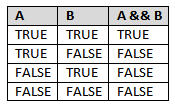
**Learn Advanced SoapUI Groovy Scripting Concepts**

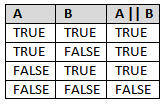
### ****#1. Control or logical statements:****

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

“And” Operator truth table:

[](http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2015/06/CONDITIONAL-STATEMENTS-IN-GROOVY-1.jpg)

“OR” operator truth table:

[](http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2015/06/CONDITIONAL-STATEMENTS-IN-GROOVY-2.jpg)

“NOT” operator for negation purpose

**Conditional statement:** programming languages support the following conditional statements:

* If…else statement
* If…else if…else statement
* Ternary operator
* Switch statement

**A) if…else block syntax:**

if <condition>  
{  
<Statements to be executed>  
}  
Else  
{  
<default statements>  
}

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

Look at the example code snippet.

|  |  |
| --- | --- |
| 1 | int a=100 |
| 2 | int b=200 |

|  |  |  |
| --- | --- | --- |
| 3 | if (a>b) | |
| 4 | { |

|  |  |  |
| --- | --- | --- |
| 5 | log.info('B is greater than A'); | |
| 6 | <To Do> |

|  |  |
| --- | --- |
| 7 | } |
| 8 | else | |

|  |  |  |
| --- | --- | --- |
| 9 | { | |
| 10 | | log.info('A is greater or both are equal'); | |

|  |  |  |
| --- | --- | --- |
| 11 | <To Do> | |
| 12 | } |

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

**B) Multiple conditional blocks: if…else if…else syntax**

if <condition1>  
{  
                <Statements to be executed>  
}  
Else if <condition2….n>  
{  
                <Statements to be executed>  
}  
Else  
{  
                <default statements>  
}

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

|  |  |
| --- | --- |
| 1 | int a=100 |
| 2 | int b=200 |

|  |  |
| --- | --- |
| 3 | int c=300 |
| 4 | if (a>b && a>c) | |

|  |  |
| --- | --- |
| 5 | { |
| 6 | log.info('A is greater than B and C'); | |

|  |  |
| --- | --- |
| 7 | } |
| 8 | else if  (b>a && b>c) | |

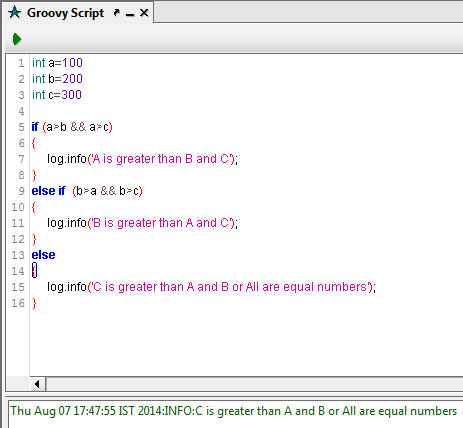
|  |  |  |
| --- | --- | --- |
| 9 | { | |
| 10 | | log.info('B is greater than A and C'); | |

|  |  |
| --- | --- |
| 11 | } |
| 12 | else | |

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

|  |  |
| --- | --- |
| 15 | } |

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

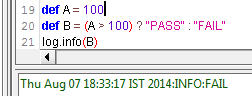
[](http://cdn2.softwaretestinghelp.com/wp-content/qa/uploads/2015/06/CONDITIONAL-STATEMENTS-IN-GROOVY-3.jpg)

**C) Ternary operator:**

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

def A = 100  
def B = (A > 100)***? “PASS” : “FAIL”***  
log.info(B)

Here’s the screenshot for the above script.

[](http://cdn2.softwaretestinghelp.com/wp-content/qa/uploads/2015/06/CONDITIONAL-STATEMENTS-IN-GROOVY-4.jpg)

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

Switch <value>  
{  
case <match1>:  
                <execute block>;  
                break;  
case <match1…match (n)>:  
                <execute block>;  
                break;

default:  
<execute default block>;  
}

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

|  |  |  |
| --- | --- | --- |
| 1 | def country="India" | |
| 2 | switch(country) |

|  |  |
| --- | --- |
| 3 | { |
| 4 | case "Japan": | |

|  |  |  |
| --- | --- | --- |
| 5 | log.info('Country matched with 1st case statement'); | |
| 6 | break; |

|  |  |
| --- | --- |
| 7 | case "Australia": |
| 8 | log.info('Country matched with 2st case statement'); | |

|  |  |  |
| --- | --- | --- |
| 9 | break; | |
| 10 | | case "India": | |

|  |  |  |
| --- | --- | --- |
| 11 | log.info('Country matched with 3st case statement'); | |
| 12 | break; |

|  |  |
| --- | --- |
| 13 | default: |
| 14 | log.info('None of the matches available :('); | |

|  |  |
| --- | --- |
| 15 | } |

**Here’s the output screenshot for the above script.**

**[](http://cdn2.softwaretestinghelp.com/wp-content/qa/uploads/2015/06/CONDITIONAL-STATEMENTS-IN-GROOVY-5.jpg)**

### ****#2. Looping or Iterative Statements:****

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

------------

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

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

* For loop
* While loop

**For loop:**

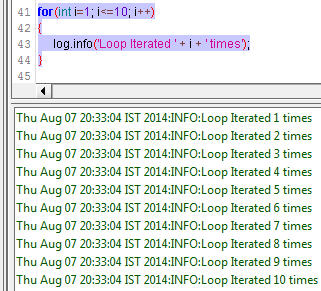
for (<initialization>; <condition>; <increment /decrement>)  
{  
                <Execute Statements>;  
}

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

See the below script

for(int i=1; i<=10; i++)  
{  
                log.info(‘Loop Iterated ‘ + i + ‘ times’);  
}

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

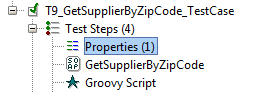
[](http://cdn2.softwaretestinghelp.com/wp-content/qa/uploads/2015/06/CONDITIONAL-STATEMENTS-IN-GROOVY-6.jpg)

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

* Add Testcase with your desired name. I have created as “T9\_GetSupplierByZipCode\_TestCase”.
* Then add a property named “Property\_Zipcode”
* Add Test Request step for “GetSupplierByZipCode” service and put the property name in the request as shown in the screenshot.

[](http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2015/06/CONDITIONAL-STATEMENTS-IN-GROOVY-7.jpg)

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

[](http://cdn2.softwaretestinghelp.com/wp-content/qa/uploads/2015/06/CONDITIONAL-STATEMENTS-IN-GROOVY-8.jpg)

In the groovy script editor, write the following script.

|  |  |  |
| --- | --- | --- |
| 1 | for (int zipCode=1; zipCode<10; zipCode++) | |
| 2 | { |

|  |  |
| --- | --- |
| 3 | testRunner.testCase.testSteps['Properties']. |
| 4 | setPropertyValue('Property\_Zipcode', '3000' + zipCode ) | |

|  |  |  |
| --- | --- | --- |
| 5 | def testStep = testRunner.testCase.testSteps['GetSupplierByZipCode']; | |
| 6 |  |

|  |  |  |
| --- | --- | --- |
| 7 | if (zipCode>5) | |
| 8 | { |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | log.info('\*\*\*\*\*\*\*\*\*\*\*\*\*\*Stopped Execution\*\*\*\*\*\*\*\*\*\*\*\*\*'); | | |
| 10 | | break; |

|  |  |
| --- | --- |
| 11 | } |
| 12 | testStep.run(testRunner,context); | |

|  |  |  |
| --- | --- | --- |
| 13 | log.info('Executed ' + zipCode + ' times') | |
| 14 | } |

The following result is received in the log as well as in the service response screen.  
(Click image for enlarged view)

[](http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2015/06/CONDITIONAL-STATEMENTS-IN-GROOVY-9.jpg)

**“while” loop:**

**Syntax:**

while <condition>  
{  
                <Execute Statements>;  
}

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

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

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

### ****#3. Arrays Collection:****

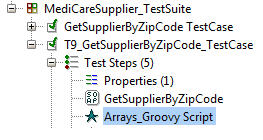
Array collection helps store multiple values in a single variable or object. Array index starts at zero by default and we need to use that index with array name to access the corresponding value stored in the array. Syntax to declare arrays in groovy script:

arrayName = new Object[5] or,  
arrayName = new Object[10][] or,  
arrayName = new Object[10][10][]

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

ArrayObj = new Object [5];  
ArrayObj[0] = “Testcase1”;  
ArrayObj[1] = “Testcase2”;  
ArrayObj[2] = “Testcase3”;  
ArrayObj[3] = “Testcase4”;  
ArrayObj[4] = “Testcase5”;

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

[](http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2015/06/CONDITIONAL-STATEMENTS-IN-GROOVY-10.jpg)

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

|  |  |
| --- | --- |
| 1 | def MAX\_LIMIT = 5 |
| 2 | def zipCodes = new Object[MAX\_LIMIT] | |

|  |  |
| --- | --- |
| 3 |  |
| 4 | zipCodes[0] = "92704" | |

|  |  |
| --- | --- |
| 5 | zipCodes[1] = "99362" |
| 6 | zipCodes[2] = "31401" |

|  |  |
| --- | --- |
| 7 | zipCodes[3] = "90247" |
| 8 | zipCodes[4] = "87102" |

|  |  |  |
| --- | --- | --- |
| 9 |  | |
| 10 | | int i=0; | |

|  |  |  |
| --- | --- | --- |
| 11 | while (i<5) | |
| 12 | { |

|  |  |  |
| --- | --- | --- |
| 13 | if (i<5) | |
| 14 | { |

|  |  |  |
| --- | --- | --- |
| 15 | testRunner.testCase.testSteps['Properties']. | |
| 16 | setPropertyValue('Property\_Zipcode',zipCodes[i]); |

|  |  |  |
| --- | --- | --- |
| 17 | def testStep = testRunner.testCase.testSteps['GetSupplierByZipCode']; | |
| 18 | testStep.run(testRunner,context); |

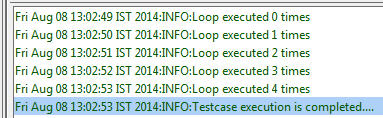
|  |  |  |
| --- | --- | --- |
| 19 | log.info('Loop executed ' + i + ' times'); | |
| 20 | } |

|  |  |  |
| --- | --- | --- |
| 21 | i++; | |
| 22 | } |

|  |  |
| --- | --- |
| 23 | log.info("Testcase execution is completed...."); |

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

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

[](http://cdn2.softwaretestinghelp.com/wp-content/qa/uploads/2015/06/CONDITIONAL-STATEMENTS-IN-GROOVY-11.jpg)

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

### ****Conclusion****

That was an overall introduction with examples to conditional or logical blocks that include IF…ELSE, switch and ternary statements. These statements are controlled by the logical operators such as AND, OR, NOT. When compared to “switch” block “if” block is fast and simple. To handle multiple collections like arrays to iterate the testing process, loop statements are critical.