Programmer's Guide

Accenture StormTest Development Center

Document ID: ST-11010

Revision Date: December 2016

Product Version: 3.4

Web: http://www.accenturestormtest.com

Contents

1 Preface

1.1 Accenture StormTest

Accenture StormTest Development Center is the leading automated test solution for digital TV services. It is designed to reduce the cost of getting high quality digital TV services to market faster.

StormTest Development Center greatly reduces the need for time-consuming, expensive and error-prone manual testing and replaces it with a more accurate and cost-effective alternative. It scales easily to large numbers and types of devices and integrates with existing infrastructure to give much greater efficiency in testing. It can be used to verify and validate services on a virtually every piece of consumer premises equipment (CPE), from set-top boxes to games consoles and from iPads to Smart TVs. It has been specifically designed to meet the needs of developers and testers of these CPE devices and the applications which run on them.

StormTest Development Center consists of:

- A choice of hardware units that can test 1, 4, or 16 devices. Each device under test can be controlled individually and independently and the audio/video from each device can be captured and analysed to determine the outcome of the test. The StormTest hardware supports capture of audio and video over HDMI interfaces and supports all HD resolutions up to 1080p. In addition there is a hardware upgrade option for the 16 device tester that will allow native capture of UHD content.
- Server software that controls all the hardware and devices in the rack as well as managing a central repository of test scripts and a central database of test results.
- A Client API that allows test scripts to interact with the server software
- A number of graphical tools that allow the user to directly control devices connected to StormTest Development Center, to create and schedule tests to run and to view the results of those test runs.

Test scripts can be run from any location – the tester needs only a network connection to the StormTest server. Video and audio output from the devices under test can be streamed over this network to any location, allowing remote monitoring and control of testing, either within a company LAN or across a WAN. Alternatively, scheduled tests can run directly on the server, negating the need for maintaining a continuous network connection to the StormTest server.

1.2 About This Document

This document describes how to use the StormTest Python API to write powerful, flexible test scripts.

It can be used by a script writer to learn how to use the StormTest API by following short, easy test script examples.

It is expected that the reader has a basic grasp of writing python code and knows how to run a python script – this document is **not** a guide to python. (However if you want python information you can check http://docs.python.org/release/2.7).

For a better understanding of how to use StormTest, the document has been split into the following chapters. We recommend going through this document chapter by chapter.

- An easy script to show you how to initialise and uninitialise your StormTest system.
- Time to Watch TV: This chapter shows you how to use a video window.
- Tune to Your Favourite Channel: This chapter shows you how to send infrared commands to your STB.
- Getting a Bigger Picture: This chapter shows you how to modify the video parameters.
- My Own PVR: This chapter shows you how to capture a video.
- Turn Off the TV: This chapter shows you how to power cycle a STB with StormTest.
- And Yet It Moves: This chapter shows you how to detect motion in a video.
- Spot the Differences: This chapter shows you how you can compare your screen to a reference image.
- The Black Sheep: This chapter shows you how to use some colour related functions.
- Are You Talking To Me?: This chapter shows you how to use the serial logging functionality.
- Reading from the screen: This chapter shows you how to read information from the screen.
- Timing a Channel Change: This chapter shows you how to measure timing for a channel change.

1.3 Who Should Read This Document

This document is targeted at the following groups:

- Test writers.
- Those running tests, who wish to gain an understanding of why a test is written in a particular way. This may aid in the interpretation of test failures.

1.4 Related Documentation

The StormTest Development Center user documentation set comprises of the following documents:

1) StormTest Developer Suite User's Manual
2) StormTest Programmer's Guide
3) StormTest Client API
4) StormTest Hardware Installation Guides (HV01, HV04, HV16)
5) StormTest Software Installation Guide
6) StormTest Server Monitor User's Manual
7) StormTest Administration Console User's Guide
8) StormTest Administration Tools User's Guide

The latest version of these documents can always be found on our support website, in the "Docs" section: https://larisa.engage.s3group.com/docman/?group_id=6.

2 Supporting Features

Before we begin to write our code and run our tests, it is important that we are familiar with two particular features of StormTest Development Center that can help to speed up the development as well as the updating of tests. The features in question are as follows:

- StormTest Navigator
- Resources

These features are very powerful in that they allow us to significantly reduce the number of lines of code that we will have to write and maintain. This will be illustrated with practical examples later in this document.

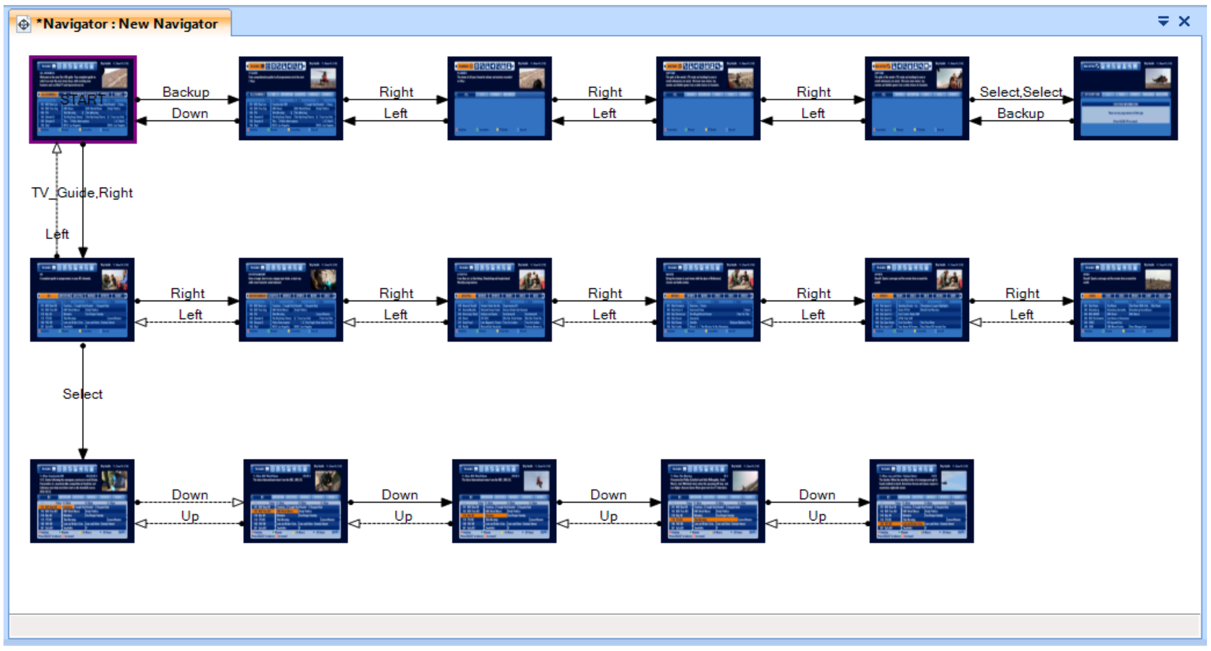
Also, when we use these features we will have the ability to quickly react to any changes that are made to menus, logos, icons, channel names or colour schemes associated with the DUT (Device under Test). Changes can be made to the Navigator or Resource in question and it will not be necessary to make changes to your code.

We will now look at these two features in more detail.

2.1 The StormTest Navigator

The StormTest Navigator is a component that makes advanced screen navigation features available to test developers. This component is designed to simplify the writing of tests using StormTest.

The navigator file holds all the screens that make up the navigation area of a DUT. It knows how to navigate between screens using the IR key presses needed and how to determine if a screen is correct using a collection of image comparisons, colour comparisons, OCR strings and detect video motion areas.



Once all this has been defined, controlling the DUT navigation between screens is just a matter of telling the navigator which screen you want to go to using StormTest API calls in your python code. If, for any reason your menus change, then you can simply update your navigator file and the changes will be reflected in all tests that use that navigator.

A Python based script has a set of API functions to open and use the navigator. It can use multiple navigators in one script and has access to the full power of the StormTest Navigator.

2.1.1 Navigator Screen Types

The navigation map is a collection of screens joined together by links. The Navigator has 3 types of screen, each treated slightly differently. They are set from the properties grid - the property called screen type. They screen types are as follows:

- **Start Screen** You must define exactly one start screen in a Navigation Map. A start screen is a screen which can be reached from any other screen in the navigation map.
- **Screen In Map** The majority of screens in the navigation map are of type ScreenInMap.
- **Ignored Screen** We can specify a screen to be ignored when validating the navigation map.

For detailed information on screen types, please refer to the Developer Suite User’s Manual.

2.1.2 Traversing the links

Links connect screens together. Each link will consist of one or more button presses. The link graphic tells you information about the link since the link can be in different states. Examples of link states are *validated and passed*, *not validated*, *currently being validated* and *failed validation*. All screens should have the status validated and passed before using the navigator.

When asked to navigate between two screens, the Navigator uses Dijkstra's shortest path algorithm, a well-known algorithm in computer science to find the shortest path between two points. The Navigator calculates a time for each link and considers the shortest path to be that with the lowest time. In calculating the time to travel from screen A to screen B, it calculates the delays of all the key presses and then adds in an estimate of time for each test of the destination screen.

2.1.3 Building a Navigator

For detailed information on how to create a navigator, please refer to the Developer Suite User's Manual. For API information see the StormTest Development Center API Document.

2.2 StormTest Resources

Resources are stored in the central StormTest database and are available for everyone to use. They can be used in a Navigator, a Test Creator test or a Python test script. They are stored and can be viewed in a hierarchical set of folders.

2.2.1 Resource Types

Resources allow us to make changes across a large number of screens and navigators from the Resource Editor applet. StormTest stores the following resource types:

- Region - this is simply a rectangle and defined by 4 numbers
- Image - an image along with the intended size of the image
- Color - this is the RGB colour, tolerance to use when matching, flatness and peak error. All the values that are needed as input to a colour comparison
- String - a string of characters
- Screen – this is a Screen Definition Object (SDO). It is an object that can contain a number of verification tests. Please see chapter 15 for more details.

Resources can be created using the ribbon or from a right mouse click context menu in the resource browser.

2.2.2 Resource Management

The Resource Editor is the central point for editing named resources. From here you can create, view, update and delete any named resource. You can also reserve a DUT using the main ribbon and capture images from it to insert into a named image.

Just like when a Navigation map changes, the use of resources allows you to change some information in a central location and the change will have an impact on any test that is referencing that resource.

Examples of where this might be useful include, when the location of an image or some text on screen is modified in the menu (Region), when a channel logo is changed (Image), when a channel name is changed (String) or when a colour in a menu system is altered (Color). Any of these resource types can be changed centrally to eliminate the need to change your code.

The correct syntax to use with resources will be described later in this document. It is worth noting that the StormTest Python API uses the concept of relative paths and absolute paths. Absolute paths start with a '/' character but relative paths do not. Navigator and Test Creator always use absolute paths when referencing resources.

2.2.3 Creating Resources

For detailed information on how to create resources, please refer to the Developer Suite User's Manual. For API information see the StormTest Development Center API Document.

Resources will be used in some of the code examples later in this document.

3 First StormTest script

This chapter will demonstrate how to use StormTest basic functionality.

We will use a simple "Hello World!" script and walk you through the basic functionality, step by step:

```
# StormTest Client API
import stormtest.ClientAPI as StormTest

# Test environment definitions
import TestEnvironment

# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName"]

# Test Code

def test():
     # Connect to the StormTest server
     StormTest.ConnectToServer(TestEnvironment.ServerName, "Hello World 
Test")

     # Reserves a slot on the server and start serial logging
     StormTest.ReserveSlot( 5, TestEnvironment.RC_KeySet, logParam, False )

     # Print "Hello World" on console
     print "Hello World"

     # Wait 3 seconds
     StormTest.WaitSec(3)

     # End of test - release slots back to the server
     StormTest.ReleaseServerConnection()

     # Return test results
     return (StormTest.TM.PASS)

if __name__ == '__main__':
     StormTest.ReturnTestResult(test())
```

<u>Step 1: Import additional modules</u>

Depending on the complexity of the test, a number of different modules may need to be included. At a minimum, the StormTest Client API module has to be imported in order to use StormTest functionality:

```
import stormtest.ClientAPI as StormTest
```

The ClientAPI module is in the 'stormtest' package, which is installed by the StormTest Application installer. So the above line means import the ClientAPI module from the stormtest package and for the rest of this test script, refer to it as 'StormTest'. If you do not use the **as** keyword then you need to prefix every call to a function in the api with **stormtest.ClientAPI…** e.g:

```
import stormtest.ClientAPI
stormtest.ClientAPI.ConnectToServer(server)
```

Using the **as** keyword saves us some typing!

To write re-useable code it is always advisable to combine global functionality or definitions in one script and import this functionality to every other script where needed.

In our case we have combined all environment settings in an external file “TestEnvironment.py” that we can now import into all our test scripts:

```
import TestEnvironment
```

We advise you to use this mechanism in order to make your own life as easy as possible.

Let’s make a short detour and see what is defined in this file. In this very simple script we are only defining the name of the StormTest server, a remote control definition and global return values. This will allow us to change these globally used parameters in one location for all our other scripts without touching all files.

```
ServerName="StormTest_server "
RC_KeySet="default"
```

In the first two lines a name substitute for a string is defined.

In our case we have defined ServerName to represent our StormTest server as defined in our network. As an alternative, the IP address string can be used, e.g. “10.0.1.1”.

The RC_KeySet is a reference to the remote control IR database file. It is not really needed for this first simple test script as we are not sending any IR key presses to the box. We will cover this functionality in chapter 5.

Step 2: Global variable definitions

Of course it is also possible to define every variable in every script. As an example we want to mention the serial connection parameter here:

```
logParam = [57600, 8, "none", 1, "none", "TestName"]
```

These parameters will be used later on to initialise the serial connection from the STB to the StormTest server. The detailed information about each parameter and valid values can be found in the **Error! Reference source not found.** document.

For our purpose it should be enough to know that we have the following serial port parameters: baud rate, data bits, parity, stop bit, flow control and prefix for the log file.

Step 3: The test function

```
def test():
```

It is good practice to put your test steps in a function although it is not strictly necessary for simple test scripts.

Step 4: Connect to the StormTest server

Before you can run a test script you have to establish a connection to the StormTest server. This is done with the following command:

```
StormTest.ConnectToServer(TestEnvironment.ServerName, "Hello World Test")
```

As you can see, the previously defined server is passed as a parameter to the function call. StormTest is trying to establish a connection to our StormTest server. If we fail at this step the script is stopped automatically.

The second parameter is an optional description to allow you to identify the test. If you consistently pass the test name/description in this parameter then other users can query the server and see who is running what tests in the slots in the server.

Step 5: Reserve a slot

In this step we will explain how to reserve a slot on the StormTest server.

If you do not successfully reserve any slots then all subsequent calls to the StormTest client API will be meaningless.

The Reserve API call will also setup the remote control IR database to use and configure the serial port logging.

This is done with the following command:

```
StormTest.ReserveSlot( 5, TestEnvironment.RC_KeySet, logParam, False)
```

In detail, what we are doing is

- Reserving slot 5 for our test script.
- Telling the server to use our previously defined remote control configuration file.
- Configuring the serial port with the parameters as defined in Step 2.

- Disabling the video analysis capability. The last parameter indicates to the server whether the client needs video streamed to it for video analysis. It is not needed for the moment, so we set it to False.

This function allows us to reserve a slot exclusively so that nobody else can send commands to our slot and interfere with our tests. This slot is now reserved for us until we give it back to the system. Step 7 below illustrates how this is done.

Step 6: Execute test

This is the place where we execute all the test functionality.

In our case we are just printing "Hello World" on the client console and instructing the StormTest server to wait for 3 seconds:

```
print "Hello World"
St.WaitSec(3)
```

Step 7: Clean up

When we are finished with our test we have to make sure that all the resources get freed and are available again for the next person to execute any tests.

This is done with the following line:

```
StormTest.ReleaseServerConnection()
```

This function releases all previously reserved resources including the slot, serial interface, video capability (if reserved) and finally disconnects us from the StormTest server.

Step 8: Return result

When the test is executed it will jump to the line:

```
if __name__ == '__main__':
     StormTest.ReturnTestResult(test())
```

This will call the **test()** function and the result returned from that function is passed as a parameter to the **ReturnTestResult()** function which will then exit from the script.

Note that the reason we include the line:

```
if __name__ == '__main__':
```

is that this allows the test script file to be imported into another python file without being executed. If that line were not there then the **StormTest.ReturnTestResult(test())** line would be executed as soon as the file was imported.

4 Time to Watch TV

After working our way through the first test script, it is now time to reward ourselves by watching some TV.

How this is done is shown in our second Python script. See section 19.3 “videoWindow.py”.

The basics of how to use StormTest is described in chapter 2. This chapter describes how to write your first Python function.

Step 1: Python function

In this example we will put a part of our test in a function. A function is a block of code with a name that it is reusable. Therefore it can be executed from as many different points in your python script as required by referring to the name you have given it.

```
def runTest(args1 = 10):
```

This is a standard Python function definition which expects one input parameter.

The indented code following the “`def  FUNCTION_NAME():`” will not be executed until the function it is explicitly called.

Therefore we do not care about any functionality executed within this function for now. This will be described in-depth later on.

We include this information as the first step to show that a function exists in Python and that the code within it is not executed until explicitly called.

Step 2: Activate video capability

This time we want to use the video analysis capability even though it is only being used to display the video. Therefore we have to tell StormTest to reserve it for us:

```
StormTest.ReserveSlot( 7, TestEnvironment.RC_KeySet, logParam, True)
```

You see that the **ReserveSlot()** function call is almost identical compared to the same function call in section 3 “First StormTest script”.

But this time we want to use the video capability so we pass the flag “True” as last parameter. This is an optional parameter which defaults to *True* so in most cases the parameter can be left out. Once the **ReserveSlot()** function returns successfully the server starts to stream video to the client. However, we can’t see the video until we open a video window.

Step 3: Start video window

Now that everything is set up, we want to actually see the video. A simple function call will do that for us:

```
StormTest.ShowVideo()
```

The video window will pop-up on your client PC.

Step 4: The runTest function

In this example all test functionality that should be executed while the video window is seen should be included in the runTest function. The function takes one argument that is used by the WaitSec function.

Notice that if no argument is passed to the function, the function definition states that the args1 argument will be set to 10 by default.

Step 4: Stop the Video pop-up

```
StormTest.CloseVideo()
```

This function call closes the video window.

Note:
The code described in this section is only necessary if you wish to view the video from the STB while the test is running. In the case where tests are being run automatically there is probably no need to display the video window. The video window can be particularly helpful while developing and debugging your test cases.

In the case of running tests automatically the video function can be omitted and the test can be coded as one simple function along the lines of:

- Connect to server
- Reserve slots
- Execute test steps
- Disconnect from server

5 Tune to Your Favourite Channel

After seeing the video, wouldn't it be nice to be able to tune to your favourite channel now?

This chapter will make you familiar with some easy infrared (IR) commands.

The setup has not changed so we will start immediately in the runTest function where we discuss the IR commands.

You can find the Python script for this chapter in the appendix "19.4 irCommands.py".

Before we can discuss the IR functionality we want to bring your attention to the IR database. You have already seen in chapter 3 that we have a global definition for a remote control database:

```
RC_KeySet="default"
```

The name defined in the TestEnvironment.py has to either have the value *default* or it must correspond to a file on the StormTest server that holds the definitions of all the IR commands.

If *default* is used, then StormTest will automatically select the correct IR file for the STB that is in that slot (this link is setup up by your StormTest system administrator). This is the recommended value to use in your scripts. However, the default can be overridden by passing a file name here (such as "generic_remote.txt"). You may wish to use this if an STB supports more than one remote, and you need to test with these alternate IR codes.

<u>Step 1: Infrared commands</u>

Let's see the first command:

```
StormTest.PressButton("Power")
```

With PressButton() one IR command from the previously defined remote control database will be sent to the STB, on the reserved slot.

As you may remember, we defined the remote control database when we called ReserveSlot() (see chapter 3, Step 5).

The button string (e.g. "Power") should match the string defined in the remote control database.

One tip to make your script less error-prone is to use definitions for your IR commands as shown in the TestEnvironment.py example. If you misspell an IR key name, an error will be raised during test execution. Using an incorrect string directly has the disadvantage that the string will be sent to the StormTest server and will not be handled because it does not exist in your database.

Note that in this example, the PressButton() function is used without any slot parameter. This will send this command to all reserved slots.

Step 2: More infrared commands

We recognise that tuning to a channel, or pure digit entry, is sometimes inconvenient so we created a function that allows you to enter digits:

```
StormTest.PressDigits("123")
StormTest.PressDigits(123)
```

This function allows you to send as many digits to your STB as you want. As you can see above it is not important if you enter a string of digits or pure digits.

Step 3: More power

Sometimes different STBs require different IR power levels. The next function allows you to do just that and also to adjust the power level to your needs:

```
StormTest.IRSetSignalPower(3)
```

The IR device supports three different power levels. To achieve the most favourable results you will have to play with the parameter and see which ones suits you best.

Again, we allow you to adjust the level on one or all reserved slots at the same time.

We recommend not setting the power level to a high level as a default or to a level higher than needed because your IR device might have an effect on neighbouring STBs as well.

Alternatively a Navigator can be used to change to a channel, you will find the python script for this in section 19.4 “irCommands.py”.

```
StormTest.OpenNavigator()
```

This command, when used with no parameters opens the default navigator associated with the reserved slot. A default navigator must be associated with the DUT in the StormTest Admin Console.

```
StormTest.NavigateTo(“MAIN_MENU”)
```

Now navigate to the named MAIN_MENU screen, from the default start screen. All IR key presses and screen validation steps are performed along the way “within” this command. The screen named MAIN_MENU **must** exist in the default navigator.

Now we will navigate to the TV_GUIDE destination screen from the MAIN_MENU screen by using the above NavigateTo command with an extra parameter, the source screen.

```
StormTest.NavigateTo(“TV_GUIDE,”MAIN_MENU”)
```

6 Getting a Bigger Picture

In this chapter we show you how to change the video parameters. This is a useful feature e.g. if you intend to stream many videos to only one client PC.

You can find the Python script for this chapter in the appendix “19.5 videoParameter.py”

Step 1: Change the parameter

There are three different functions to change the video parameter:

```
StormTest.SetMaxBitRate(512000)
StormTest.SetResolution("2CIF")
StormTest.SetFrameRate(2)
```

These functions allow you to change the bit rate, resolution and frame rate of your video.

These functions will change all reserved slots or only one specific slot if specified via a second optional parameter.

Please check the **Error! Reference source not found.** document, to see which parameters can be p assed to each function.

Step 2: Get the actual parameter

Of course there exists for every set function a corresponding get function. These are:

```
StormTest.GetBitRate()
StormTest.GetResolution()
StormTest.GetFrameRate()
```

All of these functions will return an array of lists with one entry for each of your reserved slots.

This feature can be used to check if your modifications had an effect or to check if a modification is necessary at all.

NOTE

If the video settings are changed in a test, the new settings will remain for all subsequent tests connecting to that slot.

In the case of HD, the resolution is set automatically through the HDMI negotiation when the user first connects to the DUT, so the API SetResolution has no effect.

7 My Own PVR

In this chapter you will see how to use StormTest to capture a video for later analysis.

You can find the Python script for this chapter in the appendix “19.6 videoCapturing.py”

In the sample script you can see that it is not necessary to start the video window to record a video.

Step 1: Reminder – Activate video capability

To use the video capability please ensure that you enable it when reserving the slot.

```
ReserveSlot( 6, TestEnvironment.RC_KeySet, logParam, True)
```

Step 2: Start video capturing

One simple function call will allow you to start the video capturing:

```
StartVideoLog("VideoPrefix")
```

The video prefix parameter is optional and should help you to identify the recorded video stream for later use.

After starting the video logging function, all the video streamed to the client from the reserved STBs will be saved to a file in the ‘output’ directory. There will be a separate file for each reserved slot. In addition to the video file, StormTest will also store a copy of the test script and several other files. Those files tie the video file, the serial log and the test script together and contain time synchronisation information.

This allows the user to play back the test run in the StormTest Log Viewer and to step through all the log files in a synchronised way. See the **Error! Reference source not found.** for more information on t he StormTest Log Viewer.

Step 3: Stop capturing the video when you are finished

After you are finished with your test script but before you release your reserved slot you have to stop the video capturing:

```
StopVideoLog()
```

After calling the function above the recorded video file will be closed and nothing else will be recorded.

This function does not stop the StormTest server from streaming video to the client, e.g. an open video window will still show the actual video of the STB.

8 Turn Off the TV

From time to time a reset of one of your STBs may be required. With StormTest there is no longer a need to *manually* power cycle each STB requiring a reset.

In this chapter you will see how easy it is to power cycle a STB with StormTest.

You can find the Python script for this chapter in the appendix “19.7 powerFunctionality.py”.

Step 1: Power On

There is one simple function you can use to turn on your STB:

```
PowerOnSTB(100)
```

As you can see in the sample above this function takes two parameters (both optional). This function combines the power-on command with an additional time out to give the STB enough time to boot before the Python script continues execution.

Step 2: Power Off

The power-off function is as simple to use as the power-on function:

```
PowerOffSTB(100)
```

This function allows you to use optional parameters to specify a timeout which permits the STB to power off before allowing the script to continue.

9 And Yet It Moves

Most of the features presented so far have been related to changing and manipulating the STB. Now we want to show you some ways to analyse the effects of the features described in the previous chapters.

This chapter will introduce you to the feature that will allow your script to detect motion. Furthermore we will show you how to deal with the return value.

You can find the Python script for this chapter in the appendix “19.88 motionDetection.py”.

Step 1: input the slot number as a parameter

Now is time to improve our setup. Most of the time you will want to choose the slot where to run the script. In this first step we will show you how to input a slot number.

Firstly, we have to tell the Python script which slots we want to reserve. The easiest way to do this is by passing the number of the slots as an input parameter to the Python script.

These slots can then be accessed within the Python script via the sys.argv parameter. To use the functionality you first have to import it:

```
import sys
```

Now we can use sys.argv to get the input parameter.

When using this input parameter, you must first check if it is present:

```
if len(sys.argv) == 1:
    ReturnTestResult(TestEnvironment.FAIL)
```

If no additional parameters are passed to the script, the lines above will cause the script to terminate.

After that we verify that all input parameters are integer values because we don’t want any invalid input parameters to cause problems. The next line should be to verify the correctness of these parameters:

```
try:
       # Check that the argument is integer value
        slotNb = int(sys.argv[1])
except:
        # Invalid parameter found -> test fails
        print 'Invalid parameter passed - only integers are allowed'
        StormTest.ReturnTestResult(StormTest.TM.FAIL)
```

The short test above checks whether or not all input parameters are integer values.

Step 2: Detect motion

Sometimes it is necessary to check if your STB has crashed or whether a tuning request was successful. To do this, it is important to be able to detect motion.

Different ways to do this are shown as follows:

```
rect = [10,10,300,100]

returnValue = DetectMotionEx(None, 3)
returnValue = DetectMotionEx(rect, 3)
returnValue = DetectMotionEx(rect, 3, 7)

#Using resources
rect = '/MotionRegionResource'
returnValue = DetectMotionEx(FindNamedRegion(rect), 3)
```

In the first version we check the whole screen and wait up to 3 seconds to detect if there is some kind of motion present.

Sometimes you don't want to check the whole screen for motion, e.g. if you have to detect motion in a menu with a quarter screen video. The second version of the function call illustrates how the first parameter can be used to pass a rectangular area of interest. Please see the **Error! Reference source n ot found.** document for the format of the rectangle.

To get the coordinates a very easy and efficient way to do so is using the StormTest Test Creator Image Trainer. Please refer to the StormTest Developer Suite User Manual for further information about how to use this tool.

The third version shows an additional third parameter that can be passed to the function. This parameter will specify, in percent, how much change should be present between frames. The default value in function call 1 and 2 is 5%. In the third example, subsequent frames need to be different by greater than 7% before StormTest will declare that motion is detected.

The fourth example uses a resource which has been created previously, called 'MotionRegionResource'. This will have been created using the Resource Editor, and contain the rectangle of interest.

All four versions will check the reserved slots for motion. The next step will show you how that is done.

Step 3: Analyse the result

This step demonstrates a simple way to analyse returned results.

The return value from the DetectMotionEx() function is a list of lists. We have to check every parameter of interest in the list separately. The easiest way to do that is by using a for-loop over the result.

```
for ret in returnValue:

     slot= ret[0]
     status = ret[1]

     if status == False:
          print "No motion detected on slot " + str(slot)
          returnValue = 0
```

We are only interested in the status flag and slot number. If it is False, our DetectMotionEx() function has timed out without any motion detected. Note that the DetectMotionEx() function also returns other information listed in the StormTest API documentation .

10 Spot the Differences

Quite often you will have questions such as: did my button press have the desired effect or am I really in the right menu?

In this chapter we will show you an easy way to compare your actual screen with a pre-captured reference screen.

You can find the Python script for this chapter in the appendix “19.10 compareScreen.py”.

Step 1: Reference Image

Before you can do any image comparison you will have to capture a reference image. The best and easiest way to do that is with the help of the StormTest Remote control application. How to capture a reference image is outside the scope of this document but it is explained in detail in the **Error! Reference source not found.**.

Before you can proceed with this chapter you will have to record all reference images and retrieve any coordinates that might be needed.

When you record the reference image (and coordinates) you need to remember the key presses to reach this screen. This is because you will have to add them into the Python script at the provided place holder.

A second method is to use the setupCompareScreen.py (can be found in appendix 19.9.)

This Python script has been prepared to go to a specific screen, take a snapshot and store it for use later in the compare script.

All you have to do is add the desired key presses in the runTest() function. This script opens a video window to allow you to check if the key presses lead you to the correct screen.

Step 2: Setup

If you look at the script you will see that the setup is the same as in all previous scripts. It use the runTest() function where the compare functionality will be executed. There you will also find a place holder to allow you to add all the key presses needed to reach the screen to be compared.

Step 3: Compare the screen to your reference image

There are many different ways to perform image comparison. Choose the one that best suits your particular circumstances.

Before you use these functions make sure that the name of your reference image matches the name used in the script as well as the path. In our case the reference image and the Python script have to be in the same directory.

```
returnValueArray = WaitImageMatch("cap_refImage.bmp")
```

The function above shows the easiest way to use the compare functionality. It just needs a reference image for the function.

```
returnValueArray = WaitImageMatch("cap_refImage.bmp", timeToWait = 5,
waitGap = 1)
```

The function above needs a reference image, a timeout, and a waitGap for the function. The timeout is the maximum time to try to compare the reference image to the actual screen. The time gap defines the time between two comparisons.

```
returnValueArray = WaitImageMatch("cap_refImage.bmp",percent = 85,
timeToWait = 5, waitGap = 1)
```

If the default percentage is not good enough for your purposes, the second version of the function call shows you how to adjust it. In our case, we have set it to 85%, the percentage that the reference image has to match.

Sometimes it is not possible to compare the whole screen to the reference image because parts of the screen may include video (e.g. quarter screen), or other changing and unpredictable content (e.g. EPG content).

If you are not interested in the whole screen there are two ways to specify a region of interest.

```
rectangle_compScreen = [10, 10, 150, 100]

returnValueArray = WaitImageMatch("cap_refImage.bmp", 85, includedAreas =
rectangle_compScreen, timeToWait = 5, waitGap = 1)

#Using resources
refRegionResource = '/CompareScreenRectResource'
refImageResource = '/RefImageResource'

returnValueArray = WaitImageMatch(FindNamedImage(refImageResource), 85,
includedAreas = FindNamedRegion(refRegionResource), timeToWait = 5, waitGap
= 1)
```

The first way is to specify a rectangular area somewhere within the boundaries of the screen that contains the region of interest. In our case this is defined by rectangle_compScreen = [10, 10, 150, 100].

Please note that the four values that define the rectangle represent the top left hand corner of the rectangle (the first two values) and the width and height of the rectangle (the second pair of values)

Using this method it becomes possible to check for a list of icons in a stepwise fashion, by adding an offset to a known location to indicate the location on the image to compare. An example of this can be seen in the appendix, ”19.13 compareScreenOffset.py ”.

The second method in the above example shows how Image and Region resources can be used with the WaitImageMatch API. There is an assumption that these resources pre-exist in the database before the script is run.

Step 4: Analyse the result

After we have chosen the function that suits our needs best, it is now time to analyse the result.

```
for ret in returnValueArray:

        statusFlag = ret[1]
        slot = ret[0]
        imageObject = ret[2]

        if statusFlag == False:
            # Image does not match reference
            print "Image does not match reference on slot", slot

            saveStatus = imageObject.Save("cap_resImage.png")
            if saveStatus:
                print "A screen capture has been saved"
```

The way to check the results does not differ from the way we have seen in chapter “9 And Yet It Moves”. Again we get a list of lists of result information.

As you can see above, the statusFlag is False if the live video does not match the reference image within the provided parameters.

You can also see how a StormTestImageObject, returned by the comparison function can be saved in an image file.

11 The Black Sheep

This chapter will introduce you to the colour related feature of StormTest. And this time instead of waiting for something to appear we will wait for a colour to disappear.

```
You can find the Python script for this chapter in the appendix "0
###############################################################################
# Import modules
##################################################################################

# StormTest Client API
import stormtest.ClientAPI as StormTest
# Global return value definitions
import TestEnvironment

# System library
import sys

###############################################################################
# Variables
##################################################################################

# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName"]

# Image Capture Screen (captured used setupCompareScreen.py)
image_CompScreen = "cap_refImage.bmp"
# Rectangle, area to check
rectangle_compScreen =  (10, 10, 150, 100)


refRegionResource = '/CompareScreenRectResource'
refImageResource = '/RefImageResource'

# Region to check for when comparing images
StormTest.WriteNamedRegion(refRegionResource,(10, 10, 150, 100))

#Image to check for
image_file_obj = open("cap_refImage.bmp","rb").read()
StormTest.WriteNamedImage(refImageResource,(image_file_obj,704,576))

# Return value flag
returnValue = 1

###############################################################################
# Local functions
##################################################################################
def runTest():
    global returnValue

    ############################################################
    # Add here all key sequences to reach your reference screen
    StormTest.PressButton(TestEnvironment.RCSELECT)

    ############################################################

    #check reference screen over live video

    returnValueArray = StormTest.WaitImageMatch ( image_CompScreen)
    # Different variations of the same function
    #returnValueArray = StormTest.WaitImageMatch ( image_CompScreen, 85 timeToWait = 5,
waitGap = 1)
    #returnValueArray = StormTest.WaitImageMatch ( image_CompScreen, percent = 85,
includedAreas = rectangle_compScreen, timeToWait = 5, waitGap = 1)


    # returnValueArray = StormTest.WaitImageMatch (
StormTest.FindNamedImage(refImageResource), percent = 85, includedAreas =
StormTest.FindNamedRegion(refRegionResource), timeToWait = 5, waitGap = 1)
```

```
    #check the returned value
    #[slotNb, statusFlag,StormTestImageObject,percentage,nbCapturesPerformed, totalTime]
    for ret in returnValueArray:

        statusFlag = ret[1]
        slot = ret[0]

        if statusFlag == False:
            # Image does not match reference
            print "Image does not match reference on slot", slot

            saveStatus = ret[2].Save("cap_resImage.png")
            if saveStatus:
                print "A screen capture has been saved"

            returnValue = 0



##############################################################################
# Test Code
#################################################################################
# So that this module can be used as reusable module, or as standalone program
if __name__ == '__main__':

    if len(sys.argv) == 1:
        print 'Af slot number must be passed to this test as an argument'
        # No port specified -> test fails
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Verify that the argument is valid
    try:
       # Check that the argument is integer value
        slotNb = int(sys.argv[1])
    except:
        # Invalid parameter found -> test fails
        print 'Invalid parameter passed - only integers are allowed'
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Connect to the StormTest server, e.g. "justin", or "123.123.123.123",
    # and send a comment to help others to identify the purpose of the test
    StormTest.ConnectToServer(TestEnvironment.ServerName, "compareScreen")

    # Reserves a slot on the server and start serial logging
    slotAvailable = StormTest.ReserveSlot( slotNb, TestEnvironment.RC_KeySet, logParam, True )

    # Check if the slot is available
    if slotAvailable == False:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    #Show video
    StormTest.ShowVideo()

    # compare function
    runTest()

    #Close video window
    StormTest.CloseVideo()

    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()
    # Return from module
    if returnValue == 1:
        StormTest.ReturnTestResult(StormTest.TM.PASS)
    else:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)
```

colorDetection.py". The setup for this script is the same as in previous examples.

<u>Step 1: Pick up the colour</u>

Colour comparison is another tool that can be used to perform automatic testing. In this example we will wait for the “no video” screen to disappear. Note that by default, StormTest will display a “no video” message on the screen when the device in that slot is not outputting any video (such as when it is powered down).

We then need to pick up the right colour references, and coordinates. To do so you can use the StormTest Test Creator compare colour trainer. How to use this tool is outside the scope of this document. Please refer to the StormTest Test Developer Suite Manual for detailed information on the compare colour trainer.

This procedure enables us to declare the following variables:

```
rect = [200, 301, 60, 50]
color = [0,0,0]
flatness = 98
tolerances = (0,0,0)
```

Note that a detailed description of those parameters is available in the StormTest API documentation. In further testing it is likely that the tolerance and the colour you will pick up will be slightly different! Using the StormTest Resource Editor, a colour resource can be created to simplify the API, as shown in the second part of Step 2 below.

<u>Step2: power it on</u>

```
StormTest.PressButton(“Power”)

returnValueArray = StormTest.WaitColorNoMatch(color,tolerances,flatness,
includedArea = rect, timeToWait = 5,waitGap = 1)

#Using resources for regions and colours
refRegionResource = ‘/RegionResource’
refColorResource = ‘/ColorResource’

returnValueArray = StormTest.WaitColorNoMatch(
StormTest.FindNamedColor(refColorResource),includedArea =
StormTest.FindNamedRegion(refRegionResource), timeToWait = 5,waitGap = 1)
```

We consider that when the script starts, the box is in stand by and displays the “No Video” message. (Which indicate that no signal is output by the box)

The idea then is to power the box on and wait for the no video screen to disappear using the colour parameters provided.

<u>Step3: analyse the result</u>

After we have chosen the function that suits our needs best, it is now time to analyse the result

```
for ret in returnValueArray:

        statusFlag = ret[1]
        time = ret[7]

        if not statusFlag:
            print "colour did not disappear"
            returnValue = 0
        else:
            print str("colour vanished in %2.5f seconds" % time)
```

The way to check the results does not differ from the way we have seen in chapter “9 And Yet It Moves”. Again we get a list of lists of result information.

There we chose to print the time it took for the colour to vanish.

12 Are You Talking To Me?

Until now, we have seen a way to detect motion and to compare the screen to a reference image. Now it is time to show you a way to use the serial monitoring functionality.

You can find the Python script for this chapter in appendix “19.12 observeSerial.py”.

StormTest provides you with an active and a passive way to monitor your serial port. Before we discuss them in more detail you should note that the setup does not differ from the setup as seen in the previous chapters.

Step 1: Passive Serial Monitoring

The passive way of serial monitoring logs all occurrences of your chosen string in a separate log file.

```
StormTest.AddObserve("Error")
```

Take the function call above: this will cause StormTest to log all occurrences of “Error” into a separate log file (the name of this log file will be the same as the log of all the serial data with the addition of an ‘`_mon`’ suffix). This function will do nothing other than logging.

Step 2: Active Serial Monitoring

Sometimes it is not enough to just log the serial; sometimes you may have to react at runtime to special events.

Next we will show you how to use the serial monitoring feature in a more active way. To achieve this, you should know that active serial monitoring consist of two parts.

The function call:

```
StormTest.AddObserve("Warning", callBackWarning)
```

And the call-back, in case the desired string occurs:

```
def callBackWarning(slotNo, key, string):
     # Print the string that was found on the port
     print "Found", key, "on Slot",slotNo,". The complete line that was
     logged is:",string
```

The call-back must be a function that takes 3 input parameters.

You have to specify the call-back when calling the monitor serial function (as seen above). This will cause StormTest to use your defined call-back function instead of the default call-back (logging into a file).

Step 3: Stop monitoring

Of course StormTest also provides you with a function to disable monitoring. You can use this function as soon as you are done with monitoring the serial port, or when you enter a stage where you expect misleading serial prints.

```
StormTest.RemoveObserve("Warning")
```

As you can see, you only have to specify the kind of monitoring you want to stop by using the same string used to start the logging.

Step 4: Change serial parameter

Sometimes it is necessary to change the serial parameter “on the fly”:

```
StormTest.ChangeSerialParameters(logParam2)
StormTest.ChangeSerialParameters(logParam2, 2)
```

Above you can see two ways to change the serial port parameters within a Python script. The first parameter is mandatory and specifies the new serial port parameters. The format of the serial port parameters is the same as seen in chapter 3 “First StormTest script”. The second parameter is the optional slot number as seen before.

This function will only change the serial port parameters and keeps logging into the same debug file (i.e. the one opened when the `ReserveSlot()` function was called).

Step 5: Log to a new file:

Sometimes you may want to log the content into a new file, and perhaps change the serial port parameters:

```
StormTest.StopSerialLog()
StormTest.StopSerialLog(2)

StormTest.StartSerialLog(logParam2)
StormTest.StartSerialLog(logParam2, 2)
```

The stop function above will stop all actual active logging (or just one slot, depending on the usage of the second optional parameter).

StartSerialLog() allows you now to start the logging again in a new log file. Of course you will have to specify the name of the new file in the serial port parameter structure (logParam2, please see the **Error! Reference source not found.** document if you are not familiar with that structure).

A new file will always be started when StartSerialLog() is called, even if the same serial port parameters and file name prefix are passed in again.

If you just intend to log everything into a new log file, there is no need to call StopSerialLog() first. StartSerialLog() will close any active log file first before starting a new one.

Step 6: Check for Logs

After you are done with your tests you normally would like to see if some error occurred or not.

The next function shows you a way to retrieve the log file details within a Python script:

```
returnValueArray = GetObserveLog()
```

This will return an array with log information for all reserved slots:

```
if returnValueArray == []:
    print "No log information has been found!"
else:
    for slot, fileSize, logFile in returnValueArray:
        if fileSize == 0:
            print "No data was logged from the serial port on slot", slot
        else:
            print filesize, "bytes of data were received from the serial
port on slot", slot, "and logged to file", logFile
```

We are interested in the log file size. This will show us if any of the 'observed' strings have actually been found in the serial data streamed from the STB. A file size bigger than zero indicates that some serial information has been logged in the '_mon' log file. The logFile string contains the actual file name and path of this '_mon' log file.

Step 7: Talk to the STB

Sometimes it might be desirable to send a command to the STB via the serial port and wait for the result:

```
StormTest.SendCommandViaSerial("Tune_To_CH101", "Tuned", 3)
```

The command above will send a tune request to the STB and waits a maximum of 3 seconds or until the string "Tuned" is printed on the serial port.

Of course it only makes sense to use this function if your STB supports commands via serial port.

Step 8: Send the serial data to a custom parser

Perhaps the STB that you are testing outputs specially formatted text that could be useful to your test scripts. For example, the serial port output might indicate the channel that the STB is tuned to, which would give you an alternative way to check that your IR commands to the box have been successful.

In this case, you can write your own custom parser for the serial data. This parser can then be 'hooked' in to the serial stream and all serial data from the box will be passed to that parser.

The parser should be written as a function with two arguments, such as the following:

```
def customParserFunction(slot, data):
    print "Received:", data, "on slot number", slot
```

Obviously, your parser function will be more complex than this and may make use of global variables to store information received on the serial port for later checking in your test script.

Once your parser function is defined, it can be hooked into the serial data stream very easily by calling:

```
StormTest.SetExtParser(customParserFunction)
```

Step 9: Writing your own messages into the serial log file

Finally, you might want to write your own messages to the serial log file. Perhaps in order to aid the later checking of log files, it would be useful to write messages directly into the serial log files to indicate what test step is happening.

This can be achieved by a simple function call:

```
StormTest.CommentToLog("Event A is about to happen")

StormTest.CommentToLog("A failure has happened on slot 5", 5)
```

Similarly to all other functions in this section, it is always possible to add a slot number parameter to the end of the argument list, which has the effect of executing the function for that slot number only. In the first example above, the statement will appear in the serial logs of all the reserved slots. In the second example, the statement will only be written to the serial log of slot 5 (assuming of course that slot 5 has been reserved by the user)

13 Reading from the Screen

In most cases it is sufficient to use the image comparison functionality of StormTest in order to verify that the STB that is being tested has reached the menu or screen that you expect in your test. However, sometimes, you will actually need to know what text is on the screen. Perhaps the text in question differs on each STB or perhaps it differs depending on the time of day.

In a general manner when you know what you expect from the UI it is better to use image or colour comparison. When you don`t and it is text, OCR is appropriate.

Either way, you can use StormTest's OCR (Optical Character Recognition) feature to literally 'read' the text off the video displayed by the STB.

Using the OCR functionality is extremely simply and can be achieved in two ways. You can find a sample script for this chapter in appendix "19.14 OCRsample.py".

Method 1: Reading the text off the screen

The first method involves reading text that is being displayed in the live video stream.

```
ocrResult = StormTest.OCRSlot(None)
```

This example will attempt to read all the text it can find on the live video coming from all the slots that have been reserved by the test.

Since the video output from a STB can contain a lot of text (especially on a programme guide screen), the user should always restrict the OCR process to a smaller part of the screen. This will increase the accuracy of the results and will also give the test script more useful data.

This can be done by passing a rectangle into the function. The rectangle is defined in the same way as it is for the image comparison function seen in chapter 10 – i.e. (x, y, width, height) where x, y is the top left hand corner of the rectangle. This rectangle can be created as a resource and looked up in the database, using the StormTest Resource Editor.

```
ocrResult = StormTest.OCRSlot((100,100,200,50))

#Or using a region resource
refRegionResource = '/RegionResource'

ocrResult = StormTest.OCRSlot(StormTest.FindNamedregion(refRegionResource))
```

IMPORTANT: In order to improve the accuracy of the OCR process, the image that is passed to the OCR engine is **always** captured at the full 4CIF resolution (i.e. 704x576) or at the agreed HD resolution on the server side, even if the test has set the video resolution to a different value. This means that the rectangle that is passed to the OCRSlot() function should be using this as its frame of reference (even though you may be using a different frame of reference for your image comparison functions)

As with other StormTest API functions, the test script can restrict the OCR so that it is only carried out on one of the reserved slots by passing an optional slot number parameter:

```
ocrResult = StormTest.OCRSlot((100,100,200,50), 5)

#Or using a region resource
refRegionResource = '/RegionResource'

ocrResult =
StormTest.OCRSlot(StormTest.FindNamedregion(refRegionResource),5)
```

Method 2: Reading the text from a screen capture

The second method of using OCR is to pass it an image, rather than using the live video stream. This image can be any image or can be a resource, but will most likely be a screen capture that was generated by the test script.

A bounding rectangle can also be passed to this function, but since it is **not** operating on the video from the slots, there is **no** optional slot number parameter (but you can use the slot number when capturing the image that is used).

```
captureResult = StormTest.CaptureImageEx(None,"cap.jpg")

#Or using a region resource
refRegionResource = '/RegionResource'

for result in captureResult:

     slot, status,imageObject,filename

     if status == True:
          ocrResult = StormTest.OCRFile((50,10,100,50), filename)
          #Performing the same task using a resource
          ocrResult =
StormTest.OCRFile((StormTest.FindNamedregion(refRegionResource),filename)
```

As you might have realised by now, this allows you to carry out an OCR on a screen capture that is the same resolution as the video resolution set by the test script.

Checking the OCR results

The return value from the two OCR functions is quite complex and contains a lot of information. Its structure is shown below:

```
[
    Total suspicious symbols,
    Total unrecognised symbols,
    Total number of symbols found,
    Total number of words found,
    Array of Results:               [
                                        [Slot No x,    String detected]
                                        [Slot No y,    String detected]
                                        [Slot No z,    String detected]
                                    ]
]
```

When checking the result of an OCR operation, the first thing that you should do is check that the statistics in the structure have a value greater than zero (it is enough to check just one value). If they do not, then there has been an error during the OCR process. Note that in the case of an error, a string describing the error will often be found in the 'Array of Results' that is returned.

An example of how to access this results array is shown below:

```
#Using a resource
rect = '/RegionResource'

ocr_result = StormTest.OCRSlot((114,470,165,30))
#Performing OCR, overwriting the above results, using a resource
ocr_result = StormTest.OCRSlot((StormTest.FindNamedregion(rect))

if ocr_result[2] > 0:          # Check the total no. of symbols
    for slot, txt in ocr_result[4]:
        # Remove the spaces from the start and end of the text
        tmp = txt.strip()

        if tmp == "Expected Text":
            print "Expected text on screen on slot", slot
        else:
            print "Error - unexpected text on screen for slot", slot
else:
     print "Error when using OCR"
```

Please note that when the optional slot number parameter is passed to OCRSlot(), the fourth element in the OCR result array is not another array. Rather it is a simple string with the OCR result from the slot that was requested.

<u>Using pre processing filter</u>

In some case you may want to apply some filter to the image that is analysed by the OCR engine. This improves the OCR reliability if for example the background colour of the text is close to the test colour itself, or if that background is not a constant colour.

```
#Using a resource
rect = '/RegionResource'

StormTest.OCRSetImageFiltering([("Blur","Gaussian3x3"),("Color","Gray")])
ocrResult = OCRSlot((100,100,200,50), 5)
#Performing the same OCR as above but taking the region from a resource.
ocr_result = StormTest.OCRSlot((StormTest.FindNamedregion(rect),5)
```

All available filters are listed in the StormTest API documentation.

Recommendations

This is a set of recommendation that we strongly suggest you to use when using OCR:

- Make sure that if you perform OCR on a banner it does not disappear before the OCR is complete
- If the expected result is numbers then convert incorrect characters by using the StormTest API StringsAutoCorrection(string1,type = OCRStringCorrection.AllNumbers)
- In the same way if the expected result is letters , convert incorrect characters using the StormTest API StringsAutoCorrection(string1,type = OCRStringCorrection.AllCharacters)
- Coordinates that you are using should be closely cropped (1-2 pixels above and below the text)

14 Timing a Channel Change

Now you have covered the main areas of the StormTest API it's time to look at a more advanced part of the API. The high speed video timer allows a StormTest user to accurately measure the amount of time that a device under test takes to zap from one channel to another. StormTest uses a patented method to detect the change in video which indicates that a channel zap has occurred. This is dependent on a "black screen transition" occurring during a channel change – the picture goes black momentarily before the new channel appears. This is normal for devices with standard tuners, but for IPTV that may not be the case.

The zap measurement is a limited resource that must be reserved before use. This means you have to reserve the video timer before using it and release it once the measurement is complete.

You can find a sample script for this chapter in appendix "19.15 ZapSample.py".

Step 1: Reserve the Video Timer

Because the zap measurement is resource intensive it must be reserved before use.

```
if StormTest.ReserveVideoTimer(2):
```

This function reserves a video timer resource for the specified slot i.e. for slot 2 in this case. It should be noted there is a limit to the number of video timers that can be reserved at any one time (for an SD rack this is 4 simultaneous video timers per 16 slot rack; for an HD rack, it's not limited, but if more than one is used, memory availability may become an issue).

Step 2: Start the Zap Measurement

Once the video timer has been reserved, it is possible to start measuring the zap times between channels.

```
Zapstatus =
StormTest.StartZapMeasurement("Channel+",0,[0,0,704,405],1000,5,2)
```

In this example the arguments used are:

"Channel+" (irKey)	The IR key stroke which initiates the zap time measurement - the zap time is measured from the start of this keystroke.
0 (keyCount)	The number of irKey's to ignore before starting the measurement. For example, setting irKey = '1' and keyCount = '1' would allow the client to send the ir command '101' and have zap time start on the second '1' instead of the first.

[0,0,704,405] (rect)	The area of the video to examine when determining zap time. This can also be a region resource.
1000 (colorCount)	The threshold for determining ZAP. When the color count falls below this value, the zap is considered to have started and when it rises above this value, the zap is considered to have ended.
5 (timeout)	The total time in seconds to wait for the zap to complete
2 (slotNo)	The slot number on which to start the zap measurement

When passing the area of interest (rect) to StartZapMeasurement(), you should take into account if a banner is displayed somewhere on the screen. If there is, this area should be left out of the area of interest. This is because the banner can stop the number of colours from dropping below the specified "colorCount". See section 14.1.1 “Selecting correct rectangle and color threshold” below.

"colorCount" should also be set to an appropriate level. This can be tuned by observing the number of colours during a zap with GetRawZapMeasurement(). See section 14.1.1 “Selecting correct rectangle and color threshold” below.

<u>Step 3: Do channel + and wait 5 seconds</u>

Once the start zap measurement has been set up, you need to perform the IR action that it is waiting for, in this case it is a channel + key. Then wait for the zap to finish.

```
StormTest.PressButton(“Channel+”)
StormTest.WaitSec(5)
```

<u>Step 4: Retrieve Zap Times</u>

Once the zap has completed the zap times can be retrieved using GetZapTimes().

```
zaptimes = StormTest.GetZapTimes()
```

GetZapTimes() returns a list which contains the zap times that have been measured. Below is an example of the zap times returned.

```
zaptimes = [(2, True, 0.317, 0.67400000000000004)]
```

The first value in the returned list is the slot number that the results are from. The second value indicates whether the zap measurement was successful or not. The third value is the time (in seconds) from the trigger to start of zap, i.e. the time until the screen goes black to change channel. The fourth is the time (in seconds) from the trigger to the end of the zap, i.e. the full zap time. If the second value is "False" then the third and fourth values do not represent the zap time.

Step 5: Retrieve the Raw Data Measurements

Now that the measurements have been taken, it is possible to retrieve the raw zap measurements taken using GetRawZapMeasurements().

```
rawZapMeasurements = StormTest.GetRawZapMeasurements( )
```

There can be quite a lot of measurements retrieved. 25 measurements for PAL, (30 on NTSC systems) are performed every second. Below is an example of the values that are returned.

```
[(2, [(1, 53281.0), (2, 54279.0), (3, 55143.0), (4, 54849.0), (5, 55142.0),
(6, 53417.0), (7, 80.0), (8, 78.0), (9, 85.0), (10, 86.0), (11, 79.0), (12,
81.0), (13, 81.0), (14, 83.0), (15, 82.0), (16, 79.0), (17, 83.0), (18,
88.0), (19, 82.0), (20, 87.0), (21, 82.0), (22, 75.0), (23, 77.0), (24,
83.0), (25, 87.0), (26, 76.0), (27, 83.0), (28, 81.0), (29, 75.0), (30,
79.0), (31, 79.0), (32, 84.0), (33, 83.0), (34, 72.0), (35, 80.0), (36,
79.0), (37, 79.0), (38, 79.0), (39, 81.0), (40, 79.0), (41, 85.0), (42,
80.0), (43, 78.0), (44, 84.0), (45, 80.0), (46, 83.0), (47, 76.0), (48,
83.0), (49, 77.0), (50, 79.0), (51, 73.0), (52, 79.0), (53, 77.0), (54,
16623.0)])]
```

The return is a list of tuples. The first value is the slot number that the zap measurements are being run for. The second (and subsequent values) are a pair of values, the first value in this pair is the frame number since the irKey has been issued, the second value in this pair is the number of YUV colours in the rectangle of interest for that frame.

You can see in the example above: at frame 7 the colour count drops sharply. This is the point at which the screen goes blank. At frame 54 the colour count goes back up again. This is the point at which the new channel starts playing.

Step 6: Free the Video Timer

Once the zap measurement has been carried out it is necessary to free the previously reserved video timer.

```
StormTest.FreeVideoTimer()
```

14.1 Zap Timer Tips

14.1.1 Selecting correct rectangle and color threshold

In order to select the correct rectangle coordinates, and the correct colorCount threshold, it is recommended to do an initial zap test run with some additional API calls. You can just make a rough initial estimate for rectangle coordinates and colorCount for this test run. The additional API calls are:

- Before calling StartZapMeasurement(), make a call to EnableCaptureFrames(). This instructs the system to save the video frames during the zap
- After the zap has completed, call GetCountImagesSaved(), and using the count value returned, call GetRawZapFrame() in a loop to retrieve all the captured frames.
- Call GetRawZapMeasurements()

Now you have all the tools you need: a list of the colorCount measurement for each frame, and an actual image for each frame. Go through all the images using any image viewer (these frames will be *.png files in the log directory). Check first what the most appropriate rectangle would be: it should exclude any banner which appears (or remains) on screen. This becomes the “rect” parameter of StartZapMeasurement(). Then look at the colorCount values for each frame. You should see the colorCount dropping down for the first “blank” image, and going back up again when video resumes. You can now judge by the colorCount values what an appropriate threshold value should be. This can differ greatly depending on

- The size and location of the rectangle chosen
- Whether the video is SD or HD
- Interference effects: in some cases for a screen with no video, the appearance of a banner outside the chosen rectangle can still increase the colorCount inside the rectangle. In this case the colorCount threshold should be higher than the value for a frame that includes the banner.

14.1.2 Choosing timeout value

Be sure to choose a timeout value that exceeds the maximum zap time (as measured from the trigger to the resumption of video). Be aware too that only half of this time is allocated for the colorCount to initially go below the threshold: if that does not happen a timeout will occur.

14.1.3 Measuring other performance values

Sometimes it is required to measure the time taken for an action which has no blank screen transition, such as bringing up the EPG, or channel change on an IPTV device. In this case the Zap Timer cannot be used. Instead you can use the High Speed Timer. Proceed as follows:

- Reserve the video timer
- Call EnableCaptureZapFrames()
- Call TriggerHighSpeedTimer()
- Trigger the required action, and wait a few seconds for it to happen
- Call StopHighSpeedTimer()
- Retrieve the saved images using GetRawZapFrame()
- Now you can go through the saved images checking for a known image pattern, using e.g. CompareImageEx() or OCRFile().
- The number of frames required to reach the known pattern gives a time result e.g. for PAL video, 25 frames = 1 second.

See section 19.16 “HSTSampleEPG.py” below.

NOTE

In a case where the video freezes on a channel change, go through the saved captured images and compare each image against the next. At the point where they differ this indicates the point where video has resumed and the zap has completed.

15 Screen Definition Objects

A new concept was introduced in version 3.2 of StormTest: the Screen Definition Object (SDO). Using SDOs in Python code is for more advanced users; it requires an understanding of object oriented programming. However a brief description will be given here. See the StormTest API document for full details. A helper script has been provided here to make it easier to work with SDOs.

The idea of a Screen Definition Object is that multiple operations on a single screen can be combined into one single "object" in code. For example: a compare image, and OCR and a colour compare on one screen can all be combined into a single SDO.

An SDO can contain validation items and read items. All will be carried out on the same screen. A validation item is something like compare image – it will pass or fail. A read item is something like OCR: it returns a value but there is no pass/fail.

An SDO can also contain Detect Motion and Detect Audio items. In these cases the tests will not be carried out on a single screen, but instead on a sequence of frames.

Practical advantages of using an SDO are:

- All operations are carried out on the server (rather than on the client or client daemon). This saves time and saves network bandwidth.
- All operations are carried out on the same captured screen (except for detect motion or audio detect, which can also be done in an SDO). So you get more consistent results across different tests.

15.1 Using SDOs

SDOs can be used in Python code, and in the StormTest Navigator (see section 2.1 for information on the Navigator).

SDOs can be saved as resources (see section 2.2 for information on resources).

15.1.1 Creating and Using an SDO in the Navigator

To create an SDO in the Navigator: first create your validation tests on a screen as normal. Now select "Group" from the menu ribbon. You will see a Screen Definition icon appear under the screen. You have now combined your tests into an SDO. You can use the Navigator as normal, and any validation on this screen will use this SDO.

15.1.2 Saving an SDO as a resource

Right click on the SDO icon (mentioned above) in the Navigator. Now select "Convert to Resource" and then "Screen". You can now give this SDO a name and save it as a resource.

15.1.3 Using an SDO in Python code

Once an SDO has been created, there are two principle ways it can be used in Python code:

- Calling a Navigator function from Python (on a Navigator which includes SDOs)
- Calling an SDO Python API function directly.

If you call a Navigator function such as ValidateNavigator() or NavigateTo(), and if that Navigator includes SDOs, then the Navigator will use those SDOs to do the screen tests.

From Python code you can call an SDO directly. Functions such as WaitScreenDefMatch() are available for this (see the Python API for details).

Examples of these Python calls are given in the section 15.1.5 below.

15.1.4 Analysing SDO results

One of the more difficult aspects of using SDOs is checking the results of an SDO call. A call to a function which runs SDO tests returns an SDO (object) which contains many sub-objects, such as:

- An overall pass/fail for the SDO
- A pass/fail for each test
- Percentage matches
- OCR string read for an OCR test
- A captured image

These are all encapsulated in the single SDO object which is returned. To extract the separate items you must use object oriented programming techniques and it can be quite complex.

To make this easier we have provided a script which analyses SDO results. Please see the accompanying script in Appendix 19.20: “SDO_Parser.py”.

This full test script will not run on your system because it uses a Navigator and an SDO (saved as a resource) which you will not have. However it illustrates some important concepts regarding usage of SDOs. And the function called sdoParser() can be taken and re-used in any other Python script (just make sure you have imported stormtest.ClientAPI as “StormTest” in your script). Just pass an SDO as argument to the function.

15.1.5 SDO Code examples

Look at the code near the end of the full test script in Appendix 19.20, just after the “Test Code” header: after connecting to a server and reserving a slot, it opens a Navigator and calls NavigateTo().

```
res = StormTest.OpenNavigator("OrchidSDO")
screenDefName, res, sdoResult = StormTest.NavigateTo("HD")[0][1][2][0]
```

The return value from this NavigateTo() call includes an SDO (because the Navigator contained an SDO as validation criteria). As you can see this return value contains many nested lists, and we extract from this the sdoResult item, which is actually in the form of a Python dictionary. Next, we extract the actual SDO object from this dictionary and pass it to our sdoParser utility function:

```
sdoParser(sdoResult['Screen Definition'])
```

The parser function prints out values for all the tests contained in the SDO, something like in this extract:

```
CONSOLE   region.Name OCR Compare String 1
CONSOLE   region.Comment None
CONSOLE   region.ReturnScreenImage 128
CONSOLE   region.VerifyStatus True
CONSOLE   region.Rect [125, 309, 208, 29]
CONSOLE   region.Verify True
CONSOLE   region.PassOnNoMatch False
CONSOLE   region.expectedText [u'ALL CHANNELS']
CONSOLE   region.MaxDistance 0
CONSOLE   region.AutoCorrect None
CONSOLE   region.Languages [u'eng']
CONSOLE   region.ImageFilters None
CONSOLE   region.LegacyInvert False
CONSOLE   region.ResultText ALL CHANNELS
CONSOLE   region.RawText ALL CHANNELS
CONSOLE   region.OCRStatistics [0, 0, 11, 2]
COMMENT   OCR criteria
```

This extract just shows the output for an OCR test; other sections will show output for image compare, motion detect etc. Important items in this extract are:

- VerifyStatus = True, this means the OCR test passed
- ResultText = "ALL CHANNELS", this is the actual text read by OCR.

You can use this sdoParser utility in any script, to analyse the results from a returned SDO.

Next, as another example, we get a separate SDO that we had previously saved as a resource. We call WaitScreenDefMatch() on this SDO, which runs the SDO tests.

```
    sdo = StormTest.ReadNamedScreenDefinition("/Ronan's Resources/OrchidHD_SDO")[0][1]
    res = StormTest.WaitScreenDefMatch(sdo,5)
```

Again the returned value includes an SDO. We take this resulting SDO and pass it into our SDO parser:

```
    sdoParser(res[0][1])
```

The StormTest API document contains full details of the various sub-objects and methods that can be contained in a Screen Definition Object. It also contains full details of the returned values from calls to NavigateTo() and WaitScreenDefMatch().

Tip: if the return value from an API call is complicated, like that of NavigateTo() for example: just put it into a variable and print it out.

```
    returnVal = StormTest.NavigateTo("HD")
    print returnVal
```

Then run the script and you can examine the returned value to see its structure.

16 Video Quality Analysis

Another advanced feature of StormTest is Video Quality Analysis, or VQA. This feature allows users to analyse the quality of a video stream. It does not require a reference stream - any live or recorded video stream may be used.

Please note that an additional license is required to use the StormTest VQA feature.

This chapter will guide the user through writing a Python script to use the VQA feature. It will also give some brief information on how to interpret the results.

For more background on VQA, please refer to the document "StormTest Video Quality Analysis", which is at Engage at this link:

https://larisa.engage.s3group.com/docman/view.php?group_id=6&docid=53

16.1 Writing a Python script for VQA

The API functions for VQA are object-oriented. You can start by setting up the options for VQA. To do this, first create VideoOptions object, and then set the required attributes.

```
    vqaOptions = StormTest.VQACreateVideoOptions()
    vqaOptions.Format = "H264"
    vqaOptions.FrameRate = 0
    vqaOptions.Backlog = 1500
    vqaOptions.Jerkiness = True
    vqaOptions.Crop = None
    vqaOptions.DesignSize = None
```

The 'Crop' and 'DesignSize' options were added in release 3.3.3 of StormTest. The default 'Crop' value is set to 'None', meaning VQA is performed on the entire video frame. To set this value a list of 4 integers is used. When set, VQA will then only be performed on this defined area of the screen. The minimum final size for the crop area is 176x144.

If the 'DesignSize' is set to 'None', the default value, then the 'Crop' values are considers to be an absolute rectangle. If the 'DesignSize' is not 'None', then the 'Crop' values will be scaled as necessary by the VQA engine to analyse the same intended region.

The meanings of the various VQA options are described in the StormTest API, under the section VQAVideoOptions Class Reference.

Next, we create an actual VQA engine object, passing in the options we have just defined.

```
    vqaEngine = StormTest.VQACreateEngine(vqaOptions)
```

At this point we could in theory just start VQA analysis, and stop it at some later point in time. Instead, we'll define start and stop triggers. Our start trigger will be based on time: a delay of 5 seconds.

```
    startTrigger = StormTest.VQACreateTrigger()
    startTrigger.Delay = 5
```

Our stop trigger will be after 1500 frames have been analysed.

```
    stopTrigger = StormTest.VQACreateTrigger()
    stopTrigger.FrameCount = 1500
    stopTrigger.FromNow = False
```

Now we can call functions to both start and stop VQA, passing in the respective trigger objects. This works fine because both actions only happen when they are triggered.

Additionally we should pass in a callback function to the start VQA call. This enables us to receive events back from VQA, and this is how we can get results from the analysis.

```
    StormTest.VQAStartAnalysis(vqaEngine, trigger = startTrigger, callbackFunction = 
CallbackFunction, slotNo = slotNumber)

    StormTest.VQAStopAnalysis(vqaEngine, trigger = stopTrigger)
```

What does the callback function look like? It takes three parameters:

```
    def CallbackFunction(slotNo, engine, event):
```

The event item is most relevant here. This is an object of type VQAEventType. It has attributes Type, Value, TimeStamp and FrameNumber. If the event Type is VideoResult, then the Value attribute will contain a list of VQA results. Each item in this list will also have its own FrameNumber attribute.

Here is an example, from our sample script, of how we can check if the event Type is VideoResult, and if so we write some of the Value elements to a file:

```
    if event.Type == StormTest.VQAEventType.VideoResult:
        for i in event.Value:
            out.write(str(i.FrameNumber)+", "+str(i.MOS)+", "+str(i.FrameQuality)+", 
"+str(i.Jerkiness)+", "+str(i.Blockiness)+"\n")
```

There are many more elements of data for each frame than are mentioned here, see the API document "VQAVideoResults Class Reference" section for more details.

We now have a complete Python script to run StormTest VQA. Please see the sample script "VQASample.py" for a complete VQA script which you can run. The script creates a file with .csv extension and writes the results to this file.

16.2 What does it all mean?

If we run the sample script and open the results .csv file in Excel, we see data that looks something like this:

frame	MOS	FrameQuality	Jerkiness	Blockiness
0	None	100	0	0.179022
1	None	100	0	0.282737
2	None	32.49398	0	0.182216
3	None	100	0	0.285837
4	None	31.20553	0	0.178393
5	None	31.85452	0	0.287049
6	None	31.96704	0	0.179114
7	None	33.36023	0	0.282689
8	None	32.68434	0	0.17854

What do the various pieces of data mean? On the left is the frame number – this identifies the frame starting from the point at which VQA was triggered to start. After this, these are the data items:

16.2.1 MOS

MOS stands for Mean Opinion Score. It gives an overall assessment of the quality of the video based on the previous 5 seconds of video. This is why for the first set of frames there will be no MOS score. The scale is from 0 to 100.

16.2.2 Frame Quality

This score indicates the perceived quality of each individual frame. This is independent of other frames. The scale is from 0 to 100.

16.2.3 Jerkiness

This score indicates if the video is considered to be moving, or “frozen”. The scale is from 0 to infinity: 0 means moving video, any other value indicates the duration (in milliseconds) for which video was frozen.

16.2.4 Blockiness

This score indicates any artefacts, or macroblocking, for this frame. The scale is from 0 to 255: typically a value from 0 to 3 means good video, and anything higher indicates distorted video.

Please note that the above is only a subsection of the data which is returned from VQA – you can choose to save as much or as little as you like.

Please see the document “StormTest Video Quality Analysis” for more detailed explanation on the VQA data.

16.3 Some additional tips

16.3.1 Setting the video format

One of the VQA engine options is “Format”. This can be MPEG2 or H264. It’s important to know which format is used in your video – if this is not set correctly then VQA results will not be as accurate.

16.3.2 Setting the backlog size

Another of the VQA engine options is “Backlog”. This determines the size, in frames, of the input frame queue. In our sample script we did not set this, so the default value of 500 is used. You can set this to whatever size you want, but the maximum value depends on memory availability.

The backlog is important if the rate at which you receive frames is greater than the rate the VQA engine can analyse them. It means that for a certain time no frames will be dropped, but then when the backlog is full frames will start to be dropped.

16.3.3 Setting the frame rate

In our sample script we set the VQAFrameRate option to 0. This means the VQA engine tries to analyse all the frames at whatever rate they arrive. If it cannot keep up when its input frame buffer is full, it will start dropping frames.

An alternative is to set a fixed frame rate that you know the VQA engine can handle. In this situation frames will be dropped in a deterministic manner (e.g. every 2nd frame is dropped, or 1 out of every 5 frames is dropped). If frames must be dropped, then dropping them in a deterministic manner is likely to give more consistent VQA results.

The backlog, frame rate and test duration are all interdependent. You can play around with these variables to find the best results for you.

16.3.4 Waiting for a VQA event

It’s possible also to call a function which waits on specific events, while VQA is running. This was not included in the sample script – please see the API documentation for details.

17 Good Coding Practices

By now you know how to write some basic StormTest scripts. As you write more and more tests it becomes very important to capitalize on what you have done. You don`t want to write the same piece of python again and again.

You will also realise that it is very important to output comprehensive test logs. Indeed, depending on your organisation, the people writing the tests may be different from the people reviewing the test results. So, ideally, by reviewing your test log one should understand what the test is about and spot quickly where something went wrong (if it did!)

In this chapter we will present two important concepts: test steps and utilities.

17.1 You can find the sample scripts for this chapter in appendix 19.17 "VQASample.py

```
# ==========================================================================
# Copyright (c) 2010-2015 Silicon & Software Systems Ltd.(S3 Group).
# All rights reserved. This source code and any compilation or derivative
# thereof is the proprietary information of Silicon & Software Systems Ltd.
# and is confidential in nature. Use of this source code is subject to the
# terms of the applicable Silicon & Software Systems Ltd. license agreement.
#
#  MODULE: VQASample.py
#
#  DESCRIPTION: Simple python script showing how to use the Video
#               Quality Analysis (VQA) module.
#               Note that an extra license is required for VQA.
#
###############################################################################

##############################################################################
# Import modules
################################################################################

# StormTest Client API
import stormtest.ClientAPI as StormTest
# Global return value definitions
import TestEnvironment
# Python module used to create StormTest Image object
import Image

# System libraries
import sys
from time import strftime, localtime

##############################################################################
# Variables
################################################################################

# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName"]

# Trigger image file - this must be present in the same directory
triggerImageFile = "VQAstartImage.png"

# Global variable to indicate VQA has finished
VQAfinished = False

# Output file: we create a .csv file to write the results to, and we fill in some headings.
# This file can be opened with a spreatsheet such as Excel, or a text editor.
resultsFile = "VQA_Results_"+strftime("%d%m%Y_%H%M%S")+".csv"
out = open(resultsFile, 'w')
out.write("frame, MOS, FrameQuality, Jerkiness, Blockiness\n")
```

```
##############################################################################
# Local functions
##############################################################################
def runTest(slotNumber):
    global VQAfinished
    global triggerImageFile

    ############################################################
    # Add here all key sequences to play the video you want to test
    StormTest.PressButton('SKY')
    StormTest.WaitSec(3)
    ############################################################

    #Create a VQAVideoOptions object and fill in some details
    vqaOptions = StormTest.VQACreateVideoOptions()
    #Format - this describes the video format of the source. This is important for MOS scores
- see API documentation
    vqaOptions.Format = "H264"
    #FrameRate = 0 means it will analyse at the same rate as the incoming frames
    vqaOptions.FrameRate = 0
    #Backlog: this is the size of the frame queue. Here we set it to 1500, and we'll choose to
analyse only a total of 1500
    #frames (see below), so we are guaranteed no dropped frames.
    #In reality you may want to analyse for a much longer period so you will need to consider
the backlog size and
    #frame rate carefully.
    vqaOptions.Backlog = 1500
    #Jerkiness: We set it to True (the default). This means that any jerkiness or image
freezing will affect the MOS score.
    vqaOptions.Jerkiness = True
    #Other options are left at default - see API description for details.

    #Create a VQAEngine object and pass in the options object
    vqaEngine = StormTest.VQACreateEngine(vqaOptions)

    """
    #This section shows how you could use a specific image to trigger the start of analysis
    #Open an image we'll use to match as a trigger
    pil = Image.open(triggerImageFile)
    #Convert it to a StormTestImageObject
    startTriggerImage = StormTest.Imaging.StormTestImageObject(pil)
    #Now create a trigger object
    startTrigger = StormTest.VQACreateTriggerFromImage(startTriggerImage,rect=[0, 0, 1920,
1080],threshold=96)
    """
    #Here we'll trigger the start of analysis with a fixed delay. Note that you can also use
FrameCount, an IR keystroke,
    #a Screen Definition Object or other methods to trigger the start of analysis - see API
documentation for details
    startTrigger = StormTest.VQACreateTrigger()
    startTrigger.Delay = 5

    #now we create a trigger object for a stop trigger
    stopTrigger = StormTest.VQACreateTrigger()
    #we choose to stop after 1,500 frames = 1 minute
    stopTrigger.FrameCount = 1500
    stopTrigger.FromNow = False

    #start the engine - it will start analysis when the start trigger condition is satisfied
    #vqaEngine.Start(trigger = startTrigger, callbackFunction = CallbackFunction, slotNo =
slotNumber)
    StormTest.VQAStartAnalysis(vqaEngine, trigger = startTrigger, callbackFunction =
CallbackFunction, slotNo = slotNumber)

    #we can now call the stop function here - it will only stop when it's stop trigger
condition is satisfied
    StormTest.VQAStopAnalysis(vqaEngine, trigger = stopTrigger)

    #here we wait until the callback indicates that VQA analysis has actually finished.
    while not VQAfinished:
        StormTest.WaitSec(1)

    #close the file we used to write the results
    out.close()                         # close the csv file

    #close the VQA analyser completely
    StormTest.VQACloseAnalyzer(vqaEngine)



def CallbackFunction(slotNo, engine, event):
    """
    Callback function for VQAEngine that can be used in Start() method
    Parameters:
      - slotNo: indicates the slot that this callback data is for
      - engine: the engine object being used for VQA analysis
      - event: this indicates an event that the callback is reporting
    Description:
    Here we analyse the event parameter that is passed in. If it is a VideoResult, then we
write the details of that result
    to a results file (opened earlier).
    """

    global VQAfinished

    StormTest.WriteDebugLine("Callback: Event type" + str(event.Type) )

    if event.Type == StormTest.VQAEventType.VideoResult:
        StormTest.WriteDebugLine('Video analysis results event starting at frame number ' +
str(event.Value[0].FrameNumber))
        for i in event.Value:
            out.write(str(i.FrameNumber)+", "+str(i.MOS)+", "+str(i.FrameQuality)+",
"+str(i.Jerkiness)+", "+str(i.Blockiness)+"\n")

    if event.Type == StormTest.VQAEventType.Start:
        StormTest.WriteDebugLine('VQA has started')

    if event.Type == StormTest.VQAEventType.Stop:
        StormTest.WriteDebugLine('VQA has stopped accepting new frames')

    if event.Type == StormTest.VQAEventType.Finish:
        StormTest.WriteDebugLine('VQA has finished last frame')
        VQAfinished = True

    if event.Type == StormTest.VQAEventType.DroppedFrame:
        StormTest.WriteDebugLine(str(event.Value[0]) + ' frames have been dropped by the VQA
engine')

    if event.Type == StormTest.VQAEventType.ResChange:
        StormTest.WriteDebugLine('Resolution has changed. New resolution: ' +
str(event.Value[0:2]) + '. New frame rate: ' + str(event.Value[-1]))

    return None



###############################################################################
# Test Code
################################################################################
# So that this module can be used as reusable module, or as standalone program
if __name__ == '__main__':

    if len(sys.argv) == 1:
        print 'Af slot number must be passed to this test as an argument'
        # No port specified -> test fails
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Verify that the argument is valid
    try:
       # Check that the argument is integer value
        slotNb = int(sys.argv[1])
    except:
        # Invalid parameter found -> test fails
        print 'Invalid parameter passed - only integers are allowed'
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Connect to the StormTest server, e.g. "justin", or "123.123.123.123",
    # and send a comment to help others to identify the purpose of the test
    StormTest.ConnectToServer(TestEnvironment.ServerName, "VQA sample")

    # Reserves a slot on the server and start serial logging
    slotAvailable = StormTest.ReserveSlot( slotNb, TestEnvironment.RC_KeySet, logParam, True )
    # Check if the slot is available
    if slotAvailable == False:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    #Show video
    StormTest.ShowVideo()

    # compare function
    runTest(slotNb)

    #Close video window
    StormTest.CloseVideo()

    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()
    StormTest.ReturnTestResult(StormTest.TM.PASS)
```

template.py" and appendix 19.19 "testUtil.py"

Step 1: import what you need

Firstly we import functions from the file testUtil.py. Notice we only import the part of the file that we need and not the entire file. This is good practice.

```
from testUtil import beginTestStep, endTestStep, setup, path_dict,
pressButtonPath
```

Step 2: call a function that sets up the connection

So far, in all the test scripts which have been written, the steps to establish a connection to the StormTest server were part of the script.

In this example we call a function named setup

```
if __name__ == '__main__':
     setup(runTest, test_name)
```

Looking at this function definition in testUtil.py, all of the steps to connect to the server are now contained in that file. This means that it is no longer necessary to repeat the connection steps and it is simply possible to call the function "setup". In addition the results of your test are also reported back to your StormTest server within this function.

This makes the script cleaner, and really becomes the focus on what you want to achieve.

Step 3: Utilities – part 1

We name a function that is using StormTest API functions and that can be re–used a utility.

Also notice the use of the function name as an argument to the "setup" function. This is supported in Python. The function "runTest" will basically be called from the "setup" function.

Step 4: Test Steps and Utilities – part 2

Now, check the "runTest" function. In it we are using Test Steps. Test Steps are used to define a test scenario. By defining each step and its outcome it will make it easier to understand the goal of your test.

Once again in this example we are using utilities.

```
beginTestStep(step_name)
```

and

```
endTestStep(step_name, step_result, step_comment)
```

Checking the testUtil.py, you see that these functions are actually using StormTest API functions themselves.

```
def beginTestStep(step_name):
      """ begin step """
      StormTest.WriteDebugLine(step_name)
      StormTest.BeginTestStep(step_name)

def endTestStep(step_name, step_result, step_description):
      “””End Step and take a screen capture”””
      file_name = "Screencap_" +
datetime.datetime.now().strftime("%H_%M_%S_%f") + ".png"
      StormTest.CaptureImageEx(None,file_name)
      StormTest.EndTestStep(step_name, step_result, step_description)
```

When you run a test that contains test steps you will see immediate benefits. Checking the Test Results view in the StormTest Developer Suite, you will see that each and every step is logged.

If you check the log file you will see that steps are actually represented as coloured nodes. This makes it way easier to review the results of a test run.

If you check EndTestStep, the function we want to call is StormTest.EndTestStep. But to make our log better we want to take a screen capture when a step ends. So instead of calling those 2 functions each and every time you can write your own sets of function (utilities).

As you write more and more scripts you should have more and more utilities that will make it faster to write your tests.

Step 6: Python Dictionary

Another tool introduced in this script is a python dictionary. Dictionaries allow you to store information. And as in a “paper” dictionary values are indexed.

In this informative example we check if the value “STEP_PATH_1” is in the dictionary, if it is, we process the path with a utility function. The dictionary is defined in the testUtil.py file.

```
if "PATH_STEP_1" in path_dict:
        path = path_dict("PATH_STEP_1")
```

You could use a dictionary to store all kind of values related to your test environment or device under test.

Here is an example of a dictionary embedding other dictionaries used to store information relative to a channel line-up:

```
Channel_line_up=dict(

        REFERENCE_CHANNEL=dict(
                    NAME="BBC 1 East (E)",
                    DCA=101,
                    LINEUP_POS=1,
                    DEF="HD",
                ),
        DCA_101=dict(
                    NAME="BBC 1 East (E)",
                    DCA=101,
                    LINEUP_POS=1,
                    DEF="HD",                  ),
        DCA_102=dict(
                    NAME="BBC 2 England",
                    DCA=102,
                    LINEUP_POS=2,
                    DEF="HD",
                )
)
```

Step 7: Going further

You could use this script as a template for your future scripts.

When you master the concept of utilities and understand the benefits of test steps you should consider having a look at the StormTest Framework, which is based on those simple concepts.

18 Conclusion

If you've reached this point you should be more familiar with some of the different StormTest features that help you test your STB. We recommend using the attached example scripts to deepen your knowledge about StormTest. Of course, they can also be used as a basis for your own scripts. The StormTest API contains many more useful functions that you can experiment with while developing your tests. If the functions you have learned here don't give you your desired results, be sure to have a look at the API where you're likely to find a function that can help you achieve what you're looking for.

We've shown you many different ways to use the StormTest functionality. However this does not mean that our way is the only valid way to use StormTest. This document should be considered as an introductory guide only.

Finally, we wish you a lot of success using StormTest and developing your own scripts!

19 Appendix – Example Scripts

19.1 TestEnvironment.py

```
# Actual StormTest server
ServerName="StormTest_Server"


# User remote control file
# Use default remote
RC_KeySet="default"

# Format is: DEFINE_NAME = String provided during custom remote training
RCPOWER   = "Power"
RCSELECT  = "Select"
RCCHPLUS  = "Channel+"
```

19.2 helloWorld.py

```
# This is a Python comment, this line will not be executed, when running the script

##############################################################################
# Import modules
#################################################################################

# StormTest Client API
import stormtest.ClientAPI as StormTest
# Global return value definitions
import TestEnvironment

##############################################################################
# Variables
#################################################################################

# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName"]

##############################################################################
# Local functions
#################################################################################


##############################################################################
# Test Code
#################################################################################

def test():
    # Connect to the StormTest server, e.g. "jack", or "123.123.123.123",
    # and send a comment to help others to identify the purpose of the test
    StormTest.ConnectToServer(TestEnvironment.ServerName, "Hello World Test")

    # Reserves a slot on the server and start serial logging
    StormTest.ReserveSlot( 5, TestEnvironment.RC_KeySet, logParam, False )

    # Print "Hello World" on console
    print "Hello World"
    # Wait 3 seconds
    StormTest.WaitSec(3)

    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()

    # Return test result
    return (StormTest.TM.PASS)


# So that this module can be used as reusable module, or as standalone program
```

```
if __name__ == '__main__':
    StormTest.ReturnTestResult(test())
```

19.3 videoWindow.py

```
##############################################################################
# Import modules
##############################################################################

# StormTest Client API
import stormtest.ClientAPI as StormTest
# Global return value definitions and active StormTest server
import TestEnvironment

##############################################################################
# Variables
##############################################################################

# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName"]

##############################################################################
# Local functions
##############################################################################

# Function
def runTest(args1 = 10):

    print "Our arguments passed to this function is" + str(args1)
    # Waits for 10 seconds
    StormTest.WaitSec(args1)


##############################################################################
# Test Code
##############################################################################
# So that this module can be used as reusable module, or as standalone program
if __name__ == '__main__':

    # Connect to the StormTest server, e.g. "john", or "123.123.123.123",
    # and send a user string to help others to identify the person reserving this port
    StormTest.ConnectToServer(TestEnvironment.ServerName, "User XY")

    # Reserves a slot on the server and start serial logging
    StormTest.ReserveSlot( 7, TestEnvironment.RC_KeySet, logParam, True )

    # Opens a video window
    StormTest.ShowVideo()

    # Use a function to run stormtest related test
    runTest(5)

    # Close the video window
    StormTest.CloseVideo()

    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()
    # Return test results
    StormTest.ReturnTestResult(StormTest.TM.PASS)
```

19.4 irCommands.py

```
##############################################################################
# Import modules
##############################################################################

# StormTest Client API
```

```
import stormtest.ClientAPI as StormTest
# Global return value definitions
import TestEnvironment

###############################################################################
# Variables
##################################################################################

# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName"]

###############################################################################
# Local functions
##################################################################################

# Function to run test
def runTest( ):

    # Send IR command to STB, or any other test device
    StormTest.PressButton("Power") # "Power" -> must match the string of the recorded IR
command
    StormTest.PressButton(TestEnvironment.RCPOWER) # Now power port again, so that we can send
IR commands

    # Send a tune request via IR -> digits are send
    StormTest.PressDigits("123")
    # Waits for 2 seconds
    StormTest.WaitSec(2)

    StormTest.PressDigits('12345678901234567891234567891346789')
    # Waits for 2 seconds
    StormTest.WaitSec(2)

    StormTest.PressDigits("0123")
    # Waits for 2 seconds
    StormTest.WaitSec(2)

    StormTest.PressDigits(1)
    # Waits for 2 seconds
    StormTest.WaitSec(2)

    # If the IR power level is too low, use this function to adjust it
    StormTest.IRSetSignalPower(3)

    StormTest.PressDigits(123)
    # Waits for 2 seconds
    StormTest.WaitSec(2)


    #Using a Navigator file - this code snippet assumes there is a default navigator
#associated with the DUT in the admin console and the screens listed below exist in the
#navigator file
    StormTest.OpenNavigator()

    #Navigate to the main menu from the default start screen
    StormTest.NavigateTo(“MAIN_MENU”)

    #Navigate to the TV guide screen from the main menu
    StormTest.NavigateTo(“TV_GUIDE”,”MAIN_MENU”)


    # Return from this function
    return

###############################################################################
# Test Code
##################################################################################
# So that this module can be used as reusable module, or as standalone program
if __name__ == '__main__':

    # Connect to the StormTest server, e.g. "jeff", or "123.123.123.123",
    # and send a comment to help others to identify the purpose of the test
    StormTest.ConnectToServer(TestEnvironment.ServerName, "IR Commands")

    # Reserves a slot on the server and start serial logging
    StormTest.ReserveSlot( 5, TestEnvironment.RC_KeySet, logParam, True )
    # Start the video window
    StormTest.ShowVideo()

    #launch the test function
    runTest()

    #Stop Video Window
    StormTest.CloseVideo()

    # Wait 3 seconds
    StormTest.WaitSec(3)

    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()
    # Return test results
    StormTest.ReturnTestResult(StormTest.TM.PASS)
```

19.5 videoParameter.py

```
##############################################################################
# Import modules
#################################################################################

# StormTest Client API
import stormtest.ClientAPI as StormTest
# Global return value definitions
import TestEnvironment

##############################################################################
# Variables
#################################################################################

# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName"]

##############################################################################
# Local functions
#################################################################################

# Function to handle video window thread
def runTest( ):

    # Change bit rate
    StormTest.SetMaxBitRate(1024000)
    # Waits for 10 seconds
    StormTest.WaitSec(10)

    # See if the bit rate has changed
    bitRate = StormTest.GetMaxBitRate()
    if bitRate[0][1] == 1024000:
        print "Bit rate for slot",bitRate[0][0],"has changed"
    else:
        print "Bit rate is not 512000: " ,bitRate[0][1]

    # Change bit rate
    StormTest.SetMaxBitRate(512000)
    # Waits for 10 seconds
    StormTest.WaitSec(10)

    # Try to change bit rate
    StormTest.SetMaxBitRate(64000)
    # Waits for 10 seconds
    StormTest.WaitSec(10)

    # Change encoder format
    StormTest.SetResolution("2CIF")
    # Waits for 10 seconds
    StormTest.WaitSec(10)
    # See if the resolution has changed
    resolution = StormTest.GetResolution()
    if resolution[0][1] == "2CIF":
        print "Resolution for slot",resolution[0][0],"has changed"
    else:
        print "Resolution is wrong: " + resolution[0][1]

    # Change encoder format
    StormTest.SetResolution("CIF")
    # Waits for 10 seconds
    StormTest.WaitSec(10)

    # Try to change bit rate
    StormTest.SetResolution("4CIF")
    # Waits for 10 seconds
    StormTest.WaitSec(10)

    # Change frame rate
    StormTest.SetFrameRate(3)
    # Waits for 10 seconds
    StormTest.WaitSec(10)

    # See if the bit rate has changed
    frameRate = StormTest.GetFrameRate()
    if frameRate[0][1] == 3:
        print "Frame rate for slot",frameRate[0][0],"has changed"
    else:
        print "Frame rate is not 3:",frameRate[0][1]

    # Change frame rate
    StormTest.SetFrameRate(25)
    # Waits for 10 seconds
    StormTest.WaitSec(10)

    # Change frame rate
    StormTest.SetFrameRate(10)
    # Waits for 10 seconds
    StormTest.WaitSec(10)

    # Return from this function
    return

##############################################################################
# Test Code
#################################################################################
# So that this module can be used as reusable module, or as standalone program
if __name__ == '__main__':

    # Connect to the StormTest server, e.g. "jamie", or "123.123.123.123",
    # and send a comment to help others to identify the purpose of the test
    StormTest.ConnectToServer(TestEnvironment.ServerName, "Video Parameter")

    # Reserves a slot on the server and start serial logging
    StormTest.ReserveSlot( 7, TestEnvironment.RC_KeySet, logParam, True )

    # Start the video window
    StormTest.ShowVideo()

    #launch the test function
    runTest()

    #Stop Video Window
    StormTest.CloseVideo()

    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()
    # Return test results
    StormTest.ReturnTestResult(StormTest.TM.PASS)
```

19.6 videoCapturing.py

```
################################################################################
# Import modules
###################################################################################

# StormTest Client API
import stormtest.ClientAPI as StormTest
# Global return value definitions
import TestEnvironment

################################################################################
# Variables
###################################################################################

# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName"]

################################################################################
# Local functions
###################################################################################


################################################################################
# Test Code
###################################################################################
# So that this module can be used as reusable module, or as standalone program
if __name__ == '__main__':

    # Connect to the StormTest server, e.g. "jules", or "123.123.123.123",
    # and send a comment to help others to identify the purpose of the test
    StormTest.ConnectToServer(TestEnvironment.ServerName, "Video Capturing")

    # Reserves a slot on the server and start serial logging
    StormTest.ReserveSlot( 6, TestEnvironment.RC_KeySet, logParam, True )

    # Start to capture the video
    StormTest.StartVideoLog("VideoPrefix")

    # Wait for 20 seconds
    StormTest.WaitSec(20)

    # Stop to capture the video
    StormTest.StopVideoLog()

    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()
    # Return test results
    StormTest.ReturnTestResult(StormTest.TM.PASS)
```

19.7 powerFunctionality.py

```
################################################################################
# Import modules
###################################################################################

# StormTest Client API
import stormtest.ClientAPI as StormTest
# Global return value definitions
import TestEnvironment

################################################################################
# Variables
###################################################################################

# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName"]

################################################################################
```

```
# Local functions
#################################################################################


###############################################################################
# Test Code
#################################################################################
# So that this module can be used as reusable module, or as standalone program
if __name__ == '__main__':

    # Connect to the StormTest server, e.g. "joe", or "123.123.123.123",
    # and send a comment to help others to identify the purpose of the test
    StormTest.ConnectToServer(TestEnvironment.ServerName, "Power Functionality")

    # Reserves a slot on the server and start serial logging
    StormTest.ReserveSlot( 1, TestEnvironment.RC_KeySet, logParam, False )

    # Power on STB and wait 100 seconds. In case it is already powered, nothing will happen
    StormTest.PowerOnSTB(100)

    # Power off STB and wait 100 seconds. In case it is already powered off,
    # nothing will happen
    StormTest.PowerOffSTB(100)

    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()

    # Return test results
    StormTest.ReturnTestResult(StormTest.TM.PASS)
```

19.8 motionDetection.py

```
###############################################################################
# Import modules
#################################################################################

# StormTest Client API
import stormtest.ClientAPI as StormTest
# Global return value definitions
import TestEnvironment

# System library
import sys

###############################################################################
# Variables
#################################################################################

# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName"]

###############################################################################
# Local functions
#################################################################################


###############################################################################
# Test Code
#################################################################################
# So that this module can be used as reusable module, or as standalone program
if __name__ == '__main__':

    if len(sys.argv) == 1:
        print 'Af slot number must be passed to this test as an argument'
        # No port specified -> test fails
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Verify that the argument is valid
    try:
       # Check that the argument is integer value
        slotNb = int(sys.argv[1])
    except:
        # Invalid parameter found -> test fails
        print 'Invalid parameter passed - only integers are allowed'
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Connect to the StormTest server, e.g. "justin", or "123.123.123.123",
    # and send a comment to help others to identify the purpose of the test
    StormTest.ConnectToServer(TestEnvironment.ServerName, "motionDetection")

    # Reserves a slot on the server and start serial logging
    slotAvailable = StormTest.ReserveSlot( slotNb, TestEnvironment.RC_KeySet, logParam, True )

    # Check if the slot is available
    if slotAvailable == False:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Turn STB on
    StormTest.PressButton("Power")

    # Wait 5 seconds
    StormTest.WaitSec(5)
    rect = (10, 10, 150, 100)

    #Using resources
    StormTest.WriteNamedRegion('/MotionRegionResource',rect)

    # Verify that there is motion on screen
    returnValue = StormTest.DetectMotionEx( rect, 3, 5)
    # returnValue = StormTest.DetectMotionEx( rect, 3) -> percentage is optional

    #Using a resource to detect motion
    #returnValue = StormTest.DetectMotionEx( FindNamedRegion('/MotionRegionResource'), 3, 5)

    print "returnValue", returnValue
    # Check return values
    for ret in returnValue:
        slot = ret[0]
        status = ret[1]

        if status == False:
            # Motion was not detected -> print result
            print "No motion deteced on slot",slot
            ret = 0
        else:
            print "Motion detected on slot", slot
            ret = 1


    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()

    # Return test results
    if ret == 1:
StormTest.ReturnTestResult(StormTest.TM.PASS)
    else:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)
```

19.9 setupCompareScreen.py

```
##############################################################################
# Import modules
#################################################################################

# StormTest Client API
import stormtest.ClientAPI as StormTest
# Global return value definitions
import TestEnvironment

##############################################################################
# Variables
#################################################################################
```

```
# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName"]

# Retrun value flag
returnValue = 1

################################################################################
# Local functions
###################################################################################

def runTest( ):
    global returnValue

    ############################################################
    # Put all needed button presses here
    StormTest.PressButton(TestEnvironment.RCSELECT)

    ############################################################

    StormTest.WaitSec(2)

    # Capture the actual screen
    returnValueArray = StormTest.CaptureImageEx(None,"..\cap_refImage.bmp")

    # Check if capture screen was successful
    # returnValueArray: [SlotNo, Statusflag, StormTestImageObject, saveFileFullPath][16]
    for ret in returnValueArray:

        slot = ret[0]
        statusFlag = ret[1]

        if statusFlag:
            filename = ret[3]

        if statusFlag == False:
            # A failure occurred
            print "Failure on slot ",slot
            returnValue = 0
        else:
            print "Captured image is",filename

    # Return from this function
    return

################################################################################
# Test Code
###################################################################################
# So that this module can be used as reusable module, or as standalone program
if __name__ == '__main__':

    # Connect to the StormTest server, e.g. "james", or "123.123.123.123",
    # and send a comment to help others to identify the purpose of the test
    StormTest.ConnectToServer(TestEnvironment.ServerName, "Setup Compare Screen")

    # Reserves a slot on the server and start serial logging
    StormTest.ReserveSlot( 15, TestEnvironment.RC_KeySet, logParam, True )

    # Start the video window
    StormTest.ShowVideo()

    #launch the test function
    runTest()

    #Stop Video Window
    StormTest.CloseVideo()

    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()
    # Return from module
    if returnValue == 1:
        StormTest.ReturnTestResult(StormTest.TM.PASS)
    else:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)
```

19.10 compareScreen.py

```
##############################################################################
# Import modules
###############################################################################

# StormTest Client API
import stormtest.ClientAPI as StormTest
# Global return value definitions
import TestEnvironment

# System library
import sys

##############################################################################
# Variables
###############################################################################

# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName"]

# Image Capture Screen (captured used setupCompareScreen.py)
image_CompScreen = "cap_refImage.bmp"
# Rectangle, area to check
rectangle_compScreen =  (10, 10, 150, 100)


refRegionResource = '/CompareScreenRectResource'
refImageResource = '/RefImageResource'

# Region to check for when comparing images
StormTest.WriteNamedRegion(refRegionResource,(10, 10, 150, 100))

#Image to check for
image_file_obj = open("cap_refImage.bmp","rb").read()
StormTest.WriteNamedImage(refImageResource,(image_file_obj,704,576))

# Return value flag
returnValue = 1

##############################################################################
# Local functions
###############################################################################
def runTest():
    global returnValue

    ##########################################################
    # Add here all key sequences to reach your reference screen
    StormTest.PressButton(TestEnvironment.RCSELECT)

    ##########################################################

    #check reference screen over live video

    returnValueArray = StormTest.WaitImageMatch ( image_CompScreen)
    # Different variations of the same function
    #returnValueArray = StormTest.WaitImageMatch ( image_CompScreen, 85 timeToWait = 5,
waitGap = 1)
    #returnValueArray = StormTest.WaitImageMatch ( image_CompScreen, percent = 85,
includedAreas = rectangle_compScreen, timeToWait = 5, waitGap = 1)


    # returnValueArray = StormTest.WaitImageMatch (
StormTest.FindNamedImage(refImageResource), percent = 85, includedAreas =
StormTest.FindNamedRegion(refRegionResource), timeToWait = 5, waitGap = 1)


    #check the returned value
    #[slotNb, statusFlag,StormTestImageObject,percentage,nbCapturesPerformed, totalTime]
    for ret in returnValueArray:

        statusFlag = ret[1]
        slot = ret[0]
        if statusFlag == False:
            # Image does not match reference
            print "Image does not match reference on slot", slot

            saveStatus = ret[2].Save("cap_resImage.png")
            if saveStatus:
                print "A screen capture has been saved"

            returnValue = 0



###############################################################################
# Test Code
##################################################################################
# So that this module can be used as reusable module, or as standalone program
if __name__ == '__main__':

    if len(sys.argv) == 1:
        print 'Af slot number must be passed to this test as an argument'
        # No port specified -> test fails
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Verify that the argument is valid
    try:
       # Check that the argument is integer value
        slotNb = int(sys.argv[1])
    except:
        # Invalid parameter found -> test fails
        print 'Invalid parameter passed - only integers are allowed'
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Connect to the StormTest server, e.g. "justin", or "123.123.123.123",
    # and send a comment to help others to identify the purpose of the test
    StormTest.ConnectToServer(TestEnvironment.ServerName, "compareScreen")

    # Reserves a slot on the server and start serial logging
    slotAvailable = StormTest.ReserveSlot( slotNb, TestEnvironment.RC_KeySet, logParam, True )

    # Check if the slot is available
    if slotAvailable == False:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    #Show video
    StormTest.ShowVideo()

    # compare function
    runTest()

    #Close video window
    StormTest.CloseVideo()

    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()
    # Return from module
    if returnValue == 1:
        StormTest.ReturnTestResult(StormTest.TM.PASS)
    else:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)
```

19.11 colorDetection.py

```
###############################################################################
# Import modules
##################################################################################

# StormTest Client API
import stormtest.ClientAPI as StormTest
# Global return value definitions
import TestEnvironment
```

```
# System library
import sys

################################################################################
# Variables
###################################################################################

# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName"]

# picked up color variables from a no video screen
rect = [200, 301, 60, 50]
color = [0,0,0]
flatness = 98
tolerances = (0,0,0)

# Rectangle, area to check
rectangle_compScreen =  (10, 10, 150, 100)

# Retrun value flag
returnValue = 1

################################################################################
# Local functions
###################################################################################
def runTest():
    global returnValue

    ############################################################
    # Add here all key sequences to reach your reference screen
    StormTest.PressButton(TestEnvironment.RCPOWER)

    ############################################################

    #check reference screen over live video

    returnValueArray = StormTest.WaitColorNoMatch(color, tolerances ,flatness, includedAreas =
rect,timeToWait = 5,waitGap = 0.5)
    # Different variation of the same function
    #returnValueArray = StormTest.WaitColorNoMatch(color,peakError = 0,includedArea =
None,timeToWait = 5,waitGap = 1)



    #Using resources for regions and colours

#     refRegionResource = '/RegionResource'
#     refColorResource = '/ColorResource'
#     StormTest.WriteNamedRegion(refRegionResource,(10, 10, 150, 100))


#     returnValueArray =
StormTest.WaitColorNoMatch(StormTest.FindNamedColor(refColorResource),includedAreas=StormTest.
FindNamedRegion(refRegionResource), timeToWait = 5,waitGap = 1)


    #check the returned value
    #[slotNb, status,StormTestImageObject,actualColor, actualFlatness, actualPeakError,
nbCaptPerformed, totalTime]
    for ret in returnValueArray:

        statusFlag = ret[1]
        time = ret[7]

        if not statusFlag:
            print "color did not disappear"
            returnValue = 0
        else:
            print str("color vanished in %2.5f seconds" % time)



################################################################################
# Test Code
```

```
#################################################################################
# So that this module can be used as reusable module, or as standalone program
if __name__ == '__main__':

    if len(sys.argv) == 1:
        print 'Af slot number must be passed to this test as an argument'
        # No port specified -> test fails
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Verify that the argument is valid
    try:
       # Check that the argument is integer value
        slotNb = int(sys.argv[1])
    except:
        # Invalid parameter found -> test fails
        print 'Invalid parameter passed - only integers are allowed'
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Connect to the StormTest server, e.g. "justin", or "123.123.123.123",
    # and send a comment to help others to identify the purpose of the test
    StormTest.ConnectToServer(TestEnvironment.ServerName, "Color Detection")

    # Reserves a slot on the server and start serial logging
    slotAvailable = StormTest.ReserveSlot( slotNb, TestEnvironment.RC_KeySet, logParam, True )

    # Check if the slot is available
    if slotAvailable == False:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    #Show video
    StormTest.ShowVideo()

    # compare function
    runTest()

    #Close video window
    StormTest.CloseVideo()

    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()
    # Return from module
    if returnValue == 1:
        StormTest.ReturnTestResult(StormTest.TM.PASS)
    else:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)
```

19.12 observeSerial.py

```
##############################################################################
# Import modules
#################################################################################

# StormTest Client API
import stormtest.ClientAPI as StormTest
# Global return value definitions
import TestEnvironment

# System library
import sys

##############################################################################
# Variables
#################################################################################

# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName_log1"]
logParam2 = [9600, 8, "none", 1, "none", "TestName_log2"]

# Predefine what a 'warning' string looks like
Warning_String = "Warning"
```

```
##############################################################################
# Local functions
#################################################################################
# Serial monitoring callback function
def callBackWarning(slotNo, key, string):
    # Print the string that was found on the port
    print 'The callback looking for "'+key+'" on slot',slotNo,"was called"
    print "It observed the following string:",string

    # This function can be called, if logging of this string is not needed anymore
    StormTest.RemoveObserve(Warning_String)

##############################################################################
# Test Code
#################################################################################
# So that this module can be used as reusable module, or as standalone program
if __name__ == '__main__':

    if len(sys.argv) == 1:
        print 'Af slot number must be passed to this test as an argument'
        # No port specified -> test fails
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Verify that the argument is valid
    try:
       # Check that the argument is integer value
        slotNb = int(sys.argv[1])
    except:
        # Invalid parameter found -> test fails
        print 'Invalid parameter passed - only integers are allowed'
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Connect to the StormTest server, e.g. "justin", or "123.123.123.123",
    # and send a comment to help others to identify the purpose of the test
    StormTest.ConnectToServer(TestEnvironment.ServerName, "Observe Serial")

    # Reserves a slot on the server and start serial logging
    slotAvailable = StormTest.ReserveSlot( slotNb, TestEnvironment.RC_KeySet, logParam, False
)

    # Check if the slot is available
    if slotAvailable == False:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Start serial monitoring -> if "Error" occurs it will be logged to a logfile
    StormTest.AddObserve("Error")

    # Start serial monitoring -> if "Warning" occurs the callback function will be called
    StormTest.AddObserve(Warning_String, callBackWarning)

    # Wait 5 seconds
    StormTest.WaitSec(5)

    # This function can be called, if logging of this string is not needed anymore. Or it will
be released anyway as soon as the port is released
    StormTest.RemoveObserve("Warning")

    # Uncomment this function if your STB accepts commands via serial interface
    #StormTest.SendCommandViaSerial("Tune_To_CH101", "Tuned", 3)

    # Change serial logging paramter, and keep logging into the same file
    StormTest.ChangeSerialParameters(logParam2)

    # Wait 5 seconds
    StormTest.WaitSec(5)

    # Stop serial port
    StormTest.StopSerialLog()
    #StormTest.StopSerialLog(2) # Works only if port 2 has been reserved

    # Start serial port again, with new parameter
    StormTest.StartSerialLog(logParam)
    #StormTest.StartSerialLog(logParam, 2) # Works only if port 2 has been reserved

    # Press a button and wait 5 seconds
    StormTest.PressButton("Channel+")
    StormTest.WaitSec(5)

    # Stop serial port
    StormTest.StopSerialLog()
    #StormTest.StopSerialLog(2) # Works only if port 2 has been reserved

    # Get the serial logs
    returnValueArray = StormTest.GetObserveLog()

    # Check if a return value parameter is present at all
    if returnValueArray == []:
        print "No log information has been found!"
    else:
        for slot, fileSize, logFile in returnValueArray:
            if fileSize == 0:
                # No log file found
                print "No observe log file has been found on slot ",slot
            else:
                # Here you can process the data
                print "Log file exists: " + logFile

    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()
    StormTest.ReturnTestResult(StormTest.TM.PASS)
```

19.13 compareScreenOffset.py

```
##############################################################################
# Import modules
#################################################################################

# StormTest Client API
import stormtest.ClientAPI as StormTest
# Global return value definitions
import TestEnvironment

# System library
import sys

##############################################################################
# Variables
#################################################################################

# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName"]

# Return value flag
returnValue = 1

##############################################################################
# Local functions
#################################################################################
def runTest():
    global returnValue
    ###########################################################
    # User configurable parameters

    ###########################################################
    myIcon =  "film_icon.png"          #The image to compare against
    numComparisonSteps = 4             #The number of compare steps to perform

    #The values below are based on the first expected location of the image to compare
against.
    x = 640     # coordinate on the x-axis
    y = 110     # coordinate on the y-axis
    ###########################################################
    # Add here all key sequences to reach your reference screen

    ###########################################################
    returnValueArray = StormTest.CaptureImageEx (None,"cap.png")
    StormTest.WaitSec(2)
    location = (x,y)
    for csRet in returnValueArray:
        csSlot, csResult, csImageObject, csImageName = csRet
        allOk = True
        if csResult == False:
            returnValue = False
            break

        while numComparisonSteps > 0:
            offset = 48        #The height of the image is used as the offset because it's a
stepwise vertical comparison
            returnArray = StormTest.WaitIconMatch(myIcon,98,location,timeToWait = 2)

            for ciRet in returnArray:
                ciSlot, ciResult, ciImageObject,ciPercentage,ciNbCap,ciTime = ciRet
                if ciResult == False:
                    returnValue = False
                    numComparisonSteps = 0
                    break
                else:
                    y = y + offset
                    numComparisonSteps = numComparisonSteps - 1
                    StormTest.WaitSec(2)

##############################################################################
# Test Code
#################################################################################
# So that this module can be used as reusable module, or as standalone program
if __name__ == '__main__':

    if len(sys.argv) == 1:
        print 'Af slot number must be passed to this test as an argument'
        # No port specified -> test fails
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Verify that the argument is valid
    try:
       # Check that the argument is integer value
        slotNb = int(sys.argv[1])
    except:
        # Invalid parameter found -> test fails
        print 'Invalid parameter passed - only integers are allowed'
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Connect to the StormTest server, e.g. "justin", or "123.123.123.123",
    # and send a comment to help others to identify the purpose of the test
    StormTest.ConnectToServer(TestEnvironment.ServerName, "compareScreenOffset")

    # Reserves a slot on the server and start serial logging
    slotAvailable = StormTest.ReserveSlot( slotNb, TestEnvironment.RC_KeySet, logParam, True )

    # Check if the slot is available
    if slotAvailable == False:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)


    #Show video
    StormTest.ShowVideo()

    # compare function
    runTest()

    #Close video window
    StormTest.CloseVideo()

    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()
    # Return from module
    if returnValue == 1:
        StormTest.ReturnTestResult(StormTest.TM.PASS)
    else:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)
```

19.14 OCRsample.py

```
##############################################################################
# Import modules
#################################################################################

# StormTest Client API
import stormtest.ClientAPI as StormTest
# Global return value definitions
import TestEnvironment

# System library
import sys

##############################################################################
# Variables
#################################################################################

# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName"]

# Rectangle, area to check
rectangle_noVideo =  (182, 218, 339, 49)

#Or using a region resource
refRegionNoVideoResource = ‘/RegionResource’
StormTest.WriteNamedRegion(refRegionNoVideoResource, rectangle_noVideo)


# Return value flag
returnValue = 1

##############################################################################
# Local functions
#################################################################################
def runTest():
    global returnValue

    ###########################################################
    # Add here all key sequences to reach your reference screen
    StormTest.PressButton(TestEnvironment.RCPOWER)

    ###########################################################

    #OCR the no video screen

    ocr_result = StormTest.OCRSlot(rectangle_noVideo)

    # Different variation of the same function usging pre filtering
    #if OCRSetImageFiltering([("Blur","Gaussian3x3"),("Color","Gray")]):
    #   ocr_result = StormTest.OCRSlot(rectangle_noVideo)

    #Different variation of the same function using resources
    #ocr_result = StormTest.OCRSlot(FindNamedRegion(rectangle_noVideo))

    #check the returned value
    if ocr_result[2] > 0:             # Check the total no. of symbols
        for slot, txt in ocr_result[4]:
        # Remove the spaces from the start and end of the text
            tmp = txt.strip()

            if tmp == "Expected Text":
                print "Expected text on screen on slot", slot
            else:
                print "Error - unexpected text on screen for slot", slot
    else:
        print "Error when using OCR"
```

```
##############################################################################
# Test Code
#################################################################################
# So that this module can be used as reusable module, or as standalone program
if __name__ == '__main__':

    if len(sys.argv) == 1:
        print 'Af slot number must be passed to this test as an argument'
        # No port specified -> test fails
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Verify that the argument is valid
    try:
       # Check that the argument is integer value
        slotNb = int(sys.argv[1])
    except:
        # Invalid parameter found -> test fails
        print 'Invalid parameter passed - only integers are allowed'
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Connect to the StormTest server, e.g. "justin", or "123.123.123.123",
    # and send a comment to help others to identify the purpose of the test
    StormTest.ConnectToServer(TestEnvironment.ServerName, "Observe Serial")

    # Reserves a slot on the server and start serial logging
    slotAvailable = StormTest.ReserveSlot( slotNb, TestEnvironment.RC_KeySet, logParam, True )

    # Check if the slot is available
    if slotAvailable == False:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    #Show video
    StormTest.ShowVideo()

    # compare function
    runTest()

    #Close video window
    StormTest.CloseVideo()

    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()
    # Return from module
    if returnValue == 1:
        StormTest.ReturnTestResult(StormTest.TM.PASS)
    else:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)
```

19.15 ZapSample.py

```
##############################################################################
# Import modules
#################################################################################

# StormTest Client API
import stormtest.ClientAPI as StormTest
# Global return value definitions
import TestEnvironment

# System library
import sys

##############################################################################
# Variables
#################################################################################

# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName"]
```

```
# Retrun value flag
returnValue = 1

##############################################################################
# Local functions
#################################################################################
def runTest():
    global returnValue

    ##########################################################
    # Add here all key sequences to reach your reference screen

    ##########################################################

    # Region to check for determining zap time
    StormTest.WriteNamedRegion('/zapSample/zapTimerRegion',(10, 10, 600, 300))

    if StormTest.ReserveVideoTimer():

        # set STB to starting point
        StormTest.PressButton("SKY")
        StormTest.PressDigits(101)
        StormTest.WaitSec(5)
        # setup the zap timer
        zapstatus = StormTest.StartZapMeasurement(
                                    TestEnvironment.RCCHPLUS,
                                    0,
                                    StormTest.FindNamedRegion('/zapSample/zapTimerRegion'),
                                    timeout=5.0)
        StormTest.WaitSec(1)
        # Press Channel+ and wait 5 seconds
        StormTest.PressButton(TestEnvironment.RCCHPLUS)
        StormTest.WaitSec(5)

        print StormTest.GetRawZapMeasurements()
        zapTime =  StormTest.GetZapTimes()
        print str(zapTime)

        if zapTime[0][1]:
            print str("channel zapped in %2.3f seconds" %zapTime[0][3])

        StormTest.FreeVideoTimer()

##############################################################################
# Test Code
#################################################################################
# So that this module can be used as reusable module, or as standalone program
if __name__ == '__main__':

    if len(sys.argv) == 1:
        print 'Af slot number must be passed to this test as an argument'
        # No port specified -> test fails
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Verify that the argument is valid
    try:
       # Check that the argument is integer value
        slotNb = int(sys.argv[1])
    except:
        # Invalid parameter found -> test fails
        print 'Invalid parameter passed - only integers are allowed'
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Connect to the StormTest server, e.g. "justin", or "123.123.123.123",
    # and send a comment to help others to identify the purpose of the test
    StormTest.ConnectToServer(TestEnvironment.ServerName, "zap sample")

    # Reserves a slot on the server and start serial logging
    slotAvailable = StormTest.ReserveSlot( slotNb, TestEnvironment.RC_KeySet, logParam, True )

    # Check if the slot is available
    if slotAvailable == False:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    #Show video
    StormTest.ShowVideo()

    # compare function
    runTest()

    #Close video window
    StormTest.CloseVideo()

    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()
    # Return from module
    if returnValue == 1:
        StormTest.ReturnTestResult(StormTest.TM.PASS)
    else:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)
```

19.16 HSTSampleEPG.py

```
##############################################################################
# Import modules
#################################################################################

# StormTest Client API
import stormtest.ClientAPI as StormTest
# Global return value definitions
import TestEnvironment

# System library
import sys

##############################################################################
# Variables
#################################################################################

# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName"]

# Retrun value flag
returnValue = 1


##############################################################################
# Local functions
#################################################################################
def runTest():
    global returnValue

    ###########################################################
    # Add here all key sequences to reach your reference screen

    ###########################################################
    maxFrames = 50

    # Rectangle, area to check for EPG
    StormTest.WriteNamedRegion('/zapSample/EPGRefRegion',(10, 10, 600, 300))

    #NOTE: The script assumes a reference EPG image is stored under "cap_refImage.bmp".
    image_file_obj = open("cap_refImage.bmp","rb").read()
    StormTest.WriteNamedImage('/zapSample/EPGRef',(image_file_obj,704,576))

    if StormTest.ReserveVideoTimer():

        #enable capturing of frames
        status = StormTest.EnableCaptureZapFrames(maxFrames)
        if status[0][1] != True:
            print "Error in EnableCaptureZapFrames"
            return
        #set STB to starting point
        StormTest.PressButton("SKY")
        StormTest.PressDigits(101)
        StormTest.WaitSec(5)
        #setup the HST to trigger on TV Guide button
        status = StormTest.TriggerHighSpeedTimer(TestEnvironment.RCEPG,0,5)
        StormTest.WaitSec(1)
        # Press TV Guide and wait
        StormTest.PressButton(TestEnvironment.RCEPG)
        StormTest.WaitSec(5)
        #get the number of frames saved
        numFrames = StormTest.GetCountImagesSaved()
        print "numFrames: ", numFrames
        found = False
        for i in range(numFrames[0][1]):
            #get a frame
            frame = StormTest.GetRawZapFrame(i)
            print "\nFrame " + str(i)

            #check this frame against a reference image (assumed to have been stored as
resource)
            status = StormTest.CompareImageEx(
                                       StormTest.FindNamedImage('/zapSample/EPGRef'),
                                       frame[0][1],
                                       95,
                                       StormTest.FindNamedRegion('/zapSample/EPGRefRegion'))
            if status[0] == True:
                 hardwareCaptureRate =
StormTest.GetRawInputInfo()[0][1]['HardwareCaptureRate']
                timeTaken = float(i)/float(hardwareCaptureRate)
                print "\nResulting time = " + str(timeTaken) + " seconds\n"
                found = True
                break
            else:
                print "no match"
        if not found:
            print "No match found - time could not be measured"

        #release the video timer
        StormTest.FreeVideoTimer()

##############################################################################
# Test Code
#################################################################################
# So that this module can be used as reusable module, or as standalone program
if __name__ == '__main__':

    if len(sys.argv) == 1:
        print 'Af slot number must be passed to this test as an argument'
        # No port specified -> test fails
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Verify that the argument is valid
    try:
       # Check that the argument is integer value
        slotNb = int(sys.argv[1])
    except:
        # Invalid parameter found -> test fails
        print 'Invalid parameter passed - only integers are allowed'
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Connect to the StormTest server, e.g. "justin", or "123.123.123.123",
    # and send a comment to help others to identify the purpose of the test
    StormTest.ConnectToServer(TestEnvironment.ServerName, "zap sample")

    # Reserves a slot on the server and start serial logging
    slotAvailable = StormTest.ReserveSlot( slotNb, TestEnvironment.RC_KeySet, logParam, True )

    # Check if the slot is available
    if slotAvailable == False:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    #Show video
    StormTest.ShowVideo()

    # compare function
    runTest()
    #Close video window
    StormTest.CloseVideo()

    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()
    # Return from module
    if returnValue == 1:
        StormTest.ReturnTestResult(StormTest.TM.PASS)
    else:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)
```

19.17 VQASample.py

```
# ==========================================================================
# Copyright (c) 2010-2015 Silicon & Software Systems Ltd.(S3 Group).
# All rights reserved. This source code and any compilation or derivative
# thereof is the proprietary information of Silicon & Software Systems Ltd.
# and is confidential in nature. Use of this source code is subject to the
# terms of the applicable Silicon & Software Systems Ltd. license agreement.
#
#  MODULE: VQASample.py
#
#  DESCRIPTION: Simple python script showing how to use the Video
#               Quality Analysis (VQA) module.
#               Note that an extra license is required for VQA.
#
###############################################################################

##############################################################################
# Import modules
#################################################################################

# StormTest Client API
import stormtest.ClientAPI as StormTest
# Global return value definitions
import TestEnvironment
# Python module used to create StormTest Image object
import Image

# System libraries
import sys
from time import strftime, localtime

##############################################################################
# Variables
#################################################################################

# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName"]

# Trigger image file - this must be present in the same directory
triggerImageFile = "VQAstartImage.png"

# Global variable to indicate VQA has finished
VQAfinished = False

# Output file: we create a .csv file to write the results to, and we fill in some headings.
# This file can be opened with a spreatsheet such as Excel, or a text editor.
resultsFile = "VQA_Results_"+strftime("%d%m%Y_%H%M%S")+".csv"
out = open(resultsFile, 'w')
out.write("frame, MOS, FrameQuality, Jerkiness, Blockiness\n")

##############################################################################
# Local functions
#################################################################################
def runTest(slotNumber):
    global VQAfinished
    global triggerImageFile

    ############################################################
    # Add here all key sequences to play the video you want to test
    StormTest.PressButton('SKY')
    StormTest.WaitSec(3)
    ###########################################################

    #Create a VQAVideoOptions object and fill in some details
    vqaOptions = StormTest.VQACreateVideoOptions()
    #Format - this describes the video format of the source. This is important for MOS scores
- see API documentation
    vqaOptions.Format = "H264"
    #FrameRate = 0 means it will analyse at the same rate as the incoming frames
    vqaOptions.FrameRate = 0
    #Backlog: this is the size of the frame queue. Here we set it to 1500, and we'll choose to
analyse only a total of 1500
    #frames (see below), so we are guaranteed no dropped frames.
    #In reality you may want to analyse for a much longer period so you will need to consider
the backlog size and
    #frame rate carefully.
    vqaOptions.Backlog = 1500
    #Jerkiness: We set it to True (the default). This means that any jerkiness or image
freezing will affect the MOS score.
    vqaOptions.Jerkiness = True
    #Other options are left at default - see API description for details.

    #Create a VQAEngine object and pass in the options object
    vqaEngine = StormTest.VQACreateEngine(vqaOptions)

    """
    #This section shows how you could use a specific image to trigger the start of analysis
    #Open an image we'll use to match as a trigger
    pil = Image.open(triggerImageFile)
    #Convert it to a StormTestImageObject
    startTriggerImage = StormTest.Imaging.StormTestImageObject(pil)
    #Now create a trigger object
    startTrigger = StormTest.VQACreateTriggerFromImage(startTriggerImage,rect=[0, 0, 1920,
1080],threshold=96)
    """
    #Here we'll trigger the start of analysis with a fixed delay. Note that you can also use
FrameCount, an IR keystroke,
    #a Screen Definition Object or other methods to trigger the start of analysis - see API
documentation for details
    startTrigger = StormTest.VQACreateTrigger()
    startTrigger.Delay = 5

    #now we create a trigger object for a stop trigger
    stopTrigger = StormTest.VQACreateTrigger()
    #we choose to stop after 1,500 frames = 1 minute
    stopTrigger.FrameCount = 1500
    stopTrigger.FromNow = False

    #start the engine - it will start analysis when the start trigger condition is satisfied
    #vqaEngine.Start(trigger = startTrigger, callbackFunction = CallbackFunction, slotNo =
slotNumber)
    StormTest.VQAStartAnalysis(vqaEngine, trigger = startTrigger, callbackFunction =
CallbackFunction, slotNo = slotNumber)

    #we can now call the stop function here - it will only stop when it's stop trigger
condition is satisfied
    StormTest.VQAStopAnalysis(vqaEngine, trigger = stopTrigger)

    #here we wait until the callback indicates that VQA analysis has actually finished.
    while not VQAfinished:
        StormTest.WaitSec(1)

    #close the file we used to write the results
    out.close()                         # close the csv file

    #close the VQA analyser completely
    StormTest.VQACloseAnalyzer(vqaEngine)



def CallbackFunction(slotNo, engine, event):
    """
    Callback function for VQAEngine that can be used in Start() method
    Parameters:
      - slotNo: indicates the slot that this callback data is for
      - engine: the engine object being used for VQA analysis
      - event: this indicates an event that the callback is reporting
    Description:
    Here we analyse the event parameter that is passed in. If it is a VideoResult, then we
write the details of that result
    to a results file (opened earlier).
    """

    global VQAfinished

    StormTest.WriteDebugLine("Callback: Event type" + str(event.Type) )

    if event.Type == StormTest.VQAEventType.VideoResult:
        StormTest.WriteDebugLine('Video analysis results event starting at frame number ' +
str(event.Value[0].FrameNumber))
        for i in event.Value:
            out.write(str(i.FrameNumber)+", "+str(i.MOS)+", "+str(i.FrameQuality)+",
"+str(i.Jerkiness)+", "+str(i.Blockiness)+"\n")

    if event.Type == StormTest.VQAEventType.Start:
        StormTest.WriteDebugLine('VQA has started')

    if event.Type == StormTest.VQAEventType.Stop:
        StormTest.WriteDebugLine('VQA has stopped accepting new frames')

    if event.Type == StormTest.VQAEventType.Finish:
        StormTest.WriteDebugLine('VQA has finished last frame')
        VQAfinished = True

    if event.Type == StormTest.VQAEventType.DroppedFrame:
        StormTest.WriteDebugLine(str(event.Value[0]) + ' frames have been dropped by the VQA
engine')

    if event.Type == StormTest.VQAEventType.ResChange:
        StormTest.WriteDebugLine('Resolution has changed. New resolution: ' +
str(event.Value[0:2]) + '. New frame rate: ' + str(event.Value[-1]))

    return None



##############################################################################
# Test Code
################################################################################
# So that this module can be used as reusable module, or as standalone program
if __name__ == '__main__':

    if len(sys.argv) == 1:
        print 'Af slot number must be passed to this test as an argument'
        # No port specified -> test fails
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Verify that the argument is valid
    try:
       # Check that the argument is integer value
        slotNb = int(sys.argv[1])
    except:
        # Invalid parameter found -> test fails
        print 'Invalid parameter passed - only integers are allowed'
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    # Connect to the StormTest server, e.g. "justin", or "123.123.123.123",
    # and send a comment to help others to identify the purpose of the test
    StormTest.ConnectToServer(TestEnvironment.ServerName, "VQA sample")

    # Reserves a slot on the server and start serial logging
    slotAvailable = StormTest.ReserveSlot( slotNb, TestEnvironment.RC_KeySet, logParam, True )

    # Check if the slot is available
    if slotAvailable == False:
        StormTest.ReturnTestResult(StormTest.TM.FAIL)

    #Show video
    StormTest.ShowVideo()
    # compare function
    runTest(slotNb)

    #Close video window
    StormTest.CloseVideo()

    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()
    StormTest.ReturnTestResult(StormTest.TM.PASS)
```

19.18 template.py

```
# ==========================================================================
# Copyright (c) 2010-2014 Silicon & Software Systems Ltd.(S3 Group).
# All rights reserved. This source code and any compilation or derivative
# thereof is the proprietary information of Silicon & Software Systems Ltd.
# and is confidential in nature. Use of this source code is subject to the
# terms of the applicable Silicon & Software Systems Ltd. license agreement.
#
#  MODULE: template.py
#
#  DESCRIPTION:template that shows how to use utilities and test steps
#
################################################################################
# Import modules
##################################################################################

# StormTest Client API
import stormtest.ClientAPI as StormTest
# import function from the testUtil file
from testUtil import beginTestStep, endTestStep, setup, path_dict, pressButtonPath

###############################################################################
# Variables
##################################################################################
test_name = "template"
###############################################################################
# Test Code
##################################################################################

def runTest():

    ###############
    #1st Test Step
    ###############
    step_name = "1st Step"
    beginTestStep(step_name)

    StormTest.WriteDebugLine("Core of the 1st Step")

    #check if the entry is part of the dictionnary
    if "PATH_STEP_1" in path_dict:
        path = path_dict["PATH_STEP_1"]

        pressButtonPath(path)


    step_result = StormTest.TM.PASS
    step_comment = "PASSED"
    endTestStep(step_name, step_result, step_comment)

    #continue or exit
    if step_result == StormTest.TM.FAIL:
        return step_result

    ###############
    #2nd Test Step
    ###############
    step_name = "2nd Step"
    beginTestStep(step_name)

    StormTest.WriteDebugLine("Core of the 2nd Step")

    if "PATH_STEP_2" in path_dict:
        path = path_dict["PATH_STEP_2"]

        pressButtonPath(path)

    step_result = StormTest.TM.FAIL
    step_comment = "FAILED"
    endTestStep(step_name, step_result, step_comment)

    ###############
    # end of the test
    ###############
    return step_result
# So that this module can be used as reusable module, or as standalone program
if __name__ == '__main__':
    setup(runTest, test_name)
```

19.19 testUtil.py

```
# ==========================================================================
# Copyright (c) 2010-2014 Silicon & Software Systems Ltd.(S3 Group).
# All rights reserved. This source code and any compilation or derivative
# thereof is the proprietary information of Silicon & Software Systems Ltd.
# and is confidential in nature. Use of this source code is subject to the
# terms of the applicable Silicon & Software Systems Ltd. license agreement.
#
#  MODULE: testUtil.py
#
#  DESCRIPTION:  function to ease the conection to the StormTest server
#
###############################################################################


###############################################################################
# Import modules
################################################################################
import os
import sys
# StormTest Client API
import stormtest.ClientAPI as StormTest
import TestEnvironment

import datetime

###############################################################################
# Variables
################################################################################

# Serial port parameters
logParam = [57600, 8, "none", 1, "none", "TestName"]

path_dict=dict(

        PATH_STEP_1= ["Right","Righ","Ok"],
        PATH_STEP_2= ["Left","Left","Ok"],
    )
###############################################################################
# utility functions
################################################################################
def pressButtonPath(path = [], wait_time = 3):
    """ press button of the list given"""
    for button in path:
        StormTest.PressButton(button)
        StormTest.WaitSec(wait_time)
```

```
def beginTestStep(step_name):
    """ begin step """
    StormTest.WriteDebugLine(step_name)
    StormTest.BeginTestStep(step_name)

def endTestStep(step_name, step_result, step_description):
    """ End Step and take a screen shot"""
    file_name = "Screencap_" + datetime.datetime.now().strftime("%H_%M_%S_%f") + ".png"
    StormTest.CaptureImageEx(None,file_name)
    StormTest.EndTestStep(step_name, step_result, step_description)

# Function to run test
def setup(function_name, test_name):

    # Connect to the StormTest server, e.g. "jeff", or "123.123.123.123",
    # and send a comment to help others to identify the purpose of the test
    StormTest.ConnectToServer(TestEnvironment.ServerName, test_name)

    # Reserves a slot on the server and start serial logging
    StormTest.ReserveSlot( TestEnvironment.slot_nb, TestEnvironment.RC_KeySet, logParam, True
)

    # Start the video window
    StormTest.ShowVideo()

    #launch the test function
    test_result = function_name()

    #Stop Video Window
    StormTest.CloseVideo()

    # Wait 3 seconds
    StormTest.WaitSec(3)

    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()
    # Return test results
    StormTest.ReturnTestResult(test_result)
```

19.20 SDO_parser.py

```
# ==========================================================================
# Copyright (c) 2010-2014 Silicon & Software Systems Ltd.(S3 Group).
# All rights reserved. This source code and any compilation or derivative
# thereof is the proprietary information of Silicon & Software Systems Ltd.
# and is confidential in nature. Use of this source code is subject to the
# terms of the applicable Silicon & Software Systems Ltd. license agreement.
#
#  MODULE: SDO_parser.py
#
#  DESCRIPTION: Simple python script showing how to use a Screen Definition
#               Object, and containing a re-usable utility script to analyse
#               SDO results.
#
###############################################################################

##############################################################################
# Import modules
################################################################################
# StormTest Client API
import stormtest.ClientAPI as StormTest
# Global return value definitions
import sys
import Image
import datetime
##############################################################################
# Variables
################################################################################

slotNo = 3
```

```
###########################################################################
# Local functions
##############################################################################
def sdoParser(sdo):

    #get the image

    captured_image = "cap_" + datetime.datetime.now().strftime("%H_%M_%S_%f") + ".jpg"
    if sdo.Image is not None:
        sdo.Image.SaveToFile(captured_image)
    else:
        print "no image captured in SDO"

    for region in sdo.Regions:
        #if region.VerifyStatus:
        #common properties
        print "region.Name",region.Name
        print "region.Comment",region.Comment
        print "region.ReturnScreenImage",region.ReturnScreenImage

        if isinstance(region,StormTest.OCRRegion):
            print "region.VerifyStatus",region.VerifyStatus
            print "region.Rect",region.Rect
            print "region.Verify",region.Verify
            print "region.PassOnNoMatch",region.PassOnNoMatch
            print "region.expectedText",region.ExpectedText
            print "region.MaxDistance", region.MaxDistance
            print "region.AutoCorrect",region.AutoCorrect
            print "region.Languages",region.Languages
            print "region.ImageFilters",region.ImageFilters
            print "region.LegacyInvert", region.LegacyInvert
            print "region.ResultText",region.ResultText
            print "region.RawText",region.RawText
            print "region.OCRStatistics",region.OCRStatistics
            StormTest.WriteDebugLine("OCR criteria", StormTest.DebugLevel.INFO)

            if sdo.Image is not None:
                if region.Rect is not None:
                    StormTest.WriteDebugLine(" captured image = " + captured_image + ", rect =
" + str(list(region.Rect)))
                else:
                    StormTest.WriteDebugLine(" captured image = " + captured_image + ", rect =
[]")

        elif isinstance(region,StormTest.ImageRegion):
            print "region.VerifyStatus",region.VerifyStatus
            print "region.Rect",region.Rect
            print "region.Verify",region.Verify
            print "region.PassOnNoMatch",region.PassOnNoMatch
            print "region.IconMode",region.IconMode
            print "region.ReferenceImage",region.ReferenceImage
            print "region.Threshold",region.Threshold
            print "region.ActualMatch",region.ActualMatch
            StormTest.WriteDebugLine("Image criteria", StormTest.DebugLevel.INFO)

            ref_image = "ref_" + datetime.datetime.now().strftime("%H_%M_%S_%f") + ".png"
            region.ReferenceImage.Save(ref_image, 100)

            if sdo.Image is not None:
                msg = " Expected image = " + str(ref_image) + ", captured image = " +
str(captured_image) + ", rect = " + str(list(region.Rect))
            else:
                msg = " Expected image = " + str(ref_image) + ", rect = " +
str(list(region.Rect))
            StormTest.WriteDebugLine(msg)


        elif isinstance(region, StormTest.ColorRegion):
            print "region.VerifyStatus",region.VerifyStatus
            print "region.Rect",region.Rect
            print "region.Verify",region.Verify
            print "region.PassOnNoMatch",region.PassOnNoMatch
            print "region.ReferenceColor",region.ReferenceColor
            print "region.Tolerance",region.Tolerance
            print "region.Flatness",region.Flatness
            print "region.PeakError",region.PeakError
            print "region.ActualColor",region.ActualColor
            print "region.ActualFlatness",region.ActualFlatness
            print "region.ActualPeakError",region.ActualPeakError

            StormTest.WriteDebugLine("Color criteria",StormTest.DebugLevel.INFO)

            msg = "color used= " + str(region.ReferenceColor) + " tolerances = " +
str(region.Tolerance) + \
                            " with flatness " + str(region.Flatness) + "% and peak error " +
str(region.PeakError) + "%"

            StormTest.WriteDebugLine(msg, StormTest.DebugLevel.INFO)

            msg = "color found= " + str(region.ActualColor) +" with flatness " +
str(region.ActualFlatness) +\
            "% and peak error " + str(region.ActualPeakError) + "%"

            StormTest.WriteDebugLine(msg,StormTest.DebugLevel.INFO)

            #display the reference color
            ref_color_image = Image.new('RGB', tuple((250, 250)),
tuple(region.ReferenceColor))
            ref_color_image_name = "ColorRef_" +
datetime.datetime.now().strftime("%H_%M_%S_%f") + ".png"
            ref_color_image.save((St.GetLogFileDirectory() + "\\" + ref_color_image_name),
"PNG")

            StormTest.WriteDebugLine(str(ref_color_image_name))

            if sdo.Image is not None:
                msg = " Expected image = " + str(ref_color_image_name) + ", captured image = "
+ str(captured_image) + ", rect = " + str(list(region.Rect))
            else:
                msg = " Expected image = " + str(ref_color_image_name) + ", rect = " +
str(list(region.Rect))
            StormTest.WriteDebugLine(msg)

        elif isinstance(region,StormTest.MotionRegion):
            print "region.VerifyStatus",region.VerifyStatus
            print "region.Rect",region.Rect
            print "region.Verify",region.Verify
            print "region.PassOnNoMatch",region.PassOnNoMatch
            print "region.MotionThreshold",region.MotionThreshold
            print "region.Timeout",region.Timeout
            print "region.ActualMotion",region.ActualMotion

            StormTest.WriteDebugLine("Motion criteria",StormTest.DebugLevel.INFO)
            if sdo.Image is not None:
                if region.Rect is not None:
                    StormTest.WriteDebugLine(" captured image = " + captured_image + ", rect =
" + str(list(region.Rect)))
                else:
                    StormTest.WriteDebugLine(" captured image = " + captured_image + ", rect =
[]")


            #fromat the output to get an actual
        elif isinstance(region,AudioRegion):
            print "region.VerifyStatus",region.VerifyStatus
            print "region.Verify",region.Verify
            print "region.PassOnNoMatch",region.PassOnNoMatch
            print "region.MotionThreshold",region.AudioThreshold
            print "region.Timeout",region.Timeout
            print "region.Channel",region.Channel
            print "region.ActualAudio",region.ActualAudio

        elif isinstance(region,ScreenDefinition):
            sdoParser(region)


###############################################################################
# Test Code
##################################################################################
# So that this module can be used as reusable module, or as standalone program
if __name__ == '__main__':
    for arg in sys.argv:
        print arg

    # Connect to the StormTest server, e.g. "jack", or "123.123.123.123",
    # and send a user string to help others to identify the person reserving this port
    StormTest.ConnectToServer("stormtest16hd", "ronanj")

    if not StormTest.ReserveSlot(slotNo,"default",[57600,8,'none',1,'none',"Test_name"]):
        print "Failed to reserve slot"
        StormTest.ReleaseServerConnection()
        sys.exit(1)

    res = StormTest.OpenNavigator("OrchidSDO")
    screenDefName, res, sdoResult = StormTest.NavigateTo("HD")[0][1][2][0]

    sdoParser(sdoResult['Screen Definition'])

    sdo = StormTest.ReadNamedScreenDefinition("/Ronan's Resources/OrchidHD_SDO")[0][1]
    res = StormTest.WaitScreenDefMatch(sdo,5)

    sdoParser(res[0][1])

    # End of test - release ports back to the server
    StormTest.ReleaseServerConnection()

    # Return test results
    StormTest.ReturnTestResult(StormTest.TM.PASS)
```