Silk Central Test Manager Integration

# Problems:

1. SCTM supports execution of different types of scripts (i.e. VBScript, JScript, Perl, Python etc.). However not all script types can be executed directly. In the case of using Python scripts Windows Script Host (WSH) is used to launch the script – specifically CSsript.exe, unfortunately, this does not work as it is explained in the user guide.
2. If a test script / suite has many individual test cases the entire suite is reported as pass / fail – not the individual tests.
3. SCTM requires that the test script generate an XML file (named output.xml) describing the result of the test execution. In case of NUnit or JUnit, the corresponding test framework creates the XML file; however both NUnit and JUnit create only one XML file for all the test cases under the same test script (this is similar to issue #2). There is a need to report each case as an individual pass / fail – not the entire run.
4. SCTM deletes any files that are written to the “current working directory” location. When SCTM executes a test, each runs in a unique folder, thus isolating the output files. This is fine for cases where one script equals one output, but that is not our case.

# Solution:

## Python Script log file:

The Python script must create its log file in a specific location. This is so the individual test parsing code can find the log file without SCTM deleting the log file output.

This is currently set to be the directory from where the Python script is stored; therefore it is the local folder of the Silk Execution Server where SCTM copies the scripts to for execution – not the current working directory which is a temporary path. This is defined in SCTM as the “location” of the test (a network UNC and a local folder of the Execution Server).

The name of the log file is derived from the script name. By this I mean that the script BVTTest.py produces a log named BVTTest.py.log

The script needs to report the following items for the parsing routine to discover each unique test:

“Start: <name of test>”

<Detail of test output>

“Test Passed” or “Test Failed”

“End:” <name of test>

## AmanoExecution.vbs:

This is a VBScript that has been developed to serve three functions:

1. Test setup (run the Python script)
2. Result parsing (find the results for an individual test and write them to a unique output.xml)
3. Test cleanup (delete the log from the run)

This has been integrated into one script with multiple subroutines.

Sending four arguments expects that the script is going to be executed to create a log output.

Sending two arguments expects that a log file will be searched for individual test output and parse that output to a file output.xml

Sending two arguments with one argument equal to “delLog” attempts to delete the script log file that is passed.

## SCTM Configuration:

The setup in SCTM is a bit tedious but the reporting output is worth the effort. Setting parameters at parent locations makes setting the parameters much easier.

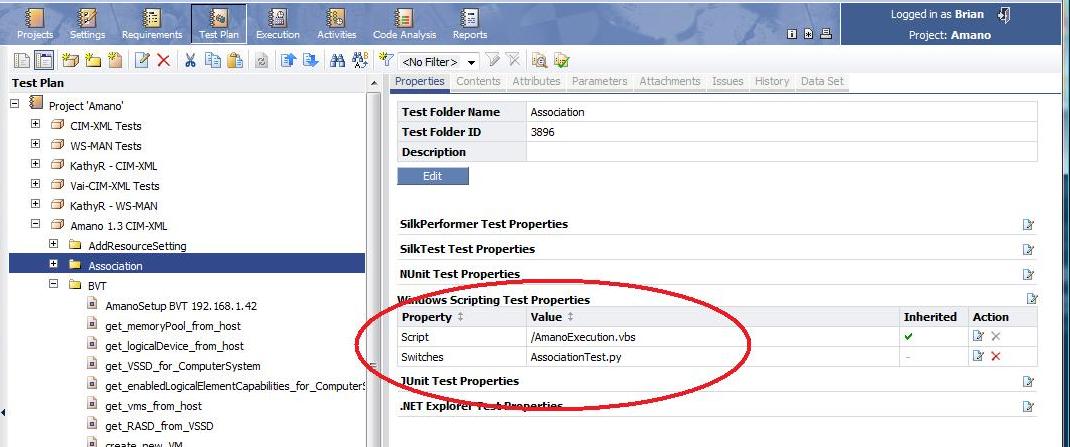
In the Test Plan Create a Test Container for the series.

In the properties of the Test Container, define the Windows Scripting Test Properties; selecting the VBScript AmanoExecution.vbs.

Under the Test Container, create a unique folder for each script.

Add the Windows Scripting Test Property of the Script, selecting “inherit from parent”

Add a second Windows Scripting Test Property for the switches, naming the individual Python script being executed.



### Creating a test case:

Under the folder add a New Child Test Definition of Type: “Windows Scripting Test”

Clicking Next will automatically insert the settings declared at the Folder level.

Click the check box to Inherit the script from the parent, for the switch add the additional parameters necessary.

There are three types of test cases necessary for each script:

Setup case: script.py <host IP or name> <username> <password>

Parsing case: script.py <test>

…additional tests...

Teardown case: script.py delLog

### Creating the Execution:

On the Execution section of SCTM, create an Execution Definition and link it to the test case folder created previously.

On the Setup/Cleanup tab add the Setup and Teardown cases respectively.

On the Assigned Test Definitions tab add the remaining test cases.

Don’t forget to click Apply.

## Execution Server requirements:

A Silk Execution server must be built properly.

In our cases I am using Server 2003 as it is compatible with the Silk Execution Services.

The only requirements are that there is a local folder that matches the Source Control settings for the project in Silk. The “working Folder” is a local folder on the Silk Execution Server that must be created.

All scripts will be synchronized from the “source control system” to this location.

The only other requirement is that ActiveState ActivePython be installed, as well as pywbem from SourceForge.