, keyName)
    newHTTPProp.setUseChunking(1) #no chunking
    transportProp.setHTTPProperties(newHTTPProp)
```

Generating UUIDs

Generate & Return Universally Unique ID (UUID)

Language:

Jython

Explanation:

The script below invokes the SOAPUtil.generateUUID() method to return a string containing a univserally unique ID (UUID)

```
###  Name:    generateUUID
###  Author:  Mark Carlson
###  Date:    09/15/2006
###
###  Description: Uses Parasoft API to generate universally unique ID (UUID)
###
###  Usage:  Returns string containing uuid
###
from soaptest.api import *
from com.parasoft.api import *

def generateUUID():
 uuid = SOAPUtil.generateUUID()  #invokes generateUUID API method to get UUID
#  Application.showMessage("UUID = " + uuid) #output uuid for debugging
 return uuid
#  return SOAPUtil.getXMLFromString([str(uuid)]) #uncomment to return uuid as
an XML string
```

Generating Unique Number to Write to XML Data Bank

Language:

Jython

Explanation:

To generate a unique number so that you may write that number to an XML Data Bank for use within another test you can use the following script. Within a method tool make sure python is the selected language and then enter this script. You can then chain an XML Data Bank to the method tool, which will allow the unique number to be saved and used in another test. You can change the MASK variable to allow you generate a number of whatever length you wish.

```
from soaptest.api import *

MASK = 1000

def getURL(input, context):
 # returns a unique number
 num = 1000 + SOAPUtil.getUniqueNumber()
 # returns XML allowing the number to be stored in the XML Data Bank
 return SOAPUtil.getXMLFromString( [ str(num) ] )

def addDataSources(context):
 return "Generated Data Sources"
```

Generate and return a UUID

Language:

Jython

Explanation:

The script below invokes the SOAPUtil.generateUUID() method to return a string containing a univserally unique ID (UUID)

```
from soaptest.api import *
from com.parasoft.api import *

def generateUUID():
 uuid = SOAPUtil.generateUUID()  #invokes generateUUID API method to get UUID
#  Application.showMessage("UUID = " + uuid) #output uuid for debugging
 return uuid
#  return SOAPUtil.getXMLFromString([str(uuid)]) #uncomment to return uuid as
an XML string
```

Java Examples

Java Example – Possible Return Values

Language:

Java

Explanation:

Tool Success Indicator Success. (this just shows what you can return, its really just another java example)

```
package tests.soaptest.classes;

public class Indicators {
    public String trueString() {
        return "true";
    }
    public String falseString() {
        return "false";
    }
    public String oneString() {
        return "1";
    }
    public String zeroString() {
        return "0";
    }
    public boolean falseboolean() {
        return false;
    }
    public boolean trueboolean() {
        return true;
    }
    public Boolean falseBoolean() {
        return Boolean.FALSE;
    }
    public Boolean trueBoolean() {
        return Boolean.TRUE;
    }
}
```

Java Example – Application.showMessage

Language:

Jython

Explanation:

In extension tool, added by choosing java, and picking:
tests.soaptest.classes.MethodToolContextFromJava

```
package tests.soaptest.classes;

/**
 * Sample Java Code used in functional regression testing
 */

import com.parasoft.api.Application;
import com.parasoft.api.ExtensionToolContext;
import com.parasoft.api.MethodToolContext;
import com.parasoft.api.ScriptingContext;

public class MethodToolContextFromJava {

      public static void doMethodToolContext(Object input, MethodToolContext
context) {
        Application.showMessage("Hello world from MethodToolContext");
      }

      public static void doExtenstionToolContext(Object input,
ExtensionToolContext context) {
        Application.showMessage("Hello world from ExtensionToolContext");
      }

      public static void doScriptingContext(Object input, ScriptingContext
context) {
        Application.showMessage("Hello world from Scripting Context");
      }
}
```

Supplying multiple parameters to a java method

Language:

Jython

Explanation:

When using the method tool to integrate custom Java code into SOAtest, you are limited to only 1 input argument to your Java methods. For example you could have the following argument parameter list:

```
// "input" is the input the method tool receives to process
public int invoke(Object input, MethodToolContext context) {
// ...
}
```

But you may want to have something like:

```
public int invoke(String arg1, String arg2, Object arg3) {
// ...
}
```

You can get around this by using the Java Class in a python script, for example:

```
from mypackage import *

def invokeMyClass():
 myClass = MyClass()
 returnValue = myClass.invoke("1st arg", "2nd arg", MyOtherClass())
 if returnValue > 0:
   return 1
 return 0
```

Use Datasource Value & Return in XML - Java (UG)

Language:

Java

Explanation:

One typical usage of an Extension tool is accessing data from a data source, manipulating it, and then storing it dynamically in an XML Data Bank or Writable Data Source. In this scripting example, we define a method called getKeywords in which we are accessing data from a data source titled “Books” with a data source column titled “keywords”. We then return an XML representation of this string so that we can send the output from this script to an XML Data Bank.

```
package examples;
import soaptest.api.*;
import com.parasoft.api.*;

public class Keyword {
public Object getKeywords(Object input, ExtensionToolContext context)
throws com.parasoft.data.DataSourceException {
String[] titles = new String[1];
titles[0] = context.getValue("Books", "keywords");
return SOAPUtil.getXMLFromString(titles);
}
}
```

Request/Response Message Manipulation

How to extract string values from an XML message

Language:

Javascript

Explanation:

How to extract string values from an XML Message

```
from com.parasoft.api import *
from soaptest.api import *

def dataAccess(input, context):
 # get the String value
 value = SOAPUtil.getStringFromObject(input)
 # print out the string in the Message window
 Application.showMessage("Transformer + Script Combo = " + value)
```

Check Header of Response for a particular string

Language:

Jython

Explanation:

This has to be chained to a response header of soap client. This particular test checks to make sure that an HTTP/1.1 200 OK response comes back, but you can check it for any string response.

```
This has to be chained to a response header of soap client


from com.parasoft.api import *
from soaptest.api import *
from java.lang import *

def checkHeader(input, context):
  header = String(SOAPUtil.getStringFromObject(input))
  if header.indexOf("HTTP/1.1 200 OK") == -1:
    return 0
  return 1
```

Search All Traffic for particular string

Language:

Jython

Explanation:

Search for a string in all of your traffic

```
from soaptest.api import *
from java.lang import *

def checkTraffic(x, context):
  traffic = x.get(SOAPUtil.XML_REQUEST)
  str = String(traffic)
  if str.indexOf("Hello World") != -1:
    context.put("bob", "bob") //or just do something
```

Adding a Processing Instruction to Request Message

Language:

Jython

Explanation:

Problem Statement: I need to execute an HTTP post of an XML message, which has an schema, but also it needs an XML header (that is not the same as an HTTP Header) This XML header is not included in the schema (because it defines the body of the message). The XML should look like this:

<?xml version="1.0" ?>
<?Label 21247|RTAV|20590|SUCCESS?> <!-- this is the XML header -->
<RtavMessage xmlns="rtav.fidelio.2.0">
<!-- Body of the message -->
</RtavMessage>

So I need an appropriate way to add the XML header in the message before being sent.

Solution: A processing instruction (http://www.w3.org/TR/2008/REC-xml-20081126/#sec-pi) is not supported in the form input or form xml views, but can be added to the request message of a messaging client by chaining an extension tool to the request message, and adding the following script:

```
var SOAPUtil = Packages.soaptest.api.SOAPUtil

function addPi(input, context) {
      var inputStr = String(input)
      var myPi = "<?Label 21247|RTAV|20590|SUCCESS?>"
      return myPi + inputStr

}
```

Strip an Element from an XML Message

Language:

Jython

Explanation:

Strip an element from an xml message. Note: This script uses internal parasoft api and is not good to give to customers unless it's a pr workaround.

```
HEADER_TO_REMOVE="UsernameToken"

from com.parasoft.api import Application
from com.parasoft.xml import XMLUtil
from javax.xml.xpath import *

def stripHeader(message):
    messageDom = XMLUtil.buildDocumentFromString(message)
    factory = XPathFactory.newInstance()
    xpath = factory.newXPath()
    expr = xpath.compile("//*[local-name(.)='" + HEADER_TO_REMOVE + "']")
    node = expr.evaluate(messageDom, XPathConstants.NODE)
    # Application.showMessage("Node: " + XMLUtil.toString(node))
    if node is not None:
        # Application.showMessage("Removing header")
        parent = node.getParentNode()
        parent.removeChild(node)
    return XMLUtil.serialize(messageDom)
```

Changing Mime type of Response

Language:

Jython

Explanation:

changing the mimetype of a response.

What if you need to change the mimetype of your response in order to for example use xml databank with the response and your response isn't text/xml.

```
from com.parasoft.api import *
def changeMimeType(input, context):
    return Text(input, "text/xml")
```

Transport/JSON Scripts

MQ Properties Scripting Hook

Language:

Jython

Explanation:

The Scripting Hook options allow you to customize MQ Properties by using scripting language such as Jython, Java, and JavaScript. If you need more information on using SOAtest's scripting utility, please refer to the Scripting section of the tutorial. For a list of scripting APIs, go to Help> Extensibility API in the SOAtest GUI. The following are scripting access keys:

- QueueManager – mqManager
- GetQueue – mqGetQue
- PutQueue – mqPutQue
- PutMessage – mqPutMessage
- GetMessage – mqGetMessage
- PutMessageOptions – mqPutMessageOptions
- GetMessageOptions – mqGetMessageOptions

For example, if you like to change the Expiry time for put message to 999:

```
from com.ibm.mq import *

def changeExpiry(context):
   putMessage = context.get("mqPutMessage")
   putMessage.expiry = 999
```

Extract a JMS Header and Capture in Databank

Language:

Jython

Explanation:

At the moment there's no other way of extracting JMS header other than through scripting. Header Data Bank only works with HTTP headers. That said, here's a way of extracting the headers and passing it to an XML Data Bank so you can re-use it for a latter test.

The above script will retrieve the entire JMS header as a java.lang.String object. You'll need to parse it to extract a specific jms header property. Then you can return it by using SOAPUtil.getXMLFromString(Object[]). This static method will serialize it into XML so that our XML Data Bank can read the input that's returned from the script.

Lastly, you'll need to place this script in a Method tool which in turn must be attached to the Call Back tool's Traffic Object. So when you add the Method tool do the following,

1. Right click on Call Back tool and select Add Output
2. Select Both > Traffic Object
3. Select Method tool

I have added some missing import statements and comments to above code . Also, just side note, you can use SOAPUtil.HTTP_REQUEST_HEADER variable instead of "Request HTTP Headers". Please, refer to this new code snapshot.

```
from java.lang import *
from java.util import *
from soaptest.api import *

def parseJMSHeader(input, context):

  jmsHeader = SOAPUtil.HTTP_HEADERS_REQUEST
  context.put("Request HTTP Headers", jmsHeader)
  #
  #   To cast between Java/Python strings, use String() and str()
  #   Over here, you can perform header processing logic by parsing jmsHeader
into different elements
  #   based on your service needs.
  #
  return SOAPUtil.getXMLFromString([jmsHeader])
```

JSON to XML script

Language:

Javascript

Explanation:

JSON to XML script

```
var XMLUtil = Packages.com.parasoft.xml.XMLUtil;
var SOAPUtil = Packages.soaptest.api.SOAPUtil;

function jsonToXML(input, context) {
 var json = input.toString();
 json = json.replace(/\\([^"])/g,"\\\\$1");
 eval("var o = " + json);
 var val = new Array();
 val[0] = convert(o);
 return SOAPUtil.getXMLFromString(val, true);
}

function convert(o) {
  var xml="";
  for (var m in o)
     xml += toXml(o[m], m, "");
  return "<json>" + xml + "</json>";
}

function toXml(v, name, ind) {
     var xml = "";
     if (v == null) {
       return xml;
     }
     if (v instanceof Array) {
        for (var i=0, n=v.length; i<n; i++)
           xml += ind + toXml(v[i], name, ind+"\t") + "\n";
     } else if (typeof(v) == "object") {
        var hasChild = false;
        xml += ind + "<" + name;
        for (var m in v) {
              hasChild = true;
        }
        xml += hasChild ? ">" : "/>";
        if (hasChild) {
           for (var m in v) {
             xml += toXml(v[m], m, ind+"\t");
           }
           xml += (xml.charAt(xml.length-1)=="\n"?ind:"") + "</" + name +
">";
        }
     }
```

```
      else {
         var value = XMLUtil.toXML(v.toString(),true);
         value = value.replace(/&#x(\d+);/g, "\\u$1"); // turn &#xnnnn; to
\unnnn
         xml += ind + "<" + name + ">" + value +  "</" + name + ">";
      }
      return xml;
   }
```

XML to JSON

Language:

Jython

Explanation:

You will need the xmltojson.jar file in this folder. Save it to the user's machine, and add it to the classpath in SOAtest under SOAtest Preferences > System Properties.This jar file / library comes from http://www.json.org/java/index.html.In order to convert XML to JSON with this library, you will need an Extension (Method) Tool. You can chain this Extension Tool to a SOAP Client, Messaging Client, etc. In the method tool, your script will look something like this:

```
from com.parasoft.api import *
from java.lang import *
from org.json import *

def XMLtoJSON(input, context):
   obj = XML.toJSONObject(String(str(input)))
   output = obj.toString()
   return output

Note that this script also works well as a Global Tool.
```

How to Accommodate File Input to the JSON Databank

Language:

Jython

Explanation:

The JSON Data Bank does not accept file input. For your reference, I have filed PR 94109 to address this deficiency. However, I have also found a solution for you. You can chain the Data Bank to an Extension Tool that returns the file contents.

If you run the Extension Tool then the JSON Data Bank will be initialized with the file contents, allowing you to extract properties

You can use the following code in your Extension Tool:

```
from com.parasoft.api import IOUtil

def test(input, context):

   myFile = context.getAbsolutePathFile("OpenBudgetGroup_Response.txt")
   return IOUtil.readTextFile(myFile)
```

Miscellaneous Scripts

XML Assertion Script, Check Evenness (UG)

Language:

Jython

Explanation:

Enter the following script, which ensures that the price value which is being checking in the xml assertor 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
```

How to Use Scripted XML in SOAP Client (old)

Language:

Jython

Explanation:

The following example shows how to use the "Scripted XML" option within a SOAP Client to dynamically generate XML.

To use this example:

1. Create a SOAP Client
2. Enter "http://soaptest.parasoft.com/calculator.wsdl" for the WSDL URI
3. Change the SOAP Envelope combo box to "Scripted XML"
4. Enter the script shown below into the text window
5. Make sure that "getBody()" is the selected method
5. Run the test and view the traffic.

```
prefix = """<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Body>
 <add xmlns="http://www.parasoft.com/wsdl/calculator/">"""

suffix = """</add>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>"""

def getBody():
   number1 = 5
   number2 = 10
   return prefix + "<x>" + str(number1) + "</x>" + "<y>" + str(number2) +
"</y>" + suffix
```

Executing a Windows OS Command

Language:

Java

Explanation:

Executing a Windows OS Command

```
import java.lang.Runtime;
import com.parasoft.api.*;

public class copyFile
{

	public static void execute(Object obj, MethodToolContext context)
	{
		Runtime rt = Runtime.getRuntime();
		try
		{
			rt.exec("vcp -pw thepassword
H:\\SOAPTests\\workFiles\\compact.sms root@stg-jav01:/var/spool");
		}
		catch(Exception e)
		{
			System.out.println(e);
		}
	}
}
```

Get IP Address of Local Machine, Save in Databank

Language:

Jython

Explanation:

I found some JavaScript that will give me the IP Address of the local machine where I am running tests and this code works quite nicely. You can than save the return value for later use in an XML Data Bank

```
from com.parasoft.api import *
from soaptest.api import *
from java.lang import *
from java.net import *

def getgetIPAddress(input, context):
    ip = InetAddress.getLocalHost()
    ipStr = String(ip.getHostAddress())
    return SOAPUtil.getXMLFromString([ipStr]);
```

Change the Default Returned XML Structure from Extension Tool

Language:

Jython

Explanation:

the return SOAPUtil.getXMLFromString(["myStringList"] usually yields:
The format is always a simple list:
<root>
<z0>myString1</z0>
<z1>myString2</z1>
...
</root>

Is there a way that I can create my own XML structure, like say:

<myTop>
<name>column1</name>
<value>Under Name1</value>
<name>column2</name>
<value>Under Name2</value>
...
</myTop>

The XML Data Bank expects an object called Text. This class has a MIME Type, and the XML Data Bank expects this to be text/xml. So there are two ways of accomplishing what you want, and both require that you build the XML yourself into a String:

1) Create and return a new Text with your XML String, and set its MIME Type to text/xml:

```
def option1(input, context):
    myXML = "<myTop>   <name>column1</name>      <value>Under Name1</value>
<name>column2</name>      <value>Under Name2</value></myTop>"
    return Text(myXML, "text/xml")
```

2) Use the SOAPUtil.getXMLFromString(String[], boolean) method. This alternative form of getXMLFromString() takes a boolean, which indicates whether the String[] argument is already an XML document. If it is, then the proper Text object is automatically built and returned.

```
from com.parasoft.api import *
from soaptest.api import *

def option2(input, context):
    myXML = "<myTop>   <name>column1</name>      <value>Under Name1</value>
<name>column2</name>      <value>Under Name2</value></myTop>"
    return SOAPUtil.getXMLFromString([myXML], 1)
```

Running a Command Line from Jython Script

Language:

Jython

Explanation:

What can I use to execute a command line command from a python method in SOATest 6.1?

If possible, I would recommend that you use the External Tool which is designed for running external commands without any scripting. There is information about this tool in the SOAtest User's Guide (Help->Help Contents) under "Reference->Available Tools->Other Tools->External".If you must do this from a python script then I would recommend using the java.lang.Runtime.getRuntime().exec() method from Sun's Java API (see http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Runtime.html#exec%28java.lang.String%29). SOAtest uses Jython for its python engine which is able to access Java classes. For example, this simple python method will run the DOS command "dir C:" then print the command's output to SOAtest's Console view:

```
from com.parasoft.api import *
from java.lang import *
from java.io import *

def runMyCommand(input, context):
  proc = Runtime.getRuntime().exec("cmd /C "dir c:\"")
  inStream = BufferedReader(InputStreamReader(proc.getInputStream()))
  outputLine = inStream.readLine()
  while outputLine != None:
  Application.showMessage(outputLine)
  outputLine = inStream.readLine()
  inStream.close()
```

Define a Custom Failure Message from Extension Tool

Language:

Jython

Explanation:

Make a extension tool fail and user a custom error message reported to quality tasks

The error will be appended to the regular SOAtest tab where all test results gets reported to. It will also make the Method/Extension tool be marked as failed (without having to return 0 and having the 'Return value indicates success' checked).

```
def foo(input, context):
   context.report("my error message")
```

Script to check lexicographical ordering

Language:

Jython

Explanation:

script to check that book titles are ordered lexicogaphically, print to console if not.Exit code indicates succes. For example, These titles will fail:

C++ How to Program (4th Edition)
Java How to Program (4th Edition)
Linux Administration Handbook
Java in a Nutshell, Fourth Edition
Oracle PL/SQL Programming, 3rd Edition
PowerBuilder 7.0 Unleashed

with this console message:
Test Failure: Titles are NOT ordered lexicographically:
Java in a Nutshell, Fourth Edition came after Linux Administration Handbook

```
from com.parasoft.api import *
from java.io import *

def isAlphabitized(x, context):
    reader = BufferedReader(StringReader(x))
    line = reader.readLine()
    lastLine = line
    while line != None:
        # Application.showMessage(line)
        if lastLine > line:
            Application.showMessage("Test Failure: Titles are NOT ordered
lexicographically:")
            Application.showMessage("              " + line + " came after "
+ lastLine)
            return 0
        lastLine = line
        line = reader.readLine()
    Application.showMessage("        Titles are ordered lexicographically")
    return 1
```

Print elapsed time between request/response

Language:

Jython

Explanation:

first script, Method - startTime: chain to the request SOAP evelope. This will save the start time into a variable which can be accessed by the next script. second script, Method - stopTime: chain to the request SOAP evelope: This will subtract the original start time from the current time which will give the elapsed time between the request/reponse.

```
#first script, Method - startTime: chain to the request SOAP evelope:


from java.lang import *
def startTime(obj,context):
  context.put("START_TIME",Long(System.currentTimeMillis()))
  return obj

#second script, Method - stopTime: chain to the request SOAP evelope:

from java.lang import *
from com.parasoft.api import *

def stopTime(obj,context):
  exeTime = System.currentTimeMillis() - context.get("START_TIME")
  Application.showMessage("Execution time = " + str(exeTime))
```

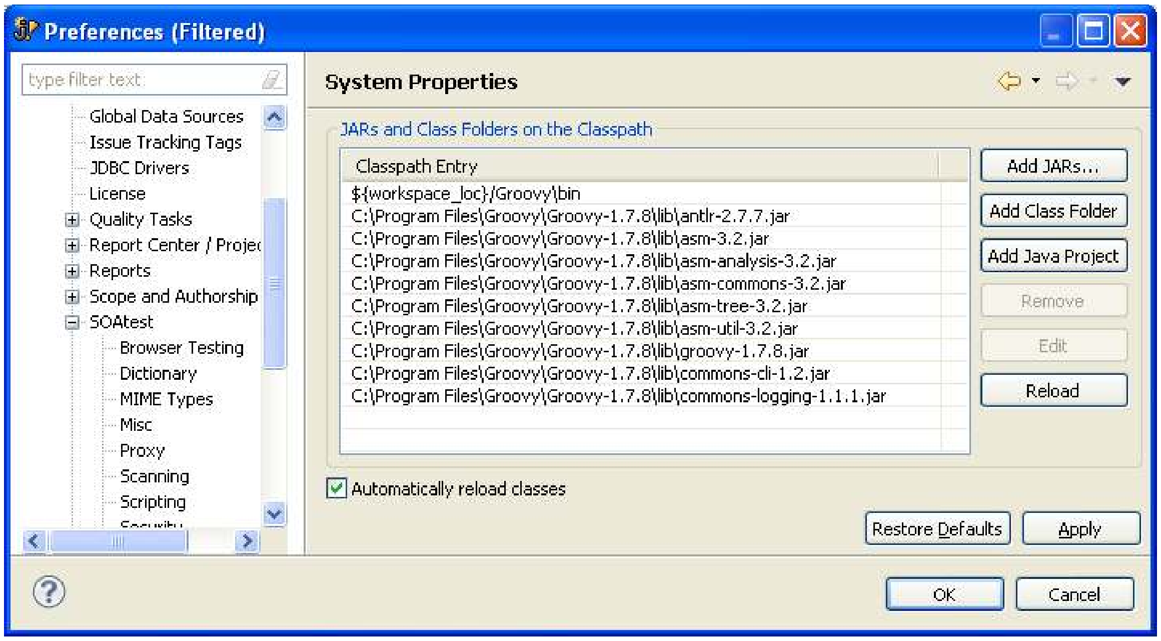
Using Groovy Scripts with SOATest

Language:

Jython

Explanation:

First we need to add some of the Groovy Jar files to the SOAtest System Properties as below:



Classpath Entry
${workspace_loc}/Groovy\bin
C:\Program Files\Groovy\Groovy-1.7.8\lib\antlr-2.7.7.jar
C:\Program Files\Groovy\Groovy-1.7.8\lib\asm-3.2.jar
C:\Program Files\Groovy\Groovy-1.7.8\lib\asm-analysis-3.2.jar
C:\Program Files\Groovy\Groovy-1.7.8\lib\asm-commons-3.2.jar
C:\Program Files\Groovy\Groovy-1.7.8\lib\asm-tree-3.2.jar
C:\Program Files\Groovy\Groovy-1.7.8\lib\asm-util-3.2.jar
C:\Program Files\Groovy\Groovy-1.7.8\lib\groovy-1.7.8.jar
C:\Program Files\Groovy\Groovy-1.7.8\lib\commons-cli-1.2.jar
C:\Program Files\Groovy\Groovy-1.7.8\lib\commons-logging-1.1.1.jar

[✓] Automatically reload classes

Note: You will also need to add the Jar file for the com.eviware.soapui.support.XmlHolder class.

Add an XML Databank to the SOAP tool that you wish to store data e.g. the XML request & response message. This will store the information in the SOAtest ‘context’ environment for use in the Groovy script.

Create a java class e.g.

```
package com.parasoft.andrew;

import com.parasoft.api.*;
import java.io.*;

public class PruGroovy {

	/**
	 * @param args
	 */

	public String runGroovyScript(Object obj, ExtensionToolContext context
)
	{
		Application.showMessage("runGroovyScript Start");

		// Check that we have a valid context object.
		if (context == null)
		{
			Application.showMessage("Context is null");
			return "false";
		}

		// Create a new GroovyShell object
		groovy.lang.GroovyShell groovy = new groovy.lang.GroovyShell();

		Object result = "false";

		try{
			// Create a File object for teh Grrovy Script. This format
below will need the script to be placed in
			// the location e.g. C:\Program Files\ParaSoft\SOAtest\9.0
			File fin = new File("test.groovy");
			java.util.List list = new java.util.ArrayList();
			Application.showMessage("runGroovyScript 2");

			// Pass the context object into the scripting world.
			groovy.setVariable("context", context);

			// Run the script and get the result.
			result = groovy.run(fin,list);
			Application.showMessage("returned Value: " +
result.toString());
		}
		catch (Exception e)
		{
			Application.showMessage("runGroovyScript Exception: ");
			Application.showMessage(e.getMessage());
			Application.showMessage(e.toString());
			e.printStackTrace();
		}

		// The Groovy script should return true or false!
```

```
                return result.toString();
        }

}
```

Call this from an Extension Tool, with ‘Exit code indicates success’ and ‘Use data source’ both checked. Please note that if you recompile the java class then use the ‘reload Class’ button on the Extension tool page to ensure the changes get picked up in SOAtest.

Select the correct method from the drop down list.

The Groovy Script:

```
import com.parasoft.api.*;
import com.eviware.soapui.support.XmlHolder

// Get hold of our Context object
def context = binding.getVariable("context")

// Check it is not null again
if (context == null)
    {
        System.out.println("null");
        return "false";
    }
else
    {
        // Define the request and response holders
        def request = new XmlHolder(context.getValue("Generated Data Source","generateIllustration" )
        def response = new XmlHolder(context.getValue("Generated Data Source","generateIllustrationResponse"));

        // Do your assertions

        // return true or false
        return true;
    }
```

Please note that I have not tried using the XMLHolder object as I have no access to this Jar.

How to use/import a jython module

Language:

Jython

Explanation:

It is often the case that you may want to import additional python or jython modules into SOAtestfor use in the extension tool or wherever custom scripts are enabled.

reference on modules:

http://jythonpodcast.hostjava.net/jythonbook/chapter7.html

At the time of this writing, the current version of jython installed on SOAtest is version 2.2.1. , so in order to be consistent with this you will want to make sure that the version of jythonthat you download and use to create modules matches this. You can download 2.2.1 here:

http://www.jython.org/downloads.html

To enable the ability of importing your jython modues into your jython scripts in SOAtest, go to Parasoft->Preferences->Parsoft->SOAtest->Scripting. Here you will be met with the option to specify your jython Home, which is the location of where you installed your jython and where your module resides.

If you have additional modules which are not accessible from jython home, than you can add them to the Jython Path. If you have them in multiple locations, then you can include multiple directories for your jython modules by typing "c:\folder1;c:\folder2". It makes no difference whether you download and use jython 2.2.1 or python 2.2.1, since jython is a java enabled version of python. you can than import your module by using a script like this:

Extract

Language:

Jython

Explanation: