StormTest QC Integration Library

Installation and Programmer's Guide

S3 Group Document ID: ST-12006

Revision Date: Jan 2012

Product Version: 1.0

Contact us at http://www.s3group.com/tv-technology or stormtest-support@s3group.com

Contents

1 What is the StormTest QC Library?

The StormTest® QC library is a framework that enables interaction between HPQC (Hewlitt-Packard Quality Center) and StormTest test automation systems. This framework allows operators who enjoy the rich StormTest automation features to execute and report on test cases within HPQC. The interface requires no changes to the existing HPQC and StormTest hardware or software already deployed and does not require new skill sets for test operators or developers. The framework allows for a mixture of automation levels such as:

- Completely manual – this is a basic HPQC feature.
- Fully Automated – tests are simply "kicked off" from QC and reports are automatically updated upon completion.
- A mixture of manual and automation – where some portions of the test may await manual input from a test operator.

1.1 Required Skill Sets

Generally, there are two types of personnel interfacing with HPQC and StormTest. There are test developers who create the actual test case scripts and StormTest tests. These tests are then put to use by the second type of personnel, the test operators, whose responsibilities include executing and reporting results of the tests and scripts.

- Test writer: StormTest API familiarity and Python skills required in order to develop the StormTest Scripts. No knowledge of the HPQC internals required as S3 Group provides a Python Framework that hides the QC complexity when it comes to writing Python scripts (creating steps, using resources, attaching files, etc...)
- Test Operators: No new or extra skills required

2 Library Features

2.1 Resources

Users familiar with HPQC will be familiar with the resources application with HPQC which is essentially a repository for storing data and files that can be associated or referenced during the execution of test cases. The library takes advantage of this repository by using it to store resources used in the execution of automated tests. Examples of resources are: reference images, python scripts and MS excel spreadsheets that can be used as configuration files for scripts etc.

2.2 Attachments

The StormTest-HPQC Interface allows for attachment of resources and other data to Test Run, Test Step, Test Set or a Test Instance. This is useful for example when a script employs a screen capture or where the text received from an OCR capture is returned to a text file.

3 Installation

The first step is to identify a PC that can be used as the StormTest/QC client PC. This PC should meet the minimum requirements for both a StormTest client and a HPQC client.

To begin with, this PC should have no copies of HPQC software, StormTest software or Python installed.

3.1 Install the StormTest Client

Install the latest StormTest client software. The version installed should match the version on the StormTest server. StormTest client software can be downloaded from the StormTest engage website:

https://larisa.engage.s3group.com/frs/?group_id=6

As well as installing StormTest client, this will also install the correct version of Python (version 2.5).

3.2 Installing Required Quality Center Add-Ins

In order to be able to use python scripts in Quality Center, a certain number of Quality Center Add-Ins must be installed prior to installing the StormTest QC integration library. These add-ins are listed below:

- HP Quality Center Client Side Setup Add-in
- HP Quality Center Connectivity Add-in
- Quality Center Explorer Add-in

To install these Add-Ins, connect with Internet Explorer to your Quality Center Server's main page and click on the "Add-Ins Page" link (see Figure 1).

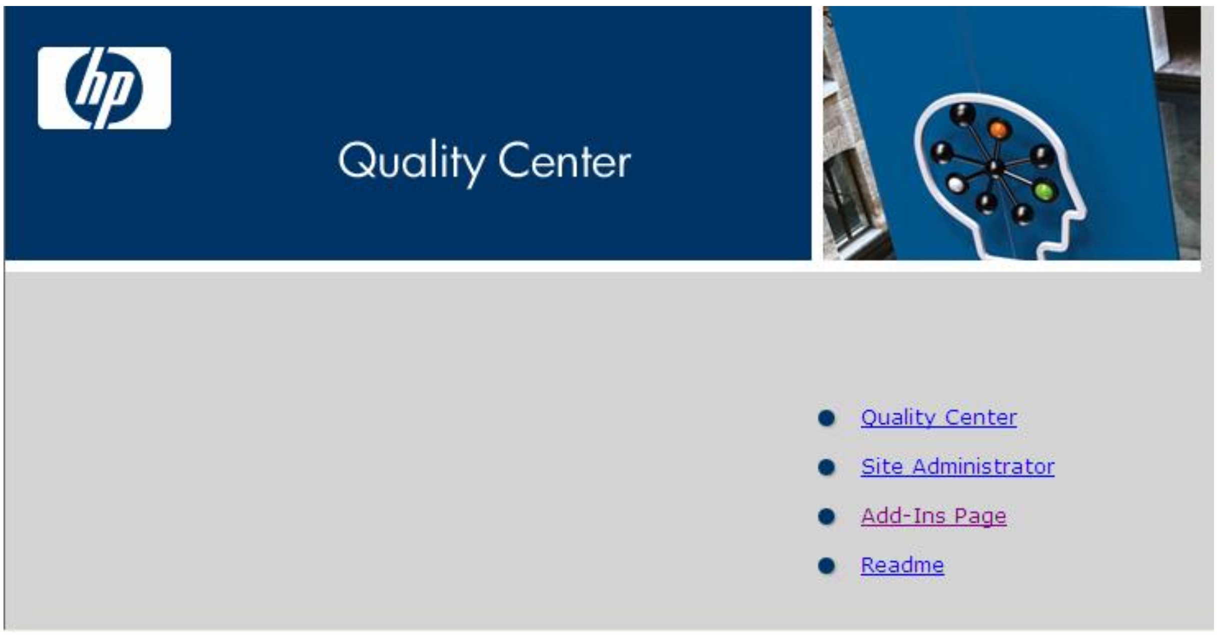
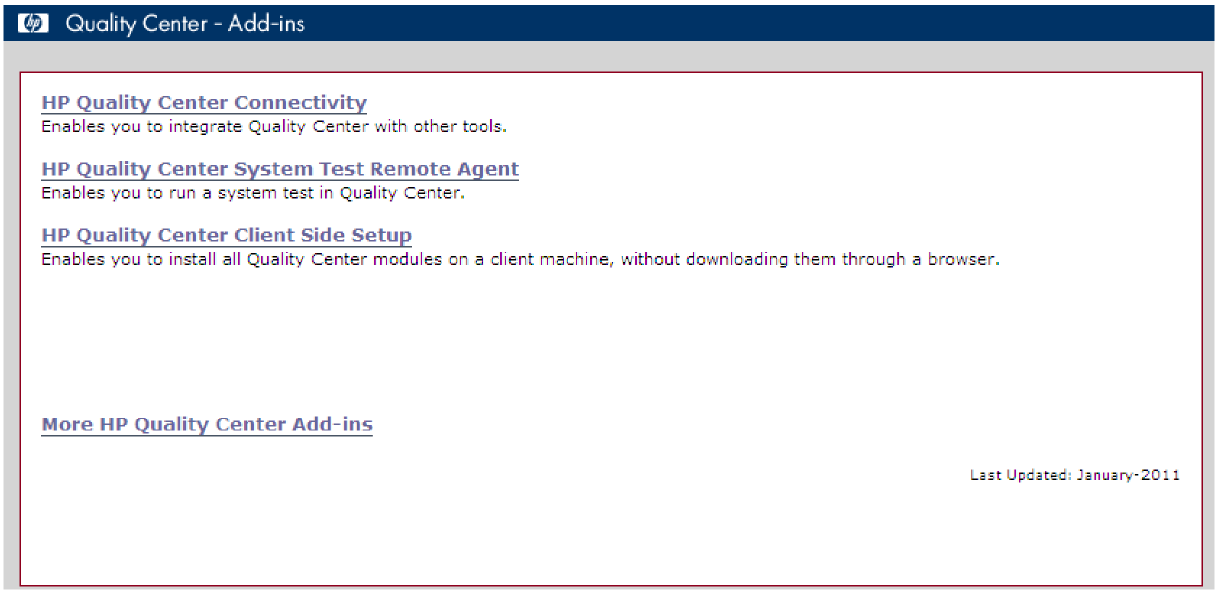


Figure 1: Quality Center Main Page

Once you have clicked the Add-Ins Page link, you should then be redirected to the main Quality Center Add-Ins page (see Figure 2)


Quality Center - Add-ins

HP Quality Center Connectivity
Enables you to integrate Quality Center with other tools.

HP Quality Center System Test Remote Agent
Enables you to run a system test in Quality Center.

HP Quality Center Client Side Setup
Enables you to install all Quality Center modules on a client machine, without downloading them through a browser.

More HP Quality Center Add-ins

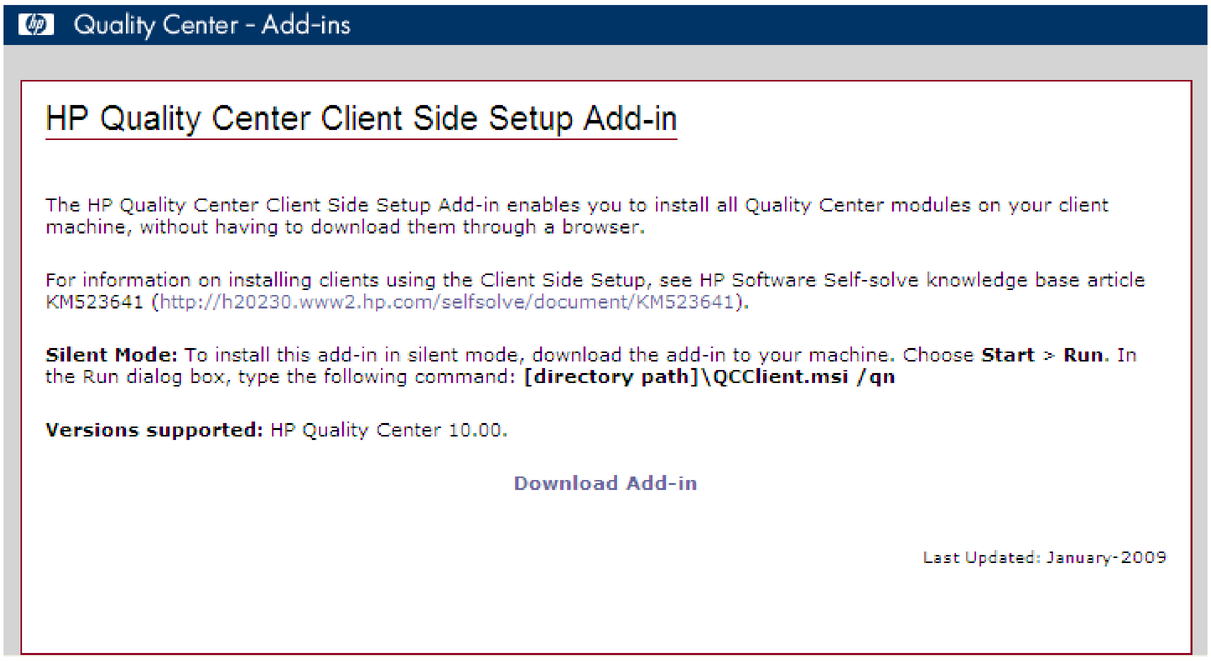
Last Updated: January-2011

Figure 2: Quality Center Add-ins page

This is the main Quality Center Add-Ins page.

3.2.1 Installing HP Quality Center Client

From the Quality Center Add-Ins page, click on the "HP Quality Center Client Side Setup" link which will then redirect you to the relevant installation page (Figure 3). All you have to do is click on "Download Add-In" and run the installer.


Quality Center - Add-ins

HP Quality Center Client Side Setup Add-in

The HP Quality Center Client Side Setup Add-in enables you to install all Quality Center modules on your client machine, without having to download them through a browser.

For information on installing clients using the Client Side Setup, see HP Software Self-solve knowledge base article KM523641 (http://h20230.www2.hp.com/selfsolve/document/KM523641).

Silent Mode: To install this add-in in silent mode, download the add-in to your machine. Choose **Start** > **Run**. In the Run dialog box, type the following command: **[directory path]\QCClient.msi /qn**

Versions supported: HP Quality Center 10.00.

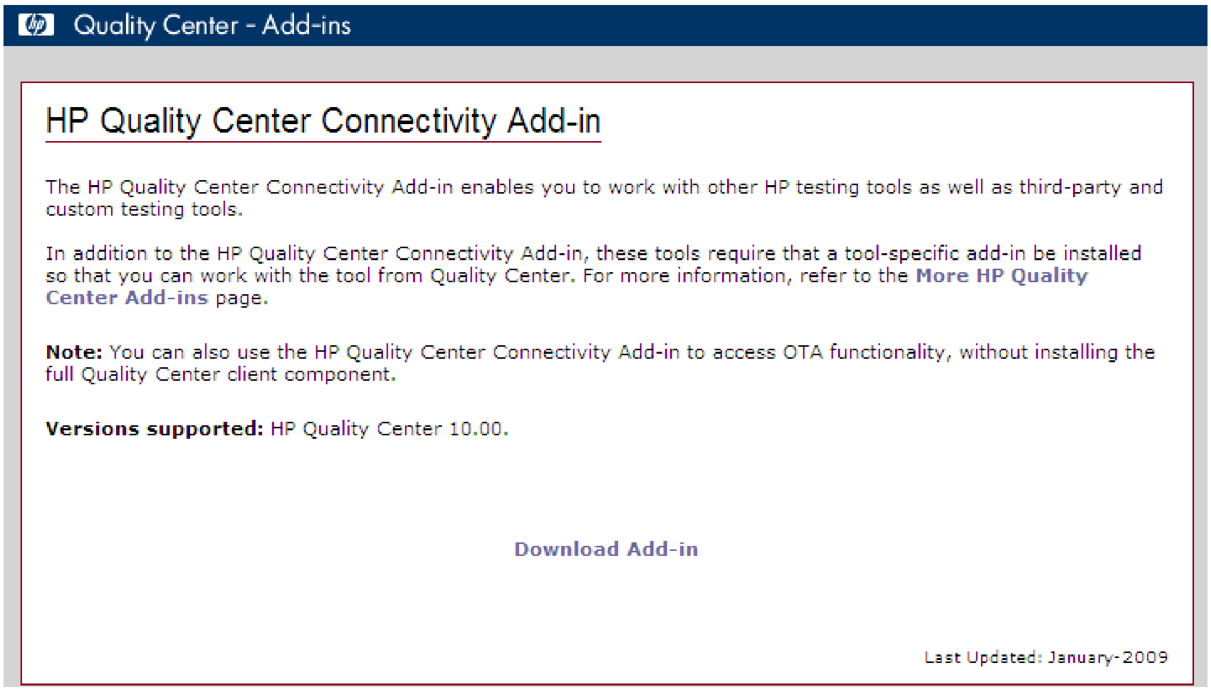
Download Add-in

Last Updated: January-2009

Figure 3: Quality Center Client Side Setup

3.2.2 Installing HP Quality Center Connectivity

As with the previous Add-In, return to the main Quality Center Add-Ins page and then click on “HP Quality Center Connectivity” link which will then redirect you to a page containing the Add-In for Quality Center Connectivity (see Figure 4). All you have to do is click on “Download Add-In” and run the installer.


Quality Center - Add-ins

HP Quality Center Connectivity Add-in

The HP Quality Center Connectivity Add-in enables you to work with other HP testing tools as well as third-party and custom testing tools.

In addition to the HP Quality Center Connectivity Add-in, these tools require that a tool-specific add-in be installed so that you can work with the tool from Quality Center. For more information, refer to the **More HP Quality Center Add-ins** page.

Note: You can also use the HP Quality Center Connectivity Add-in to access OTA functionality, without installing the full Quality Center client component.

Versions supported: HP Quality Center 10.00.

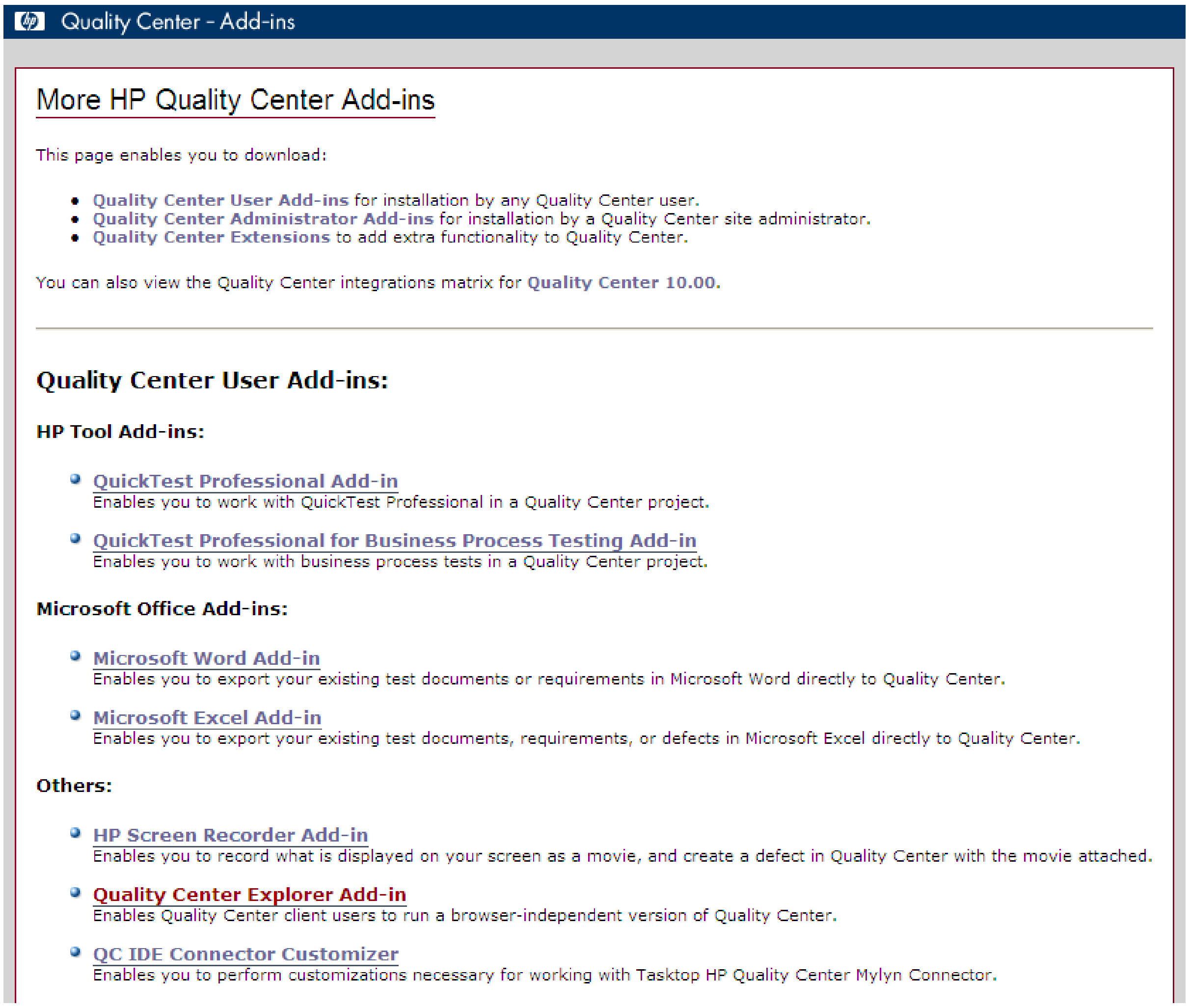
Download Add-in

Last Updated: January-2009

Figure 4: Quality Center Connectivity Add-In

3.2.3 Installing HP Quality Center Explorer

Return to the main Quality Center Add-Ins page and then click on the “More HP Quality Center Add-Ins” link which will then redirect you to a page containing all the extra Add-Ins for Quality Center (see Figure 5).



Quality Center - Add-ins

More HP Quality Center Add-ins

This page enables you to download:

- **Quality Center User Add-ins** for installation by any Quality Center user.
- **Quality Center Administrator Add-ins** for installation by a Quality Center site administrator.
- **Quality Center Extensions** to add extra functionality to Quality Center.

You can also view the Quality Center integrations matrix for **Quality Center 10.00**.

Quality Center User Add-ins:

HP Tool Add-ins:

- **QuickTest Professional Add-in**
  Enables you to work with QuickTest Professional in a Quality Center project.
- **QuickTest Professional for Business Process Testing Add-in**
  Enables you to work with business process tests in a Quality Center project.

Microsoft Office Add-ins:

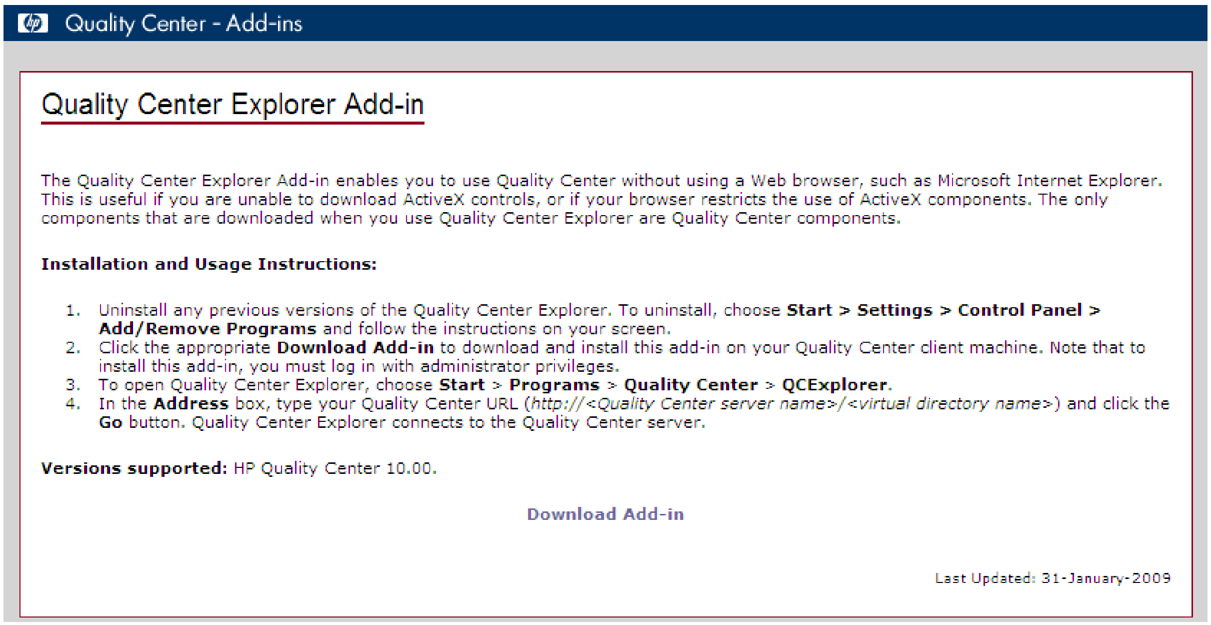
- **Microsoft Word Add-in**
  Enables you to export your existing test documents or requirements in Microsoft Word directly to Quality Center.
- **Microsoft Excel Add-in**
  Enables you to export your existing test documents, requirements, or defects in Microsoft Excel directly to Quality Center.

Others:

- **HP Screen Recorder Add-in**
  Enables you to record what is displayed on your screen as a movie, and create a defect in Quality Center with the movie attached.
- **Quality Center Explorer Add-in**
  Enables Quality Center client users to run a browser-independent version of Quality Center.
- **QC IDE Connector Customizer**
  Enables you to perform customizations necessary for working with Tasktop HP Quality Center Mylyn Connector.

Figure 5: More Add-Ins

Click on “Quality Center Explorer Add-In” and then “Download Add-In” when in the Quality Center Explorer page (see Figure 6).



Quality Center - Add-ins

Quality Center Explorer Add-in

The Quality Center Explorer Add-in enables you to use Quality Center without using a Web browser, such as Microsoft Internet Explorer. This is useful if you are unable to download ActiveX controls, or if your browser restricts the use of ActiveX components. The only components that are downloaded when you use Quality Center Explorer are Quality Center components.

Installation and Usage Instructions:

1. Uninstall any previous versions of the Quality Center Explorer. To uninstall, choose **Start > Settings > Control Panel > Add/Remove Programs** and follow the instructions on your screen.
2. Click the appropriate **Download Add-in** to download and install this add-in on your Quality Center client machine. Note that to install this add-in, you must log in with administrator privileges.
3. To open Quality Center Explorer, choose **Start** > **Programs** > **Quality Center** > **QCExplorer**.
4. In the **Address** box, type your Quality Center URL (*http://<Quality Center server name>/<virtual directory name>*) and click the **Go** button. Quality Center Explorer connects to the Quality Center server.

Versions supported: HP Quality Center 10.00.

Download Add-in

Last Updated: 31-January-2009

Figure 6: Quality Center Explorer Add-In

3.3 Verifying the Python installation

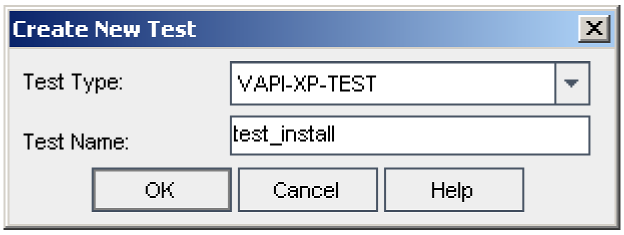
Launch “Quality Center Explorer”.

Enter the URL of the Quality Center Server (e.g. http://server:8080/qcbin/) and hit the “Go” button on the right.

You should now be redirected to the Quality Center log in page which you're already familiar with. Enter your name then authenticate, select your project and finally log in.

Go to the test plan, create a temporary dummy test whose only purpose is to test the Python installation in Quality Center.

In order to do so, create a test, select VAPI-XP-TEST as Test Type, give it a name and hit “OK”

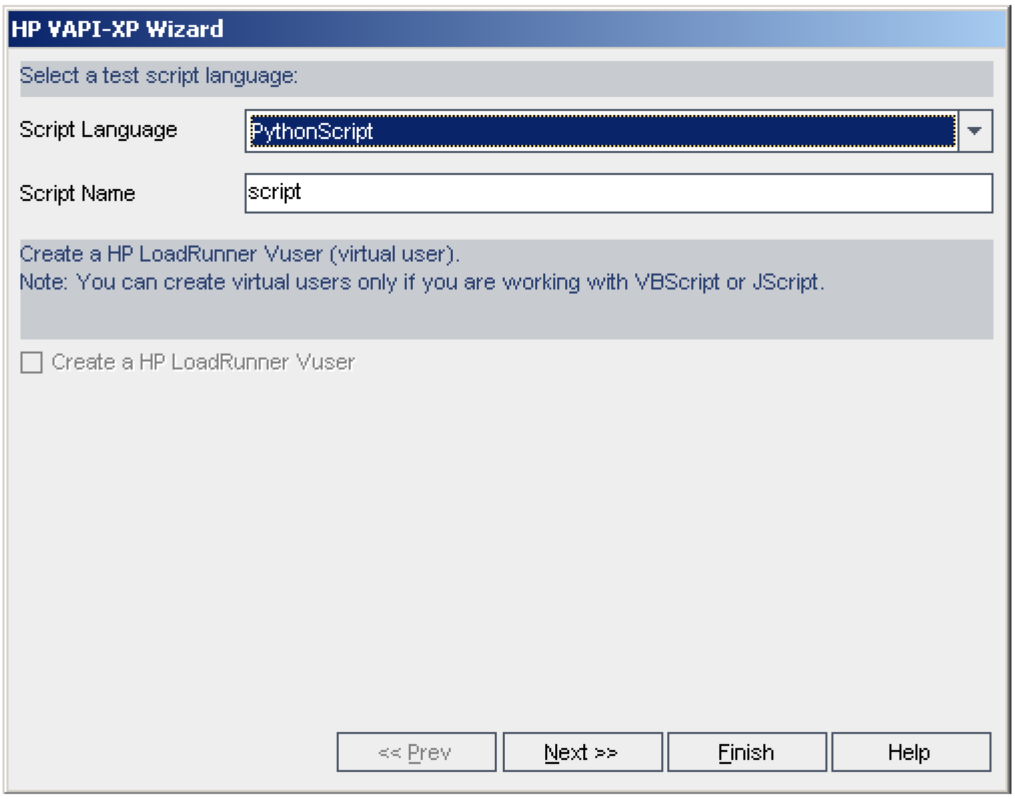


Create New Test

Test Type: VAPI-XP-TEST

Test Name: test_install

The VAPI-XP wizard then appears.



HP VAPI-XP Wizard

Select a test script language:

Script Language: PythonScript

Script Name: script

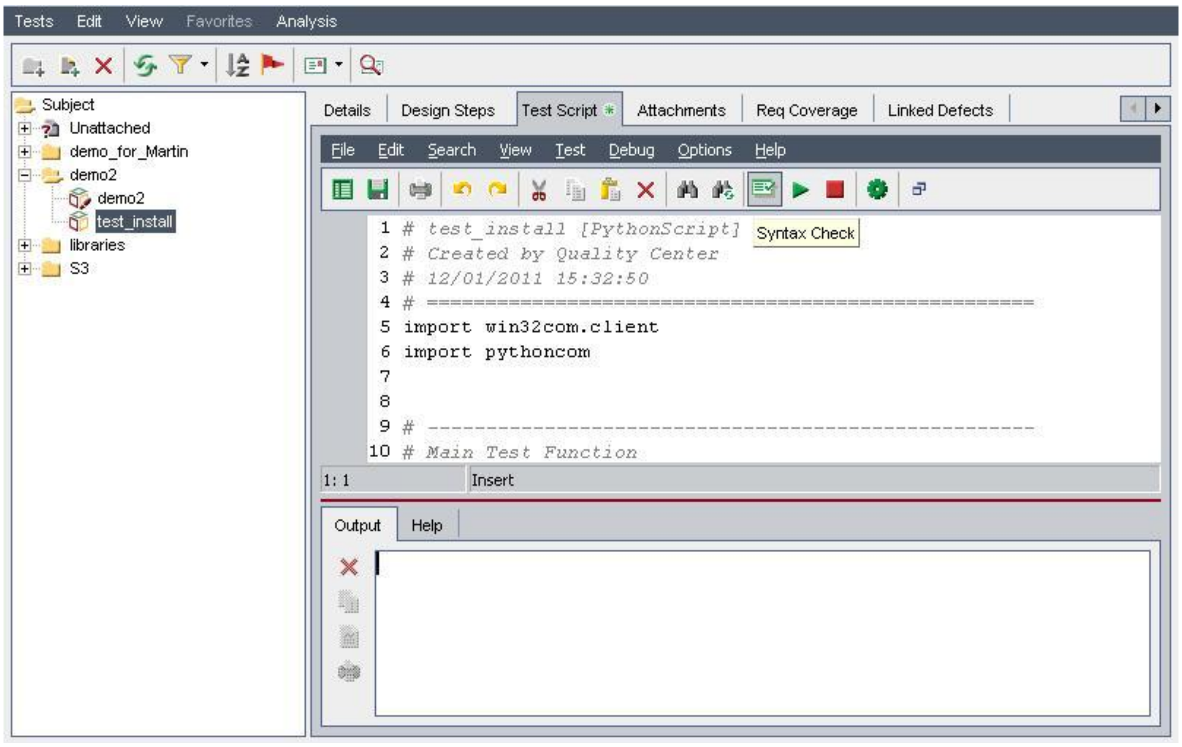
Create a HP LoadRunner Vuser (virtual user).
Note: You can create virtual users only if you are working with VBScript or JScript.

☐ Create a HP LoadRunner Vuser

Figure 7: VAPI-XP Wizard

Select PythonScript as Script language and hit the “Finish” button. The test is then created.

Open the test and select the "Test Script" tab. You should see a Python script template created by Quality Center.

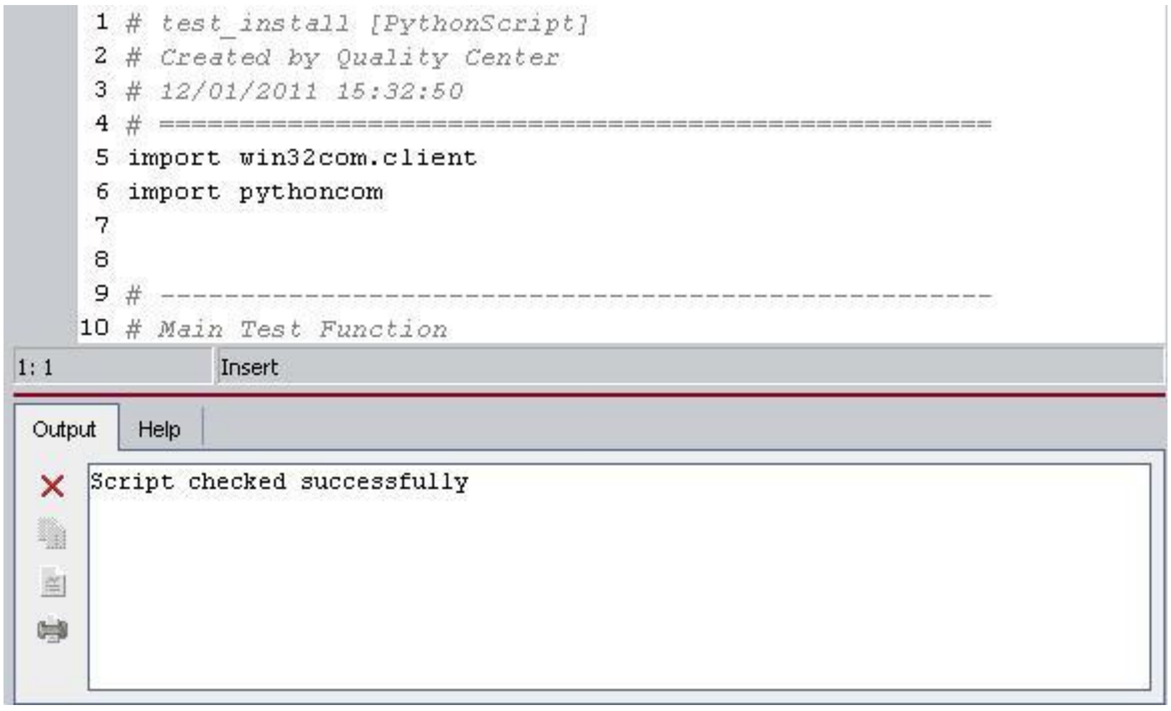

Tests Edit View Favorites Analysis
Subject
Unattached
demo_for_Martin
demo2
demo2
test_install
libraries
S3
Details Design Steps Test Script Attachments Req Coverage Linked Defects
File Edit Search View Test Debug Options Help
Syntax Check
1: 1
Insert
Output Help

```
# test_install [PythonScript]
# Created by Quality Center
# 12/01/2011 15:32:50
# ==========================================================
import win32com.client
import pythoncom


# ----------------------------------------------------------
# Main Test Function
```

Figure 8: Python Script Template

Hit the "Syntax Check" button of the script's toolbar and if the installation of the Quality Center Add-Ins is correct, "Script checked successfully" should appear in the bottom Output console.



```
# test_install [PythonScript]
# Created by Quality Center
# 12/01/2011 15:32:50
# ==========================================================
import win32com.client
import pythoncom


# ----------------------------------------------------------
# Main Test Function
```

```
Script checked successfully
```

Figure 9: HPQC Python Syntax Check

In case of any problem contact your Quality Center administrator for support.

3.4 Installing the StormTest QC Integration Library

You can now run the StormTest QC installer. This will install all the libraries needed to use the QC integration functionality in a StormTest script.

Note: If installing on Windows 7, there is a bug currently with HPQC10 and Windows 7. When you run a test, if you receive a message “Invalid Value For Registry”, you must follow the steps below to resolve:

1. Trigger "Invalid Value for Registry" by attempting to trigger QTP test from QC.
2. Open Control Panel > Administrative Tools > Event Viewer > Windows Logs > System
3. Find the most recent instance of a DistributedCOM (DCOM) error that looks like the following (values may or may not be the same)

   The launch and activation security descriptor for the COM Server application with APPID

   {6108A56C-6239-41F6-8C0F-94D9CE0D4B61}

   is invalid. It contains Access Control Entries with permissions that are invalid. The requested action was therefore not performed. This security permission can be corrected using the Component Services administrative tool.

4. Copy the APPID
5. Start > Run > Regedit
6. Select the very top item in the regedit tree (this will cause it to search the entire registry instead of starting from the currently selected item)
7. CTRL+F (or Edit > Find)
8. Paste in the APPID and start the search
9. Rename the AccessPermission and LaunchPermission keys to something else (example: AccessPermission_Backup and LaunchPermission_Backup)

4 Developing StormTest Scripts for HPQC

There are essentially two ways to develop scripts for execution under HPQC

1. Write tests from scratch to confirm to the QC view of testing, complete with test steps and attachments. Each test step can have its own status and its own attachments. We refer to this type of script as a "QC aware" script.
2. Take an existing StormTest script that works outside of QC and modify it to execute under QC. In this case, the status of each individual step cannot be recorded. Only the overall status of the test.

4.1 Developing "QC aware" scripts

There are several advantages to writing QC aware StormTest scripts:

- More user friendly to QA personnel who generally operate with the concept of a test containing multiple steps.
- Allows more granular reporting of test case pass/failure (which step of the test failed)
- Uses either a lookup table or a popup GUI to input the StormTest server/slot to be used. This means that the test operator does not need to be in any way familiar with StormTest or StormTest concepts.

To develop and run scripts for use in QC, first the HPQC-StormTest components must be installed as outlined in section 3 of this document.

Creating QC aware test scripts can be done within QC itself or within your own IDE.

We recommend that test scripts are developed in a python IDE, such as Eclipse and that the entire test is copy and pasted into the QC editor once it is ready to be run.

4.1.1 Creating the Script

Below is a sample StormTest Script that is QC aware. We will walk through this example script to describe how the script functions.

```
# ====================================================
import os
import stormtest_qc as qc
import stormtest.ClientAPI as st

testId = 'SimpleTest'

class demoStep(object):
    def __init__(self):
        self.name = 'This is a dummy step (demo)'
        self.description = 'Does nothing'
        self.expected = 'no expectations at all'

    def run(self, step, param):
        TDOutput.Print(str(param))
        St.WaitSec(5)    # StormTest API
        qc.attachments.attachFile(step, "filename")
        return (qc.Status.PASSED, 'nothing to do -> nothing done')

## Define the test steps and run the test.
def testRun(currentRun, testCommon, testEnv):
    # Create step factory used to define and run test steps
    sf = qc.steps.StepFactory(currentRun)
    ###############################################################
    # The test steps.
    ###############################################################
    # Step 1: demo step - does nothing so far
    # Parameter #1 :     <mandatory> User defined step class instance
    # parameter #2..n : [optional]  is/are passed to the run() method of the
user defined step class
    sf.createAndRun(demoStep(), "let's pass a string as parameter")

# Main test function.
def Test_Main(Debug, CurrentTestSet, CurrentTSTest, CurrentRun):
    if Debug:
        TDOutput.Print("!!! This test must be run from the Test Lab !!!")
        return None
    qc.framework.testMain(CurrentTestSet, CurrentTSTest, CurrentRun, TDOutput,
                        TDConnection, None, testId, testRun)

# If test is run from the command line, run the test using the QC simulator.
if __name__ == "__main__":

    # Run test in the simulator.
    qc.qcsim.RunSim(None, "sim_output", testId, Test_Main)
```

This first section of code imports the necessary modules to access the StormTest API and the StormTest QC library. We also assign a descriptive string to the testId variable.

```
# ====================================================
import os
import stormtest_qc as qc
import stormtest.ClientAPI as st

testId = 'SimpleTest'
```

In the next section, we create a python class in which we will define functions that will make up a “step”. In the `__init__` function (which will be run automatically when an instance of this class is created) we define a name, a description and a string called “expected” which can be used to describe the expected output of the test step when successful. The class also has a run method. The run method is where all the work for the step is carried out.

This particular run function also shows how a file can be passed back to QC. (This is useful for instance where a screen capture is taken during the test run). The run method will return a **qc.Status** which can be used to determine the success of the step.

```
class demoStep(object):
    def __init__(self):
        self.name = 'This is a dummy step (demo)'
        self.description = 'Does nothing'
        self.expected = 'no expectations at all'

    def run(self, step, param):
        TDOutput.Print(str(param))
        St.WaitSec(5)    # StormTest API
        qc.attachments.attachFile(step, "filename")
        return (qc.Status.PASSED, 'nothing to do -> nothing done')
```

The following section is where we begin to create HPQC steps and have HPQC call and execute those steps. The line:

```
sf = qc.steps.StepFactory(currentRun)
```

creates an instance of the qc step factory class. The step factory class is used to define and run steps. The next uncommented line:

```
sf.createAndRun(demoStep(), "let's pass a string as parameter")
```

calls the createAndRun method of the StepFactory class and we pass in our first step “demoStep” which was defined before as the user defined step class instance. We can create as many steps as we like using this process. First, define local functions that resemble test steps, then create a step factory object which will call those steps using the createAndRun method.

```
def testRun(currentRun, testCommon, testEnv):
    # Create step factory used to define and run test steps
    sf = qc.steps.StepFactory(currentRun)
    ################################################################
    # The test steps.
    ################################################################
    # Step 1: demo step - does nothing so far
    # Parameter #1 :    <mandatory> User defined step class instance
    # parameter #2..n : [optional]  is/are passed to the run() method of the
user defined step class
    sf.createAndRun(demoStep(), "let's pass a string as parameter")
```

The last section of code is the main test function which is used to kick off the test from QC

```
def Test_Main(Debug, CurrentTestSet, CurrentTSTest, CurrentRun):
    if Debug:
        TDOutput.Print("!!! This test must be run from the Test Lab !!!")
        return None
    qc.framework.testMain(CurrentTestSet, CurrentTSTest, CurrentRun, TDOutput,
                        TDConnection, None, testId, testRun)

# If test is run from the command line, run the test using the QC simulator.
if __name__ == "__main__":

    # Run test in the simulator.
    qc.qcsim.RunSim(None, "sim_output", testId, Test_Main)
```

In addition, there is a call to **QCSIM**. QCSIM is part of the StormTest QC library, that allows a QC aware test script to be run outside of QC. This is a very powerful tool when developing scripts as it allows you to use an IDE and debugger during script development, which is not possible if the script is executed in QC.

Those who are familiar with StormTest scripts will notice that there are no calls to the StormTest APIs ConnectToServer() or ReserveSlot(). This is because the StormTest QC framework can handle these calls automatically. For more information on this, see section 4.1.5.

4.1.2 Creating the test in QC

4.1.2.1 Login to HPQC

On your PC, open HPQC Explorer from your list of installed programs. When prompted, enter the URL for your HPQC server (Ex. http://servername:8080/qcbin/)and login credentials. You should be presented with a page similar to the one shown below.

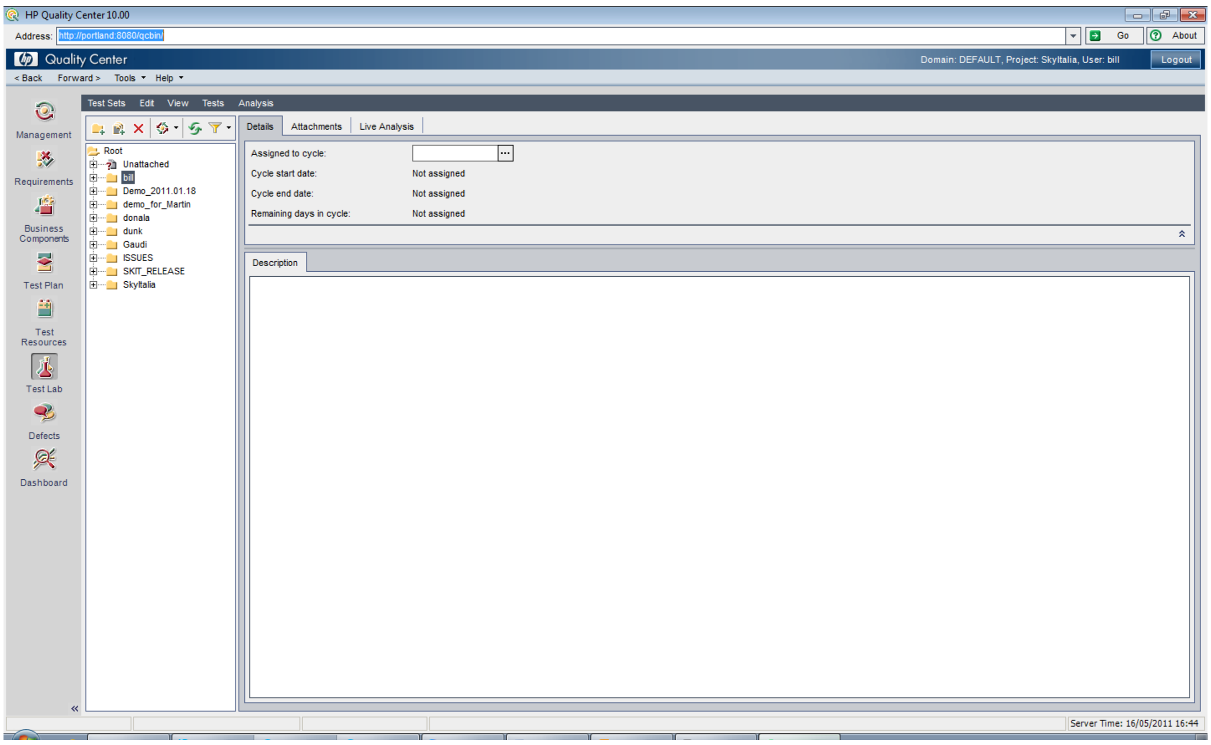


Figure 10: HPQC

4.1.2.2 Create a New Test

Tests are created in the HPQC Test Plan applet. To get started, click on "Test Plan" in the left panel of HPQC. If this is your first time using the HPQC system, it is suggested that you create a folder to store your tests and scripts in. To create a folder, click on the "Create New Folder" icon and follow the prompts to name the folder. Select the new folder and then click on the "Create New Test" icon . When prompted, select the "VAPI-XP-TEST" option and name the test. Then select OK.

Test Plan

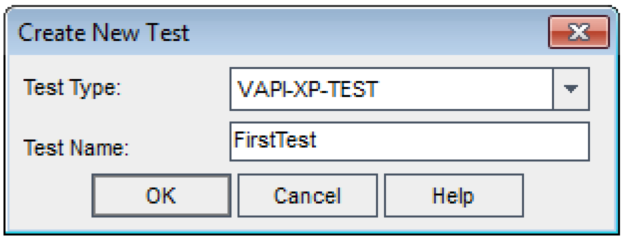


Figure 11: Create New Test

Once your test is created, you should be presented with the HPQC-VAPI-XP wizard to create your test script. The first prompt asks you to select the script type and name the script. StormTest uses Python so select Python from the Script Language drop down menu, name the script and select "Finish".

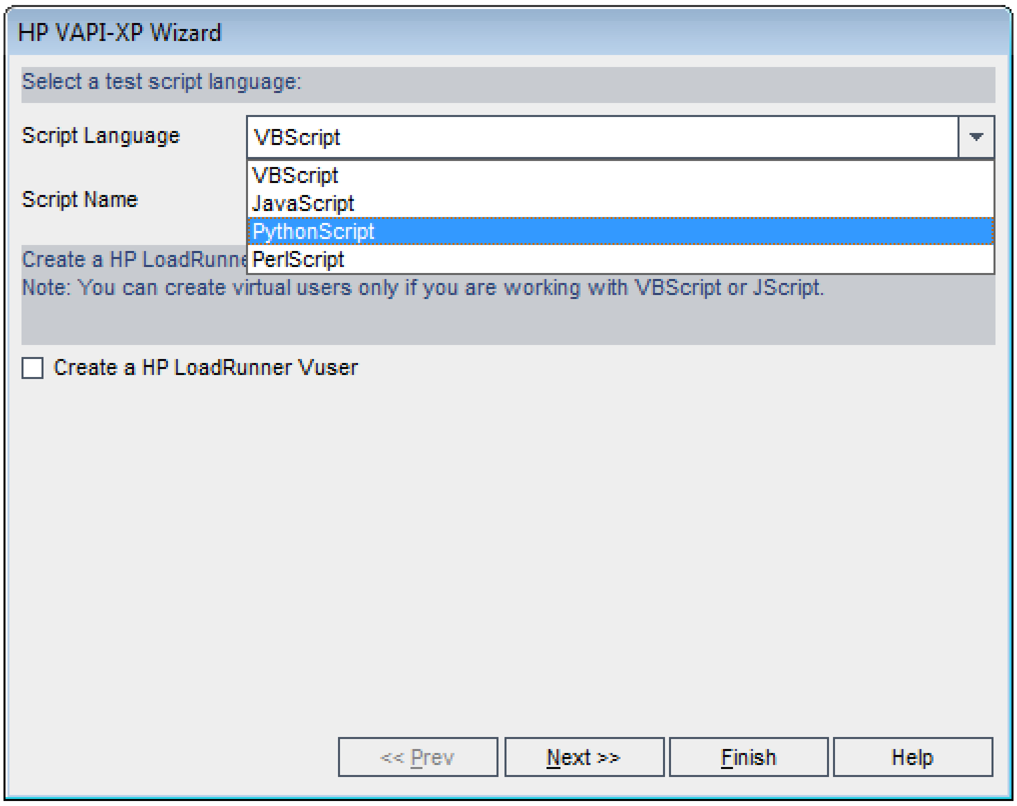
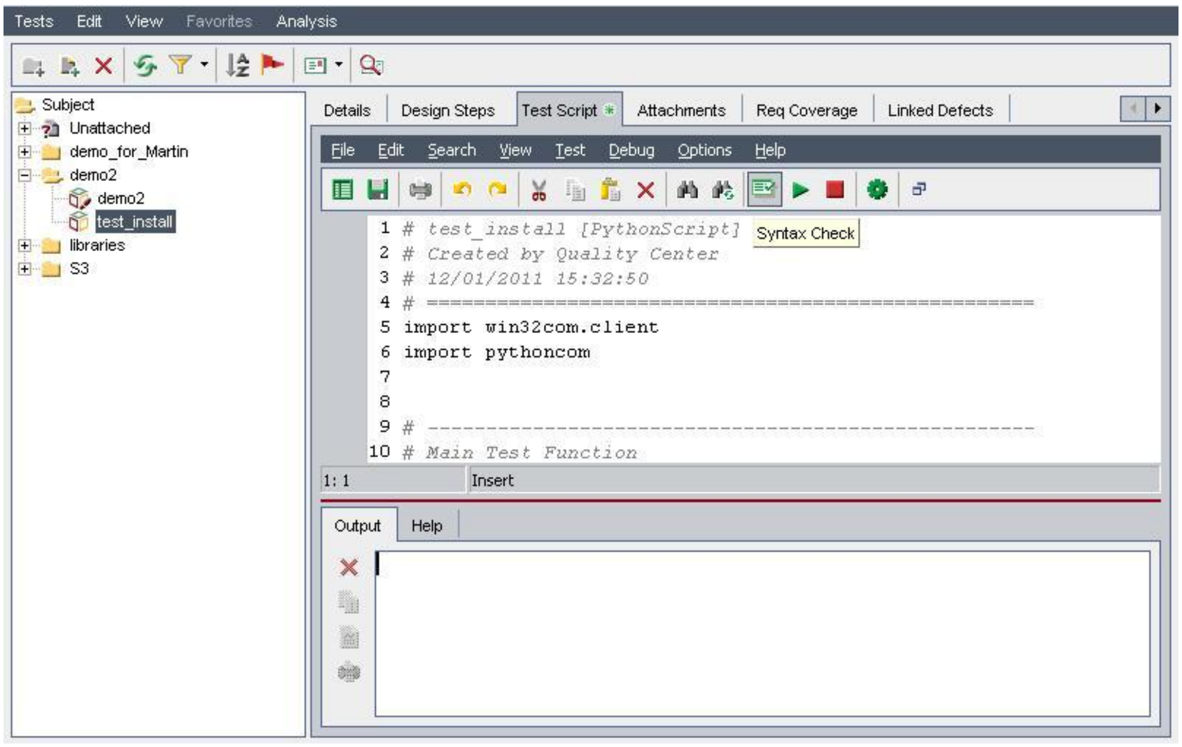

HP VAPI-XP Wizard
Select a test script language:
Script Language
VBScript
Script Name
VBScript
JavaScript
PythonScript
PerlScript
Create a HP LoadRunne
Note: You can create virtual users only if you are working with VBScript or JScript.
Create a HP LoadRunner Vuser
<< Prev
Next >>
Finish
Help

Figure 12: HPQC VAPI-XP Wizard

Once your script has been created, you can click on the new test to select it and then click on the "Test Script" tab within HPQC to view the newly created Python test script template. This template will require some manual modification to allow HPQC to interface with the StormTest server.



```
# test_install [PythonScript]
# Created by Quality Center
# 12/01/2011 15:32:50
# ====================================================
import win32com.client
import pythoncom


# ----------------------------------------------------
# Main Test Function
```

Figure 13: HPQC Python Script Template

Rather than editing the test directly in the QC explorer, it is better to delete the text in the QC explorer and paste in a complete test from another python editor/IDE.

4.1.3 Creating a Test Plan and Running your Test

Before a script can be run from within HPQC, the script must be made a part of a test set. The test lab is where you create and execute test sets which are collection of tests or scripts. To create a new test set in HPQC's Test Lab, click on the Test Lab Icon to enter the Test Lab application. If this is your first time creating a test set within HPQC, it is suggested that you create a folder that will contain your test sets.

The folder is created by clicking on the new folder icon .

Click on the folder to select it and then click on the new test set icon to create a new test set.

Once the test set has been created, the Test Lab app should open a frame that shows the test plan folder tree as in Figure 14 below. From the Test Plan tree, select the test you created previously and drag-n-drop it onto the test plan execution grid. Once the test is added to the test set, you can now run the test by clicking on the Run icon .

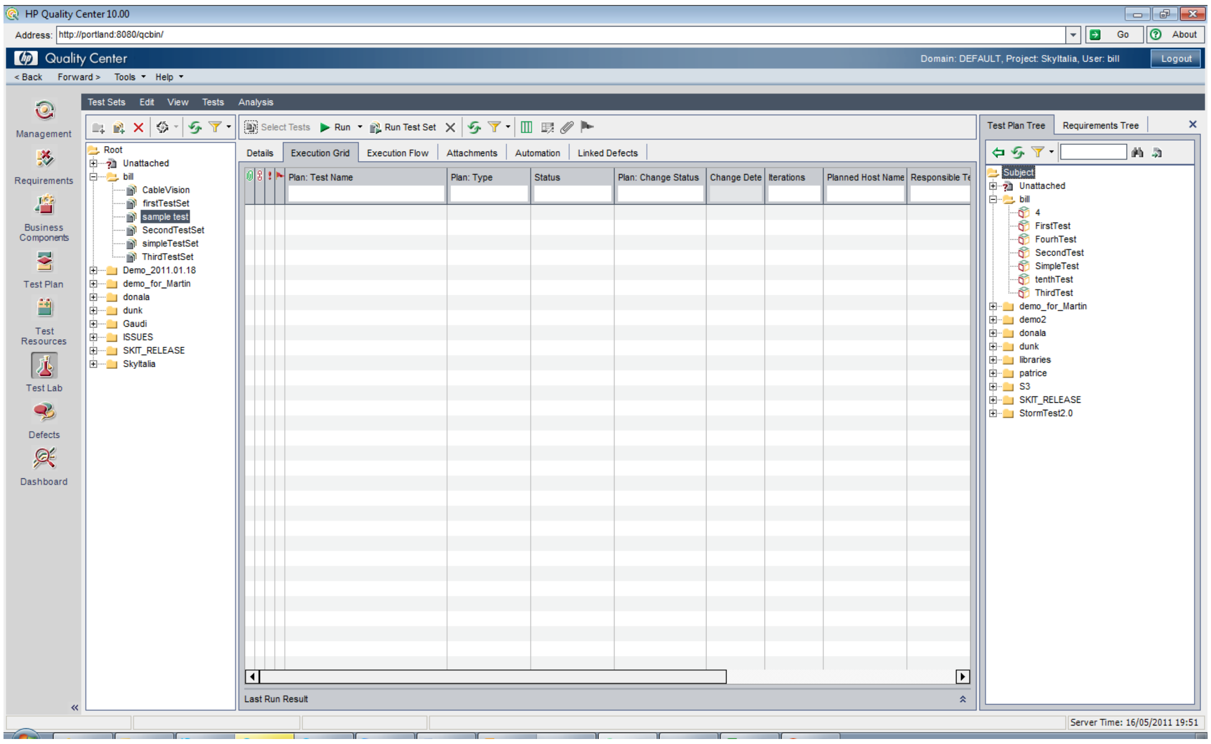
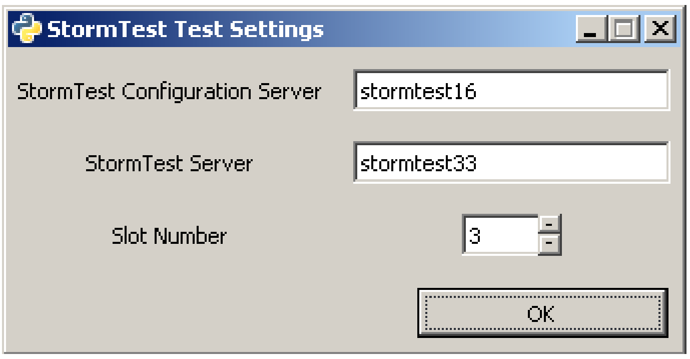


Figure 14: HPQC Test Lab

4.1.4 Running the test in QC

When the test is run from Test Lab, the user will be prompted to define the StormTest configuration server, StormTest server and slot that the test should be run against. Once the information is entered, press OK and the test will be executed. A report of the execution will be automatically generated within HPQC.



StormTest Configuration Server: stormtest16

StormTest Server: stormtest33

Slot Number: 3

Figure 15: Server, Slot prompt

The data entered into this GUI will be stored and presented to the user again if the test is run a second time.

4.1.5 Defining the StormTest parameters in an excel file

The popup GUI described above is a useful way to input the StormTest resource to be used by the test script. However, since the test operator is likely to know nothing about StormTest, there is a method of defining these parameter in an excel file. In this way, the StormTest slot resources to be used by the tests can be defined by another user and this excel file can be uploaded to QC.

Once it is a QC resource, the StormTest QC library will automatically consult it. If the test being executed has a corresponding entry in the excel file, then the data in that file will be used instead of popping up the GUI.

Name	StormTest Configuration Server	StormTest Server	Slot(s)	Arguments
example_test_case1	stormtest16	stormtest33	4,5,6	

Table 1: StormTest Slot allocation excel sheet

In Table 1, you can see an example of the table that should be created.

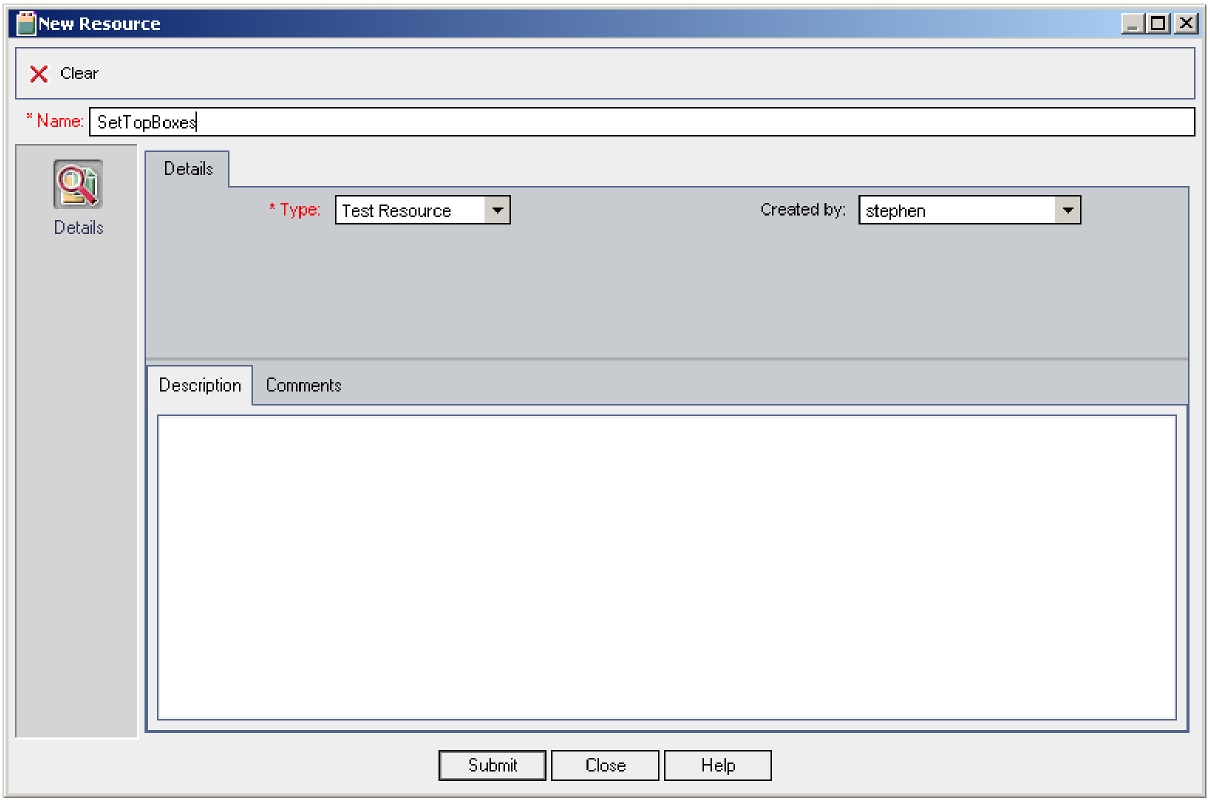
Note that all five columns are necessary and must match the naming above. Also, the worksheet must be names **StormTest**. Any other sheets in the excel file are ignored along with other columns. Also, the excel sheet must be in .xls format (Microsoft Excel 97-2003 workbook). The newer .xlsx file format will not work.

The test name (example_test_case1 above) must match the name of the test in Test Plan.

If multiple slots are listed (4,5,6 above), then the test will attempt to execute on each of those slots, serially.

4.1.5.1 Uploading the excel workbook

Once your excel workbook is created, you must upload it as a test resource. Click on "Test Resources" in QC explorer and create a new resource. The location in the tree is unimportant.


New Resource

Clear

* Name: SetTopBoxes

Details

* Type: Test Resource

Created by: stephen

Figure 16: Create new resource

The resource must be called **SetTopBoxes** and the type should be set to **Test Resource**.

When it is created, click on ‘Resource Viewer’ and on ‘Upload File’. Then point it at your excel file.

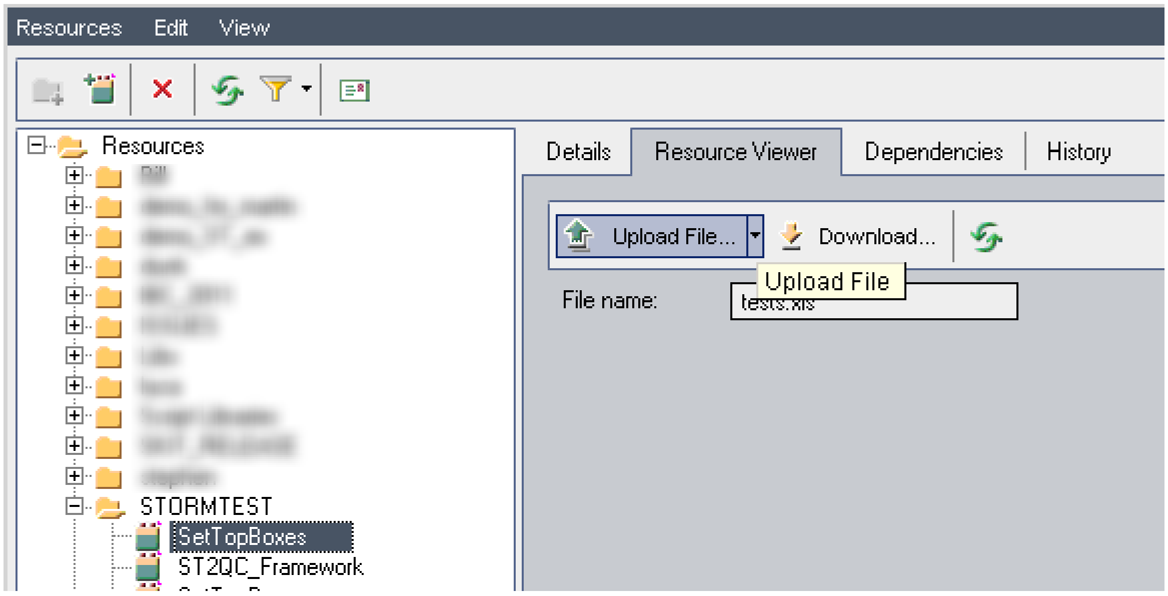


Figure 17: Upload a new resource

A template for this excel sheet is included with the StormTest QC library installer.

4.1.6 Printing messages in QC

All output from the test script to the console that is printed using the python print command is swallowed up when the test is run in QC.

If you wish your test output to be visible in QC during execution, then you must always use the TDOutput object. For example:

TDOutput.Print("Step One")

Will send the string “Step One” to the VAPI-XP output window

4.2 Running a Pre-Developed StormTest Script in HPQC

If you have existing StormTest scripts that wish to execute in QC, there are some small changes required to enable this.

First, the entire test should be enclosed in a python function. This should not be an issue, as accepted best practice is to use functions. For example, an existing StormTest test might look like this:

```
import stormtest.ClientAPI as ST

def main(argv):
    testResult = ST.TM.PASS
    ST.ConnectToServer('stormtest33')
    ST.ReserveSlot(2, 'default')
    ST.AddObserve("WARN")
    ST.PressButton("Power")
    ST.WaitSec(1)
    ST.PressButton("Power")
    ST.WaitSec(1)
    ST.ReleaseServerConnection()
    return testResult

if __name__ == '__main__':
    ret = main(sys.argv[1:])
    ST.ReturnTestResult(ret)
```

To modify this test to work in QC, we must make the following changes:

```
import stormtest.ClientAPI as ST
import stormtest_qc as qc

~~def main(argv):~~
def main(currentRun, testCommon, testEnv):
    testResult = ST.TM.PASS
    ST.ConnectToServer('stormtest33')
    ST.ReserveSlot(2, 'default')
    ST.AddObserve("WARN")
    ST.PressButton("Power")
    ST.WaitSec(1)
    ST.PressButton("Power")
    ST.WaitSec(1)
    ST.ReleaseServerConnection()

    if testResult == ST.TM.PASS:
        currentRun.Status = "Passed"
    else:
        currentRun.Status = "Failed"

    return testResult

def Test_Main(Debug, CurrentTestSet, CurrentTSTest, CurrentRun):
    result = qc.framework.testMain(CurrentTestSet, CurrentTSTest, CurrentRun,
                TDOutput, TDConnection, # These are QC global objects
                None, testId,           # No need to change these
                main,                   # This is the function that contains the
                                        # existing test
                debug=False,            # Set to True if you wish to see StormTest
                                        # log messages
                reserveSlots=False)     # Tells the framework that the test code
                                        # will handle the slot reservation

if __name__ == '__main__':
~~    ret = main(sys.argv[1:])~~
~~    ST.ReturnTestResult(ret)~~
    qc.qcsim.RunSim(None, "sim_output", testId, Test_Main)
```

This test code can now be copied into QC explorer and executed as described in section 4.1. The status, as set in the **currentRun.Status** flag will be logged to QC.

It's important to note that the changes made above mean that the test status is **not** posted to the StormTest database. It is only posted to the QC database. If you wish the status to be posted to the StormTest database also, then you must add the following lines to the **Test_Main()** function:

```
    try:
        ST.ReturnTestResult(result)
    except SystemExit:
        pass
```

5 Advanced Topics

5.1 Critical versus non-critical steps

By default, if a test step does not pass, then subsequent steps will not run and will be set to status ‘Not Run’. However, it may be the case that a step is not critical and you wish the test to continue, even if that step fails.

To do this, you must set the **critical** flag to False. An example of a non-critical step is shown below:

```
class nonCriticalStep(object):
    def __init__(self):
        self.name = 'This is a non-critical step'
        self.description = 'We don’t mind if this fails'
        self.expected = 'no expectations'
        self.critical = False

    def run(self, step, param):
        ...
```

5.2 Altering the IR database or serial parameters

When the StormTest QC library is used to reserve the slot for the test, the default IR database for the STB in that slot is used. In addition, the serial connection to the STB is not opened.

To change the IR database, you should open the StormTest Admin Console web page and change the IR database assigned to that STB model. However, this will alter it for all instances of that model. If, for some reason, you just wish to use an alternative IR database for this test, then you need to configure this parameter before starting the framework.

Similarly, if you wish to capture the serial output from the STB or interact with the serial port, then you need to pass in the appropriate serial parameters.

For example:

```
def Test_Main(Debug, CurrentTestSet, CurrentTSTest, CurrentRun):

    # Use an alternative IR database
    qc.framework.IrDatabase='Another_remote.txt'

    # Configure serial parameters (see the StormTest client API doc for the format)
    qc.framework.SerialParams=[19200,6,'none',1,'none']

    qc.framework.testMain(CurrentTestSet, CurrentTSTest, CurrentRun, TDOutput,
                          TDConnection, None, testId, runTest)
```

5.3 Passing arguments from the excel file

It is possible to pass a user-defined string to any test script run by the StormTest QC framework. The 'Arguments' column in the excel spreadsheet is used for this purpose. Any text string entered into this column for a particular test will be read and stored for use by the test.

To access this string, use the **qc.framework.arguments** variable. For example:

```
def runTest(currentRun, testCommon, testEnv):
    print "Arguments passed to this test are:",qc.framework.arguments
    ...
```

5.4 Storing reference images in QC

Reference images used in StormTest scripts can be stored in the QC test resource library, in the same way as the excel sheet mentioned earlier.

When the test executes, these images must be downloaded from QC to a temporary location.

The StormTest QC library provides routines to do this. We recommend that the test includes a python dictionary containing each reference image. The key in this dictionary is the image name in the QC resource library and the value is the local location where the image is stored. For example:

```
refImages = {
               'tv_guide':None,
               'menu':None
            }

for image in refImages:
    localPath = qc.resources.downloadResource(testEnv.tdConnection, name=image)
    if localPath == "":
        TDOutput.Print("Resource '%s' not found" % image)
    refImages[image] = localPath
```

We recommend that all reference images are downloaded at the start of a test run, for the sake of efficiency. Ideally, a separate test step will be defined and the function of this test step is simply to download the test resources.

5.5 Using resources in the QC Simulator

As mentioned in section 4.1.1, the StormTest QC library includes a QC simulator to aid in the development of test scripts. Using the simulator is as simple as adding a call to **qc.qcsim.RunSim()** to your script. This has been shown in the examples earlier in this document.

However, as well as simulating the QC COM objects, such as TDOutput and TDConnection, QCSIM can also simulate the QC test resources.

For example, if you are using the excel sheet to specify which slots to use, then you may also wish to use this file while using QCSIM. To do this, you must copy the excel file to a local directory. Then define a python dictionary and pass it to QCSIM. See example below:

```
    resources = [ { 'RSC_NAME': 'SetTopBoxes',
                    'RSC_FILE_NAME': os.path.join('resources', 'tests.xls'),
                    'RSC_ID': 1 }  ]

    # Run test in the simulator.
    qc.qcsim.RunSim(resources, "sim_output", testId, Test_Main)
```

The RSC_NAME is the name of the resource as it appears in QC. RSC_FILE_NAME is the location of the file on the local drive. RSC_ID, can be any number, but should be unique if you have multiple resources. If reference images are stored in the QC resource library, then they should be listed in the QCSIM resource dictionary also.

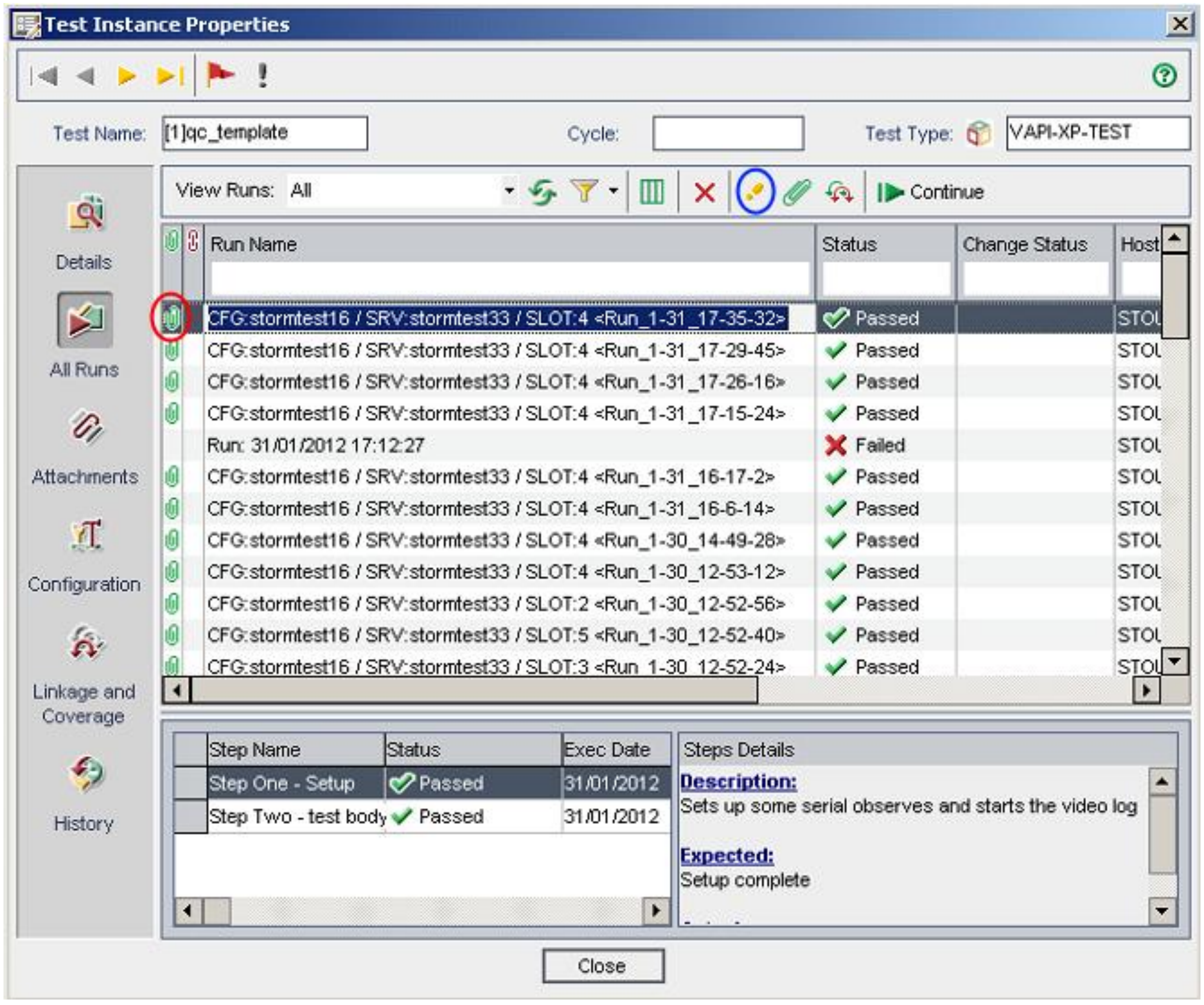
5.6 Attaching artefacts to test steps

Any artefact can be attached to a test step or a test run while executing a QC test. This is simply achieved by calling the **qc.attachments.attachFile()** function. For example, this code attaches a screen capture to a test step:

```
class AStep(object):
    def run(self,step):
        result = ST.CaptureImageEx (None, “capture.jpg”)
        if result[0][1] == True:
            # Attach the file with current step:
            qc.attachments.attachFile(step, result[0][3])
```

To attach an artefact to a test run, instead of a step, pass the **currentRun** object to the **attachFile()** function instead of the **step** object.

To view the attachments, open the test instance properties of the test in QC explorer. Click on the paperclip icon to view attachments to the test run – see red circle in Figure 18 below.


Test Instance Properties

Test Name: [1]qc_template
Cycle:
Test Type: VAPI-XP-TEST

View Runs: All

Run Name	Status	Change Status	Host
CFG:stormtest16 / SRV:stormtest33 / SLOT:4 <Run_1-31_17-35-32>	Passed		STOL
CFG:stormtest16 / SRV:stormtest33 / SLOT:4 <Run_1-31_17-29-45>	Passed		STOL
CFG:stormtest16 / SRV:stormtest33 / SLOT:4 <Run_1-31_17-26-16>	Passed		STOL
CFG:stormtest16 / SRV:stormtest33 / SLOT:4 <Run_1-31_17-15-24>	Passed		STOL
Run: 31/01/2012 17:12:27	Failed		STOL
CFG:stormtest16 / SRV:stormtest33 / SLOT:4 <Run_1-31_16-17-2>	Passed		STOL
CFG:stormtest16 / SRV:stormtest33 / SLOT:4 <Run_1-31_16-6-14>	Passed		STOL
CFG:stormtest16 / SRV:stormtest33 / SLOT:4 <Run_1-30_14-49-28>	Passed		STOL
CFG:stormtest16 / SRV:stormtest33 / SLOT:4 <Run_1-30_12-53-12>	Passed		STOL
CFG:stormtest16 / SRV:stormtest33 / SLOT:2 <Run_1-30_12-52-56>	Passed		STOL
CFG:stormtest16 / SRV:stormtest33 / SLOT:5 <Run_1-30_12-52-40>	Passed		STOL
CFG:stormtest16 / SRV:stormtest33 / SLOT:3 <Run_1-30_12-52-24>	Passed		STOL

Step Name	Status	Exec Date
Step One - Setup	Passed	31/01/2012
Step Two - test body	Passed	31/01/2012

Steps Details

Description:
Sets up some serial observes and starts the video log

Expected:
Setup complete

Figure 18: Test instance properties

To view attachments to test steps, click on the step icon – see the blue circle in Figure 18.

5.7 Enabling debug and video window

By default, the StormTest QC library will take the following actions when **qc.framework.testMain()** is called:

- Attempt to reserve a slot
- Display the video from that slot
- Disable the StormTest INFO messages.

We have already seen how the slot reservation by the library can be disabled by passing

```
reserveSlots = False
```

to **testMain()**. In a similar way, the user can control the other behaviour by using the following optional arguments to **testMain()**:

```
showVideo = False          # Disable the video window

debug = True               # Enable full StormTest messages
```

5.8 Import python file from QC resource library

It is quite possible to store python files in the QC resource library, as well as reference image files. While these can be downloaded using the same function as see above, there is a separate function available to directly import a python file from the QC resource library. For example:

```
testCommon = qc.resources.importResource(TDConnection, name='test_common')

# Now the testCommon object can be used to access all items defined in
# that module in the usual way, e.g.:
testCommon.someFunction()
```

This will import the python file stored as 'test_common' in the QC resource library. Once this line is executed, it can be used in the same way as if you had the file local and had called 'import'