API Document

Accenture StormTest Development Center

Document ID :ST-11009

Revision Date: December 2016

Product Version: 3.4

Web: http://www.accenturestormtest.com

Contents

1 Preface

1.1 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 the StormTest Client API in detail. It lists all functions available to test script developers. A detailed description of each function is provided which includes input parameters, returned parameters and a code example. This document does not include any step-by-step guide to using the API. For that the reader is encouraged to read the StormTest Programmer's Guide [2] for examples of how to use the functions in the API.

1.3 Who Should Read This Document

This document is targeted at the following group:

- Test writers

1.4 Related Documentation

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

- StormTest Developer Suite Users Manual
- StormTest Programmers Guide
- StormTest Client API
- StormTest Hardware Installation Guides (HV01, HV04, HV16)
- StormTest Software Installation Guide
- StormTest Server Monitor Users Manual
- StormTest Administration Console Users Guide
- StormTest Administration Tools Users Guide

The latest version of these documents can always be found on our support website, in the "Docs" section.

2 Function Documentation

2.1 Server Interaction API

The Server Interaction API functions are used to establish a connection to the StormTest server and to reserve a slot on the StormTest cabinet.

Enums

- enum SignalRecoveryMode
- enum TM
- enum TestResult
- enum SystemCapability
- enum LogRegionStyle

Functions

- GetFacilityData
- GetServerNames
- GetServerField
- GetStbField
- ConnectToServer
- ReleaseServerConnection
- ReserveSlot
- ReserveStb
- SetCurrentSubslot
- GetCurrentSubslot
- FreeSlot
- KillSlot
- ResetSlot
- SetSignalRecoveryMode
- GetSignalRecoveryMode
- GetLogicalReservedSlots
- GetPhysicalAllocations
- IsUnderDaemon
- ReturnTestResult
- BeginTestStep
- EndTestStep
- GetClientVersion
- CheckSlots
- GetReservedStatus
- WaitSec
- HasCapability
- GetCapabilities
- BeginLogRegion
- EndLogRegion

2.1.1 Detailed Description

The Server Interaction API functions are used to establish a connection to the StormTest server and to reserve a slot on the StormTest cabinet.

API functions are also provided that allow a test script to retrieve information about all the servers in the test facility.

A test facility is a collection of one or more StormTest servers that are physically co-located and connected to the same LAN. Each StormTest facility will have one and only one PC that is designated the StormTest Configuration Server. It will host the StormTest database, web server, subversion repository and license server. For an installation that consists of only one rack, the StormTest server in that rack will most likely also be the Configuration server.

The sort of information that the facility APIs allow a user to retrieve includes the names of all the servers in the facility, the configuration of each server and the set-top-boxes in each slot of each server.

2.1.2 Function Documentation

2.1.2.1 `GetFacilityData` (configServer)

Description

Gets the raw facility data from the StormTest configuration server. This can then be passed to other functions to retrieve more specific data.
Every StormTest facility has a single configuration server and via this server, information about the entire test facility can be retrieved.

Parameters:

← **configServer String:** The location of the StormTest configuration server. Can be a host-name or IP address.

Returns:

rawData **List:** The raw facility data.

Example:

```
>>rawData = GetFacilityData('config_server')
or
>>rawData = GetFacilityData('10.0.1.2')
```

2.1.2.2 `GetServerNames` (rawData)

Description

Get the list of servers in the test facility. This information is extracted from the raw facility data.

Parameters:

← **rawData List:** The raw data from the GetFacilityData() function.

Returns:

serverList **List:** A list of strings representing the hostnames of servers in the facility.

Example:

```
>>servers = GetServerNames(GetFacilityData(configServer))
>>print servers
>>['server1', 'server2']
```

2.1.2.3 `GetServerField` (rawData, serverName, fieldWanted)

Description

Get a specific field from the configuration data of a StormTest server. All fields can be retrieve by requesting field **'all'**.

Parameters:

← **rawData String:** The raw data from the GetFacilityData() function.

← **serverName String:** The name of the server that you want to get information on (normally got from a call to GetServerNames()).

← **fieldWanted String:** The actual field required, or 'all' for all fields.

Returns:

data **Dictionary/String/Integer**: If one field was requested, the return will be a String or an Integer. If all fields are requested then a dictionary of requested data in the form **field:value** is returned.

Example:

```
>>rawData = GetFacilityData(configServer)
>>servers = GetServerNames(rawData)
>>for server in servers:
>>  allFields = GetServerField(rawData,server,'all')
>>  oneField = GetServerField(rawData,server,'SlotCount')
>>
>>print allFields
>>{'DefaultBitRate': 2097000, 'DefaultFrameRate': 25, 'SlotCount': 16,.....}
>>
>>print oneField
>>16
```

2.1.2.4 `GetStbField` (rawData, serverName, slotNumber, fieldWanted)

Description

Get a specific field from the DUT data of the DUT instance in the slot of a StormTest server. All fields can be retrieve by requesting field **'all'**.

Parameters:

← **rawData String:** The raw data from the GetFacilityData() function.

← **serverName String:** The name of the server that the DUT is in (normally got from a call to GetServerNames()).

← **slotNumber Integer:** The slot number that the DUT is in.

← **fieldWanted String:** The actual field required, or 'all' for all fields.

Returns:

data **Dictionary/String/Integer**: If one field was requested, the return will be a String or an Integer. If all fields are requested then a dictionary of requested data in the form **field:value** is returned.

Example:

```
>>rawData = GetFacilityData(configServer)
>>servers = GetServerNames(rawData)
>>for server in servers:
>>  slotCount = GetServerField(rawData,server,'SlotCount')
>>  for slot in range(slotcount):
>>      allFields = GetStbField(raw,server,slot+1,'all')
>>      oneField = GetStbField(raw,server,slot+1,'Manufacturer')
>>
>>print allFields
>>{'Manufacturer': 'Generic', 'HardwareVersion': 'zapper',.....}
>>
>>print oneField
>>Generic
```

2.1.2.5 `ConnectToServer` (**name**, **description** = `'-'`)

Description

Establishes the connection to the StormTest server.

Parameters:

← **name String:** A server host name or ip address.

← **description String:** (optional) A description of the client connecting.

Returns:

retStatus **Bool:** 1 always. On a failure to connect throws a sys.exit exception and exits the script. NOTE: Ever since version 1, this has been the behavior - the documentation was incorrect to claim a False return on error.

Example:

```
>>ret = ConnectToServer("stormtest01","TestUser Connecting to Server")
or
>>ret = ConnectToServer("193.120.62.187","TestUser Connecting to Server")
>>print ret
>>1             #Operation was a success
```

2.1.2.6 `ReleaseServerConnection` ()

Description

This function releases the previously established connection from the StormTest server as well as all previously reserved slots.

Parameters:

None

Returns:

retStatus **Bool:** 1 success, 0 fail

Example:

```
>>ret = ReleaseServerConnection()
>>print ret
>>1              #Operation was a success
```

2.1.2.7 `ReserveSlot` (**slotNo**, **signalDb**, **serialParams** = `[]`, **videoFlag** = `True`)

Description

This function reserves a slot on the StormTest server. If serial port parameters are passed into the function, then StormTest will commence logging of the serial data from the DUT in that slot.
Note that this function will fail if a *signalDb* is passed that does not exist on the StormTest server.

Parameters:

← **slotNo Integer:** Slot number to reserve.

← **signalDb String:** IR signal database to use when controlling the DUT in the slot. If the string *'default'* is used, the StormTest will use the default IR signal database for the DUT in that slot.

← **serialParams List:** (optional) Serial port parameters to use for communicating with the serial port on the DUT (see Serial Port Setup Parameters).

← **videoFlag Bool:** (optional) Setting this to true causes video to be streamed to the client. This is necessary for video logging, viewing of video on the client and client side video analysis. If this is set to false, video logging is not possible and image capture, detect motion and similar functions perform server side capture and pass the images to the client for processing. If a large number of such captures are performed in a short period of time, this can lead to IP port exhaustion and test failures. The limit is API dependent but would be around 1000 per 4 minute interval across all slots employing server side capture (it is a server issue, not a client issue). Setting the flag to false reduces CPU loading on the client as video decode is not needed.

Returns:

retStatus **Bool:** 1 success, 0 fail

Example:

```
>># Reserve slot using the default IR signal database
>>ret = ReserveSlot(1, "default",[19200,8,'none',1,'none',"Test_name"])
>>print ret
>>1             # Operation was a success - slot is reserved
>>
>># Reserve slot using a specific IR signal database
>>ret = ReserveSlot(2, "MY_SIGNAL_DB.txt",[19200,8,'none',1,'none',"Test_name"])
>>print ret
>>1             # Operation was a success - slot is reserved
```

Note:

When the *videoFlag* = False, there is no video streaming from the StormTest server.
This reduces the load on the network and may be useful for remote testing.

2.1.2.8 `ReserveStb` (**facilityData**, **signalDb**, **modelName**, **serialParams** = `[]`, **videoFlag** = `True`, **stbSnum** = `""`, **stbManf** = `""`, **stbHwVersion** = `""`, **smartcard** = `""`)

Description

This function reserves a DUT of a particular type in a StormTest test facility. If serial port parameters are passed into the function, then StormTest will commence logging of the serial data from the DUT.
The function will reserve the first DUT that matches the model name. To be more specific, the test can also specify that the DUT must have a specific serial number/viewing card number etc.

Parameters:

← **facilityData List:** The raw data from the GetFacilityData() function.

← **signalDb String:** IR signal database to use when controlling the DUT.

← **modelName String:** The name of the DUT model that you wish to reserve.

← **serialParams List:** (optional) Serial port parameters to use for communicating with the serial port on the DUT (see Serial Port Setup Parameters).

← **videoFlag Bool:** (optional) If the test script is not performing video analysis, then this parameter must be set to False.

← **stbSnum String:** (optional) The test script can optionally reserve a DUT with a specific serial number.

← **stbManf String:** (optional) The test script can optionally reserve a DUT from a specific manufacturer.

← **stbHwVersion String:** (optional) The test script can optionally reserve a DUT with a specific hardware version.

← **smartcard String:** (optional) The test script can optionally reserve a DUT with a specific smartcard number.

Returns:

retStatus **Integer:** 1 success, 0 fail

Example:

```
>>print ReserveStb(GetFacilityData(configServer),"MY_SIGNAL_DB.txt","Zapper")
>>1     # A zapper DUT has been reserved
```

2.1.2.9 `SetCurrentSubslot` (**subslotNo**, **slotNo** = 0)

Description

In an HS64 system, one slot has four subslots and this function sets the current subslot within a slot used for testing. Due to the nature of video switching, there will be a delay (several seconds) before the client video stabilises after setting the sub slot. Scripts should wait until the video has settled before performing image processing functions such as CaptureImageEx, DetectMotionEx etc.

Parameters:

← **subslotNo Integer:** Subslot number to be used for testing.

← **slotNo Integer:** (optional): Slot number to set the current subslot for.

Returns:

retStatus **Integer:** 1 success, 0 fail

Example:

```
>># Set the current subslot to 4 for slot 1
>>ret = SetCurrentSubslot(4, 1)
>>print ret
>>1             # Operation was a success - subslot 4 was set to be the current subslot for slot 1
>>
>># Set the current subslot to 4 for all reserved slots
>>ret = SetCurrentSubslot(4)
>>print ret
>>1             # Operation was a success - subslot 4 was set to be the current subslot for all reserved slots
>>
```

Note:

If the *slotNo* parameter is omitted the default is to set the current subslot for all slots reserved

2.1.2.10 `GetCurrentSubslot` (**slotNo** = 0)

Description

In an HS64 system, one slot has four subslots and this function gets the current subslot used for testing.

Parameters:

← **slotNo Integer:** (optional): Slot number to get the current subslot for.

Returns:

currentSubslot **Integer Array:** An array of tuples consisting of slot number and subslot number.

Example:

```
# Given that slot 1 and 2 have been reserved previously in the test and the current subslot for slot 1 is 3 and the cu
# Get current subslots of all reserved slots
>>ret=GetCurrentSubslot()
>>print ret
>>[[1, 3], [2, 4]]

# Get current subslot of slot 2
>>ret=GetCurrentSubslot(2)
>>print ret
>>4
```

Note:

If the *slotNo* parameter is omitted the current subslot returned are for all slots reserved.

2.1.2.11 `FreeSlot` (slotNo)

Description

This function frees a previously reserved slot on the StormTest server.

Parameters:

← **slotNo Integer:** Slot number to free. If slot was not previously reserved, then no action will be taken.

Returns:

Integer: return value may be ignored. 0 is returned if the slotNo is not of an integer type.

Example:

```
>>FreeSlot(2)
```

2.1.2.12 `KillSlot` (slotNo)

Description

This function forcibly frees a slot. It is different from FreeSlot() in that it can be used on slots that the test has not reserved. For this reason it should be used with great care and should not be used in normal tests. Any test in progress on the slot will be disrupted.

Parameters:

← **slotNo Integer:** Slot number to be freed.

Returns:

Bool: True if a server supports this feature

Example:

```
>>KillSlot(2)
```

2.1.2.13 `ResetSlot` (**slotNo** = 0)

Description

This function resets the slot. It has no effect on SD systems. For 4K systems, its effect is limited to resetting the EEDID to the default value (it should not have been changed but if, for some reason it became corrupted then this call will restore it). On HD servers it performs a full software reset of the slot. It will disrupt the audio and video stream for several seconds so should only be used when necessary and when the slot is doing nothing of importance for the test. It has been found that occasionally, the HDMI input could get in an unusual state such that a reset would help. This has been largely superceded by safe mode in release 3.3.0 so this call should only be used in rare circumstances. ResetSlot() performs one activity that normal safe mode resolution change does not: It causes HDCP renegotiation. If ResetSlot is called while a resolution change is in progress, the behaviour depends on safe/fast mode: If fast mode is in operation, ResetSlot converts it to safe mode and notes the need for HDCP renegotiation and returns. If in safe mode, it simply notes the need for HDCP renegotiatieon but no other action is taken.

Parameters:

← **slotNo Integer:** (optional) This sets the mode for a specific slot only, 0 for all reserved slots

Returns:

None

Example:

```
>>ResetSlot(6)
```

2.1.2.14 `SetSignalRecoveryMode` **(mode, slotNo** = 0**)**

Description

This function modifies the behaviour of the hardware (and thus the system) when HDMI signal has been lost. It has no impact on SD and 4K systems (but does not throw an exception or indicate any error). HD systems can, on rare occasions, "freeze" if the signal resolution changes after it has been lost while the hardware input capture logic is active. The default mode (from release 3.3.1) is to turn off the hardware input capture when signal loss is detected. When a signal is detected then the hardware input capture is turned on at the newly detected resolution. However, this can take up to 6 seconds. If your application is time critical and you know the device connected to the StormTest system won't change its resolution then you can set Fast mode. However, if the device does change its resolution, even briefly (perhaps especially briefly) then there is a risk that the slave server will freeze. In that case, the recovery procedure is a manual power cycle of the slave server. Until power cycle, tests on any of the 4 slots on the slave server will fail. Script writers are advised to exercise extreme caution before using Fast mode and use it for the minimum possible time.

Parameters:

← **mode SignalRecoveryMode** : The mode to use for the signal recovery operation

← **slotNo Integer:** (optional) This sets the mode for a specific slot only, 0 for all reserved slots

Returns:

None

Example:

```
>>SetSignalRecoveryMode(SignalRecoveryMode.Fast, 8)
```

Note:

When a slot is released it automatically is set to Safe mode.

2.1.2.15 `GetSignalRecoveryMode` (**slotNo** = 0)

Description

This function returns the current signal recovery mode as set by SetSignalRecoveryMode. Although the signal recovery mode has no impact on SD or 4K systems, this function is supported. This function was introduced in release 3.3.1 of StormTest

Parameters:

← **slotNo Integer:** (optional) This gets the mode for a specific slot only, 0 for all reserved slots

Returns:

SignalRecoveryMode if slotNo is non zero, otherwise a list of lists with each sub list being [slot, SignalRecoveryMode] Python will actually return the associated integer from the enumeration instead of the object type.

Example:

```
>>print GetSignalRecoveryMode(8)
>>0
>>print GetSignalRecoveryMode(0)
>>[[1,0],[4,1],[5,0]]
```

2.1.2.16 `GetLogicalReservedSlots` **()**

Description

This function returns a list of logical slots reserved by the test script - these are the slot numbers passed to ReserveSlot() and thus can be used for any later function call which requires a slot number. This is only important when running under the client daemon as the actual slot used (the physical slot) is NOT the same as the slot reserved via ReserveSlot() (the logical slot). This function, together with GetPhysicalAllocations() allows the script to find out either piece of information, depending on how the script wishes to use the information. When NOT under the client daemon, physical slots == logical slots.

Parameters:

None

Returns:

List: a list of the logical slots reserved by the test.

Example:

```
>> print GetLogicalReservedSlots()
>>  [1,2,3]
```

2.1.2.17 `GetPhysicalAllocations` ()

Description

This function returns the list of physical slots allocated to the test, along with the physical server name and port used. It can be called before ConnectToServer() or ReserveSlot() to gather information about how the StormTestScheduler has allocated resources to the script It will return None if the script is not running under the client daemon (ie has NOT been scheduled by the StormTest Scheduler)

Parameters:

None

Returns:

Tuple: a tuple of 2 items, the server name:port and the list of physical slots allocated to the test. None if not under daemon

Example:

```
>> print GetPhysicalAllocations()
>> ("stormtest01:8000", [5,6])
```

2.1.2.18 `IsUnderDaemon` ()

Description

This returns a simple boolean indicating whether the test is running under the client daemon (i.e. has been scheduled or run remotely) or not.

Parameters:

none

Returns:

Bool: True if running under daemon. False if test invoked from command line manually (or by some means other than StormTest scheduler)

Example:

```
>> print IsUnderDaemon()
>> False
```

2.1.2.19 `ReturnTestResult` (**testResult**, **saveResultToDB** = `True`)

Description

This function is used to return a test result to the calling process. Calling this function will cause the test script process to exit, so this should be the very last function call in a test script. Specify the test result using the TM enum.

2.1.2.20 `BeginTestStep` (**testStepName)**

Description

This function allows a test script writer to divide the test script into "Test Steps". Each test step in the script is then able to write a result to the StormTest database. The database will also capture the length of each test step. Versions prior to 2.0 called this function BeginTestCase and that exists for compatibility purposes

Parameters:

← **testStepName String:** The name of the test step.

Returns:

Bool: True if ok, False if error.

Example:

```
>> BeginTestStep("test step one")
```

2.1.2.21 `EndTestStep` (**testStepName, testResult, testComment, startTimeStamp** = 0)

Description

This function should be called at the end of each test step. BeginTestStep() must have been called previously with the same *testStepName*. Versions prior to 2.0 called this function EndTest-Step and that exists for compatibility purposes.

Parameters:

← **testStepName String:** The name of the test step.

← **testResult Integer:** The result of the test step. This can be any integer value and is not significant to StormTest. It will be recorded in the database and displayed in the StormTest Developer Suite.

← **testComment String:** A comment regarding the test step. This can be any value and is not significant to StormTest. It will be recorded in the database and displayed in the StormTest Developer Suite.

← **startTimeStamp Integer:** (optional) The timestamp representing when the test step started. Use 0 to allow StormTest to find this automatically. Value is in seconds since the Unix epoch (1/1/1970)

Returns:

Bool: True if ok, False if error.

Example:

```
>> BeginTestStep("test step one")
>> ...
>> EndTestStep("test step one", 111, 'A comment')
```

2.1.2.22 `GetClientVersion` ()

Description

This function returns the software version of the StormTest client application.

Parameters:

None

Returns:

softwareVersion **String:** The client software version.

Example:

```
>>ret = GetClientVersion()
>>print ret
>>1.1.0
```

2.1.2.23 `CheckSlots` ()

Description

This function returns the status of all slots in the StormTest cabinet, independent of whether they are reserved by the user or not.

Parameters:

None

Returns:

slotInfo **List:** An array of 16 lists. Each one corresponding to a slot in the StormTest cabinet. See Returned CheckSlots data.

Example:

```
>>ret=CheckSlots()
>>print ret
>>[[1, 'Baggio', 'Stress Test'], [1, 'Zidan', 'Live Pause Test'],
   [0, ' ', ' '], [0, ' ', ' '], [0, ' ', ' '], [0, ' ', ' '],
   [0, ' ', ' '], [0, ' ', ' '], [0, ' ', ' '], [0, ' ', ' '],
   [0, ' ', ' '], [0, ' ', ' '], [0, ' ', ' '], [0, ' ', ' '],
   [0, ' ', ' '], [0, ' ', ' ']]
```

Note:

This function will wait a maximum of 10 seconds before returning.
These are the valid returned slot indicator values:

- 0: The slot is free.
- 1: The slot is reserved.
- 2: The slot is available but powered off.

2.1.2.24 `GetReservedStatus` (**server** = `None`, **slot** = `None`)

Description

This function returns information about the locked and reserved status of 1 or more slots, independent of whether they are reserved or not. It is not necessary to call ConnectToServer() or ReserveSlot() before calling this function. You may call SetMasterConfigurationServer() to select the facility to query but it is not necessary - the default will be used if no explicit configuration server is set. This function is available in release 3.3.4.

Parameters:

← **server String** : (optional) The name of the server where the slot of interest is located. If omitted, data is returned for all servers in the facility.

← **slot Integer** : (optional) The slot number where the slot of interest is located. If omitted, data is returned for all slots on the specified server. It is not an error to specify a slot but set the server to 'None' - however, in this case data is returned for all slots on all servers in the facility.

Returns:

slotInfo **List** of Dictionary : In all cases a list of dictionary objects is returned, one for each slot. This is the case even when an explicit server/slot is specified. Each dictionary contains the following key value pairs:
Server - the server name
Slot - the slot number
Locked - True/False indicating whether a slot is locked, see note[1]
LockUser - Name of user who has locked the slot. None if slot is not locked
LockStart - A DateTime object indicating when the lock started. None is slot is not locked
Reserved - True/False indicating whether the slot is reserved, see note[1]
ReservedUser - Name of user who has reserved the slot, None if slot is not reserved
ReservationEnd - A DateTime object indicating when the reservation ends, None if slot is not reserved
ActiveSchedule - The name of the schedule that is active on this slot, None if the slot is not reserved

Note:

1. A slot is 'Locked' if a test / user has called ReserveSlot on the slot. It is 'Reserved' if there is a non shared, non disabled schedule allocated to the slot. This terminology matches DeveloperSuite GUI.

Example:

```
>>ret=GetReservedStatus('myserver', 9)
>>print ret
>>[{'Server': 'myserver', 'Slot': 9, 'Locked': True, 'LockUser': 'me',
    'LockStart': datetime.datetime(2016, 4, 25, 14, 57, 58, 145000),
    'Reserved': True, 'ReservedUser': 'me',
    'ReservationEnd': datetime.datetime(2016, 4, 25, 19, 0, 0, 0),
    'ActiveSchedule': 'my schedule'}]

>>ret=GetReservedStatus()
>>print ret
>>[{'Server': 'myserver', 'Slot': 1, ...}, {'Server': 'myserver', 'Slot': 2, ...}, ...
   {'Server': 'server2', 'Slot': 1, ...}, ...]
```

2.1.2.25 `WaitSec` (sec)

Description

This function pauses a script execution for *sec* number of seconds.

Parameters:

← **sec Float:** Time in seconds.

Returns:

None

Example:

```
>>WaitSec(0.5)
```

2.1.2.26 `HasCapability` (**capabilityList**, **server** = `None`)

Description

This function tests for the availability of features of the API, system or a server.

Parameters:

← **capabilityList List:** list of capabilities

← **server String:** (optional) server name to query capabilites for. if the capability being tested is related to a server then the server name must be provided if HasCapability is called before ConnectToServer.

Returns:

status **bool:** If all capabilities are supported, then True is returned, else False.

Example:

```
>>HasCapability([SystemCapability.Serial],"StormTest01")
>>True   # So, server StormTest01 has serial capability
```

2.1.2.27 `GetCapabilities` (**server** = `None`)

Description

This function returns the full list of system and server capabilities.

Parameters:

← **server String:** (optional) server name to query capabilites for. If None and ConnectToServer has been called then the connected server is assumed. If called with None prior to ConnectToServer then just the system (non server specific) capabilities are returned.

Returns:

capabilities **List:** List of strings corresponding to the SystemCapability enumerated types

Example:

```
>>GetCapabilities("StormTest01")
>>['IR', 'Power', 'Serial', 'BasicAV', 'OCR', 'VTA', 'NamedResources']
```

2.1.2.28 `BeginLogRegion` (**name**, **style** = `None`, **comment** = `None`)

Description

This function marks the beginning of a region in the log file. Regions are shown in the log file with the name and are 'closed up' with the contents not visible until the user clicks to open and reveal them. This can be useful to demarcate parts of a log for ease of review. The name is mandatory and is case sensitive. The style is one of the LogRegionStyle values to indicate the style of the region.

Parameters:

← **name String:** The name of the region.

← **style LogRegionStyle:** (optional) the style of the region

← **comment String:** (optional) An optional comment, appended to the name for display purposes.

Returns:

None

Example:

```
>>BeginLogRegion("set up DUT")
```

2.1.2.29 `EndLogRegion` (**name**, **style** = `None`, **comment** = `None`)

Description

This function marks the end of a region in the log file. Regions are shown in the log file with the name and are 'closed up' with the contents not visible until the user clicks to open and reveal them. This can be useful to demarcate parts of a log for ease of review. The name is mandatory and must match a previous BeginLogRegion. It is case sensitive. The style and comment, if not None, will override any style or comment set with BeginLogRegion. It is not necessary to set the style or comment on a log region (either the begin or end)

- in that case, the default style will be used.

Parameters:

← **name String:** The name of the region.

← **style LogRegionStyle:** (optional) the style of the region

← **comment String:** (optional) An optional comment, appended to the name for display purposes.

Returns:

None

Example:

```
>>EndLogRegion("set up DUT", LogRegionStyle.Pass, "All OK, Version 1.0 found")
```

2.2 IR command API

The IR command functions control the irNetBox.

Functions

- PressButton
- EnablePressDigitsCriticalSection
- PressDigits
- GetCountIRInterfaces
- SetIRControlInterface
- GetIRControlInterface
- IRSetSignalPower

2.2.1 Detailed Description

The IR command functions control the irNetBox.

The functions in this section of the API are used to emulate the remote control keys and also to modify the power of the IR signal being sent to the box.

2.2.2 Function Documentation

2.2.2.1 `PressButton` (**button**, **slotNo** = 0)

Description

This function sends an IR key press to the DUT(s).

Parameters:

← **button String** or NamedString: The remote control key you wish to emulate. If the button is a NamedString then the resource is looked up. The format of the string is the same as the .mac file saved from the StormTest Developer Suite record macro feature. It allows you to send multiple buttons with individual delays after each button press.

← **slotNo Integer:** (optional) This sends the IR pulse to a specific slot only.

Returns:

status **Integer:** Returns 1 if the button press succeeded, otherwise 0.

Example:

```
>>PressButton(button)   #Press button for all slots
or
>>PressButton(button,7) #Press button for slot 7 only
```

Note:

The *button* parameter is a generic key name that will be translated into an IR pulse, depending on the remote type selected.
If the *slot* parameter is left out then the IR pulse will be sent on all slots that were reserved.

2.2.2.2 `EnablePressDigitsCriticalSection` (**enable** = `None`, **timeout** = `10`)

Description

This function enables or disables the critical section around the PressDigits() call. Prior to release 1.4.2, the IR was locked for the duration of a PressDigits() call. To improve performance with multiple users this was changed to avoid locking the IR. So multiple users could interleave IR key presses with another script's PressDigits() call. You can restore the 1.4.1 and earlier behaviour either by using EnablePressDigitsCriticalSection(True) at the start of the script. Calling the function with no parameters will return the current setting without making a change to behaviour.

Parameters:

← **enable Boolean:** True to put a critical section around every PressDigits() call to restore 1.4.1 and earlier behaviour. False to avoid the critical section and use 1.4.2 and later behaviour. This function affects all later calls to PressDigits() and may be called at any time. Default behaviour is no critical section.

← **timeout Integer:** Timeout on critical section, in seconds. Default is 10 and should be set to be the overall expected time of the longest PressDigits() call - never shorter.

Returns:

Tuple: (enable, timeout) as a mirror of the most recent parameters used.

Example:

```
>>EnablePressDigitsCriticalSection(True, 5)
```

2.2.2.3 `PressDigits` (**digits**, **slotNo** = 0, **waitSec** = 0.1)

Description

This function sends a sequence of IR key presses, representing a sequence of digits to the DUT(s). Typically this command will be used to switch the DUT to a particular channel. Alternatively it can also be used to send a PIN or any other sequence of digits. All other characters are ignored. Note that the digits can be passed as an integer or as a string. If the string of digits includes one or more leading zeros then it **MUST** be sent as a string.

Parameters:

← **digits Integer/String**: The sequence of digits you wish to send to the DUT(s).

← **slotNo Integer:** (optional) This sends the IR pulse on a specific slot only.

← **waitSec Float:** (optional) The interval in seconds between digits being pressed.

Returns:

status **Integer:** Returns 1 if the digit presses succeeded, otherwise 0.

Example:

```
>>PressDigits(123)       # Tune all reserved slots to channel 123
or
>>PressDigits('0123',7)  # Tune slot 7 to channel 0123 (digits sent as a string)
```

Note:

The *digits* parameter is a numeric argument that will be translated into a series of IR pulses, depending on the remote type selected.
If the *slot* parameter is left out then the IR pulse will be sent on all slots that were reserved.

2.2.2.4 `GetCountIRInterfaces` ()

Description

This function returns the number of IR control interfaces available. By default there is only one, but it is possible for a StormTest server to be configured with a secondary IR interface.

Parameters:

None

Returns:

Integer: Number of IR interfaces defined for the StormTest server

Example:

```
>>print GetCountIRInterfaces()
>>1
```

2.2.2.5 `SetIRControlInterface` **(interfaceIndex, slotNo** = 0**)**

Description

This function sets the IR interface to use for the PressButton(), PressDigits() and IRSetSignalPower() functions. The default interface is 0 and others start at 1. Inteface 0 is always available and is the default after reserving a slot. All standard systems have just one IR interface. Unless you are working on a known non standard server, you should not use this function.

Parameters:

← **interfaceIndex Integer:** The index of the IR interface to use.

← **slotNo Integer:** (optional) This sets the IR interface on a specific slot only.

Returns:

1 on success, 0 on a server error and None if slot is not reserved.

Example:

```
>>SetIRControlInterface(1)
```

2.2.2.6 `GetIRControlInterface` (**slotNo** = 0)

Description

This function gets the current IR interface.

Parameters:

← **slotNo Integer:** (optional) This fets the IR interface for a specific slot only.

Returns:

List: A list of List, one for each slot. Values are [SlotNo, InterfaceNo]. If a slot is passed in, the result is a single integer for the interface No.

Example:

```
>>print GetIRControlInterface()
>>[[1, 0]]    # Slot 1 is using IR interface 0
>>
>>print GetIRControlInterface(2)
>>1           # Slot 2 is using IR interface 1
```

2.2.2.7 `IRSetSignalPower` (**powerLevel**, **slotNo** = 0)

Description

This function sets the IR signal output power level.

Parameters:

← **powerLevel Integer:** The signal output power level.

← **slotNo Integer:** (optional) This changes the power level on a specific slot only.

Returns:

None

Example:

```
>>IRSignalPower(2,1) #Sets the signal output power level to 2 on slot 1
  or
>>IRSignalPower(2)   #Sets the signal output power level to 2 for all reserved slots
```

Note:

A valid power level is 1-3 where 1 means low, 2 means medium, 3 means high.
If the *slotNo* parameter is omitted the default is to set the signal output power level for all slots reserved.
If the power level is set too high, it may cause interference with other adjacent DUTs.

2.3 Navigator API

These functions control the high level navigation API.

Functions

- OpenNavigator
- OpenNavigatorFile
- CloseNavigator
- GetNavigatorScreens
- ValidateNavigatorScreen
- NavigateTo
- ValidateNavigator
- StopNavigator
- GetValidNavigators
- GetCurrentNavigator
- ConvertNavigatorToScreenDef
- UploadPublicNavigator
- DownloadPublicNavigator
- ListPublicNavigators

2.3.1 Detailed Description

These functions control the high level navigation API.

This is a new feature introduced in version 3.0 of StormTest providing a simple to use high level navigation function. A Navigator is designed using StormTest Developer Suite and can be saved to a file for use local to the script or uploaded using StormTest Developer Suite to the central repository for shared use. It can be associated with a DUT using the StormTest Admin Console.

2.3.2 Function Documentation

2.3.2.1 `OpenNavigator` (**navigatorName** = `None`, **slotNo** = `0`)

Description

This function opens a Navigator and associates it with a reserved slot. The Navigator used must have been uploaded to the StormTest Repository in the Navigators area, either by using StormTest Developer Suite or using the UploadPublicNavigator method.

Parameters:

← **navigatorName String:** The name of the Navigator to open. If this is None then the default Navigator allocated for the DUT in the slot will be used.

← **slotNo Integer:** (optional) Specifies the slot for which the Navigator will be opened. If 0 then all reserved slots will have the specified Navigator. If None is specified as the navigatorName then each slot will have its default Navigator (if that has been configured)

Returns:

List of lists if slotNo = 0, (each list is [slot number, string]) otherwise a single string. The value of the string is the name of the Navigator for the slot, None if the default is requested and there is no default. If an explicit Navigator is specified and it does not exist then an exception of type StormTestExceptions is thrown.

Example:

```
>> OpenNavigator("mynavigator")          # slots 4 and 5 are reserved
>> [[4,"mynavigator"],[5,"mynavigator"]]
>> OpenNavigator(None)                   # slots 4 and 5 are reserved but slot 5 has no default
>> [[4,"nav1"],[5, None]]
```

Note:

The default Navigator is configured using the Admin Console web based tool.
If there is already a Navigator open on a slot, it will be closed first. This will occur before an attempt is made to open the requested Navigator.

2.3.2.2 `OpenNavigatorFile` **(navigatorFile, slotNo** = 0**)**

Description

This function opens a Navigator from a file and associates it with a reserved slot. The file can be any path accessible to the script running.

Parameters:

← **navigatorFile String:** The path and filename of the Navigator to open. If this is not an absolute path then the file will be assumed to be in the same folder as images configured via SetDirectories function.

← **slotNo Integer:** (optional) Specifies the slot for which the Navigator will be opened. If 0 then all reserved slots will have the specified Navigator.

Returns:

List of lists if slotNo = 0, (each list is [slot number, boolean]) otherwise a single boolean. The value of the boolean is true if the Navigator was opened, false if the Navigator was corrupt or missing something important. If the file does not exist then an exception of type StormTestExceptions is thrown.

Example:

```
>> OpenNavigatorFile("mynavigator.stnav_runtime")          # slots 4 and 5 are reserved
>> [[4,True],[5,True]]
```

Note:

If there is already a Navigator open on a slot, it will be closed first. This will occur before an attempt is made to open the requested Navigator.

2.3.2.3 `CloseNavigator` **(slotNo** = 0**)**

Description

This function closes a Navigator, freeing resources and breaks the association with its slot.

Parameters:

← **slotNo Integer:** (optional) Specifies the slot for which the Navigator will be closed. If 0 then all reserved slots will have their Navigator closed.

Returns:

List of lists if slotNo = 0, (each list is [slot number, boolean]) otherwise a single boolean. The value of the boolean is true if the Navigator was closed, false if there was no Navigator open on the slot.

Example:

```
>> CloseNavigator()              # slots 4 and 5 are reserved but only 5 has a Navigator
>> [[4,False],[5,True]]
```

Note:

The API is 'smart' in the sense that if the same Navigator is assigned to multiple slots and closed on just one slot, no resources are freed and the Navigator is still usable on other slots. It closes per slot not the underlying object.

2.3.2.4 `GetNavigatorScreens` (**slotNo** = 0)

Description

This function gets the list of screen in the Navigator associated with a slot.

Parameters:

← **slotNo Integer:** (optional) Specifies the slot for which the list of screens will be returned. If 0 then the list of screens for all reserved slots will be returned.

Returns:

List of lists if slotNo = 0, (each list is [slot number, List of strings]) otherwise a single list of strings. The list of strings will be the screen names defined in the Navigator. The order of the strings is not specified. It will be the same on repeated calls to this function but may change if the Navigator file is edited. If there is no Navigator for a slot, then an empty list is returned.

Example:

```
>> GetNavigatorScreens()              # slots 4 and 5 are reserved but only 5 has a Navigator
>> [[4,[]],[5,["screen1","epg_start","main menu" ...]]]
>> GetNavigatorScreens(5)
>> ["screen1","epg_start","main menu"]
```

2.3.2.5 `ValidateNavigatorScreen` (**screenName**, **slotNo** = 0)

Description

This function validates that the current DUT screen is equal to a specified screen in the Navigator.

Parameters:

← **screenName String:** The name of a screen in the Navigator.

← **slotNo Integer:** (optional) Specifies the slot for which the screen will be validated. If 0 then all reserved slots will be validated. It is not required that all slots have the same Navigator but it is required that the screen name is valid in all slots.

Returns:

List of lists if slotNo = 0, (each list is [slot number, status tuple]) otherwise a single status tuple. The status tuple is (bool (pass/fail), List Of Tests). The list of tests is the list of validation criteria for the screen in the format [ConditionName, bool (Pass/Fail), valueObject]. The valueObject is a dictionary of keys in format of parameter=>value. If a slot does not have a Navigator defined or the screen is not valid for the slot, then an exception of type StormTestExceptions is thrown.

Example:

```
>> ValidateNavigatorScreen("epg", 4)
>> (True,[["IconExist",True, {'ImageSimilarity': 98}],["Title", True, {'ObtainedString': "MENU"}]])
>> ValidateNavigatorScreen("main", 4)
>> (False,[["RightColor",False, {'DetectedColor': [55,90,9], 'Detected Flatness': 6, 'Detected Peak Error': 67.88}]])
>> ValidateNavigatorScreen("epg")
>> [[4,(True,[["IconExist",True, {'ImageSimilarity': 98}],["Title", True, {'ObtainedString': "MENU"}]])],[5,(False,[["I
```

2.3.2.6 `NavigateTo` (**destinationScreen**, **sourceScreen** = `None`, **progressCallback** = `None`, **slotNo** = `0`)

Description

This function navigates to a specific screen. Note that this validates the source screen as the first step if the source screen is specified.

Parameters:

← **destinationScreen String:** The name of a screen in the Navigator to navigate to.

← **sourceScreen String:** The name of a screen in the Navigator which the script believes is visible on the DUT. If this is None, then no assumptions are made and the API uses the defined Start Key sequence in the Navigator to bring the DUT to a known location and then continues navigation. If there are no such keys defined in the Navigator then an exception of type StormTestExceptions is thrown. The overall script execution time will be improved if the sourceScreen is specified (this may not be possible on the first call, of course).

← **progressCallback Function:** (optional) A function to be called on each navigation event. This allows the script to get progress about the navigation. The callback function is defined by the user script and gets paramters of slotNumber, screeenName, nextScreen, keysToPress. The intended purpose of the callback is as an aid to debugging (you get to see the path) and to allow the script to stop the navigation, if for instance, the 'wrong' screen is used as an intermediate screen.

← **slotNo Integer:** (optional) Specifies the slot for which the Navigator will navigate. Due to the complexity of navigation, if 0 is used then the API will execute **if and only if** the script has reserved 1 slot. It will throw an exception if 2 or more slots have been reserved.

Returns:

List of lists if slotNo = 0, (each list is [slot number, status tuple]) otherwise a single status tuple. The status tuple is (bool (pass/fail), ScreenName, List Of Tests). The screen name is the screen the Navigator expects to be visible (if all has gone well, this is the destinationScreen, otherwise it is an intermediate screen which failed to validate). The list of tests is the list of validation criteria for the screen in the format [ConditionName, bool (Pass/Fail), valueObject]. The valueObject is a dictionary of keys in format of parameter=>value. If a slot does not have a Navigator defined or the screen is not valid for the slot, then an exception of type StormTestExceptions is thrown.

Example 1:

```
>> NavigateTo("epg", 4)
>> (True, "epg",[["IconExist",True, {'ImageSimilarity': 98}],["Title", True, {'ObtainedString': "MENU"}]])
>> NavigateTo("main", 4)
>> (False,"setup",[["RightColor",False, {'DetectedColor': [55,90,9], 'Detected Flatness': 6, 'Detected Peak Error': 67.
>> NavigateTo("epg", "setup", 4)
>> (True, "epg",[["IconExist",True, {'ImageSimilarity': 98}],["Title", True, {'ObtainedString': "MENU"}]])
```

Example 2:

```
>> def NavSpy(slot, screen, nextScreen, keys):
>>    print "navigating on slot:",slot,"screen:",screen,"next screen:",nextScreen,"with IR keys:",keys

>> NavigateTo("epg", None, NavSpy, 4)
>> navigating on slot: 4 screen: None next screen: Main with IR keys: ['Back','Back','Back','Back','Back']
>> navigating on slot: 4 screen: Main next screen: Menu with IR keys: ['Menu']
>> navigating on slot: 4 screen: Menu next screen: Menu1 with IR keys: ['Down']
>> navigating on slot: 4 screen: Menu1 next screen: epg with IR keys: ['OK']
>> navigating on slot: 4 screen: epg next screen: None with IR keys: None
>> (True, "epg",[["IconExist",True, {'ImageSimilarity': 98}],["Title", True, {'ObtainedString': "MENU"}]])
```

2.3.2.7 `ValidateNavigator` (**sourceScreen** = `None`, **progressCallback** = `None`, **slotNo** = 0)

Description

This function validates the entire Navigator attempting to navigate to all screens and using all links specified (it implements a solution to the Chinese Postman Problem, Wikpedia has details) If the source screen is specified the validation assumes that the DUT is at that screen and validates from that point. If no source screen is specified, the the validation uses the Start Key sequence in the Navigator to bring the DUT to a known location and then starts validation. If there are no such keys defined in the Navigator then an exception is thrown. Due to the complexity of validation, the slot number must be explicitly given and it is not possible to run more than one validation in a script at the same time. An exception of type StormTestExceptions is thrown if no Navigator has been opened for the slot.

Parameters:

← **sourceScreen String:** The name of a screen in the Navigator which the script believes is visible on the DUT.

← **progressCallback Function:** (optional) A function to be called on each navigation event. This allows the script to get progress about the validation. The callback function is defined by the user script and gets paramters of slotNumber, screeenName, nextScreen, keysToPress. The intended purpose of the callback is as an aid to debugging (you get to see the path) and to allow the script to stop the validation, if for instance, it is taking too long.

← **slotNo Integer:** Specifies the slot for which the Navigator will validate.

Returns:

Status tuple. The status tuple is (bool (pass/fail), ScreenName, List Of Tests). The screen name is the screen the Navigator stopped validing on an error. If all has gone well, this is None. The list of tests is the list of validation criteria for the screen that failed in the format [ConditionName, bool (Pass/Fail), valueObject]. The valueObject is a dictionary of keys in format of parameter=>value.

Example:

```
>> def Spy(slot, screen, nextScreen, keys):
>>     print "navigating on slot:",slot,"screen:",screen,"next screen:",nextScreen,"with IR keys:",keys
>> ValidateNavigator(None, Spy, 4)
>> navigating on slot: 4 screen: None next screen: Main with IR keys: ['Back','Back','Back','Back','Back']
>> navigating on slot: 4 screen: Main next screen: Menu with IR keys: ['Menu']
>> navigating on slot: 4 screen: Menu next screen: Menu1 with IR keys: ['Down']
>> navigating on slot: 4 screen: Menu1 next screen: epg with IR keys: ['OK']
>> # ... many lines here as the navigation map is traversed.
>> (True, None, None)

>> ValidateNavigator(None, Spy, 5)
>> navigating on slot: 5 screen: None next screen: Main with IR keys: ['Back','Back','Back','Back','Back']
>> navigating on slot: 5 screen: Main next screen: Menu with IR keys: ['Menu']
>> (False, "Menu", [["RightColor",False, {'DetectedColor': [55,90,9], 'Detected Flatness': 6, 'Detected Peak Error': 67
```

2.3.2.8 `StopNavigator` (**slotNo** = 0)

Description

This function stops navigation. This cannot be called on the thread that is navigating (because the navigation/validation is synchronous). So either the script must be multi-threaded or the Stop has to be called within the callback on some condition (for example, too much time has elapsed).

Parameters:

← **slotNo Integer:** (optional) Specifies the slot for which the Navigator will be stopped. If 0 then all reserved slots will have their Navigator stopped.

Returns:

List of lists if slotNo = 0, (each list is [slot number, boolean]) otherwise a single boolean. The value of the boolean is true if the Navigator was stopped, false if there was no Navigator open on the slot or the Navigator was not running. Note that if you just call StopNavigator on a reserved slot then you cannot determine whether this is due to no Navigator or it has completed the navigation before you called StopNavigator.

Example:

```
>> StopNavigator()
>> [[4,False]]
```

2.3.2.9 `GetValidNavigators` **(slotNo** = 0**)**

Description

This function gets a list of valid Navigators for the DUT in a specific slot. The Admin Console must have been previously used to configure a DUT in the slot and associate Navigators with the DUT model of the DUT instance. Any of these Navigators should work the the DUT in the specified slot and can be used by calling OpenNavigator()

Parameters:

← **slotNo Integer:** (optional) Specifies the slot for which the list of Navigators will be returned. If 0 then all reserved slots will have their Navigators returned.

Returns:

List of lists if slotNo = 0, (each list is [slot number, List of strings]) otherwise a single list of strings. The list of strings is all the Navigators which are valid for the slot with the first in the list being the 'default' preferred Navigator. Typically there is only 1 value. If no Navigators are valid, then an empty list is returned.

Example:

```
>> GetValidNavigators()                # slots 4,5 and 6 are reserved.
>> [[4,["Nav2", "Nav88"]], [5, []], [6,["nav1"]]]
```

2.3.2.10 `GetCurrentNavigator` (**slotNo** = 0)

Description

This function gets the current Navigator for the slot.

Parameters:

← **slotNo Integer:** (optional) Specifies the slot for which the Navigator is wanted. If 0 then all reserved slots will have their Navigators returned.

Returns:

List of lists if slotNo = 0, (each list is [slot number, strings]) otherwise a single string. The string is the Navigator name (if opened with a name) or a filename if opened from a file. If no Navigator is open on a slot, then None is returned.

Example:

```
>> GetCurrentNavigator()              # slots 4,5 and 6 are reserved.
>> [[4,"Nav2"], [5, None], [6,"some_navigation.stnav_runtime"]]
```

2.3.2.11 `ConvertNavigatorToScreenDef` (**screenToConvert**, **slotNo** = 0)

Description

This function converts 1 or more screens of a navigator to a ScreenDefinition object. The slotNo must have an associated Navigator.

Parameters:

← **screenToConvert String:** a string (or a list of strings) specifying the screen to convert. Each screen must be a valid name in the navigator. If an empty list is specified then all navigator screens are returned. The order is the same as returned in GetNavigatorScreens.

← **slotNo Integer:** (optional) The slot number to identify an open navigator. The slot must have a valid navigator opened on it. Can be 0 to mean all reserved slots.

Returns:

List of Lists if slot = 0, (each list is [slot number, result]) otherwise the result. The result is a single ScreenDefinition if a single string is supplied for screenToConvert but if a list is supplied for screenToConvert then the result is also a list of ScreenDefinitions

Note:

This API converts a navigator screen so that it can be used in an API call such as WaitScreenDefMatch. The Navigator evaluation of color regions would round the flatness and peak error to the nearest integer before checking against the thresholds (the GUI uses integers). Python uses floating point so the conversion will adjust the color and peak error by 0.5 percentage points to allow a test that passes in Navigator using integers to pass in Python when using floating point. This change is intentional and not a bug.

Example:

```
>> ConvertNavigatorToScreenDef("main", 5)              # slot 5 is reserved with an open navigator
>> <ScreenDefinition instance at 0x02800070>
>> ConvertNavigatorToScreenDef([], 5)              # slot 5 is reserved with an open navigator
>> [<ScreenDefinition instance at 0x0280070>,<ScreenDefinition instance at 0x0280FE0>, ...]
```

2.3.2.12 `UploadPublicNavigator` (localFile, navigatorName)

Description

This function uploads a navigator from the local machine to the public navigator area. It will overwrite any existing public navigator of the same name. You must upload the .stnav_runtime for a navigator to be used as a public navigator and you may also upload the stnav_edit file as a useful backup (but the .stnav_edit file is usually quite large compared to the .stnav_runtime).

Parameters:

← **localFile String** : The path to a navigator file (either .stnav_runtime or .stnav_edit). If a relative path is used then it is relative to the directory set by SetProjectDir().

← **navigatorName String** : The name of the public navigator to upload. It will be created if it does not exist. To associate a newly created navigator with one or more models, use the AdminConsole.

Returns:

None

Example:

```
>> # create/overwrite a navigator called 'Box5' from a local file - the file saved will be 'Box5.stnav_runtime'
>> UploadPublicNavigator("c:\\projects\\myproject\\mynavigator.stnav_runtime", "Box5")
```

This function is available in release 3.3.3

2.3.2.13 `DownloadPublicNavigator` **(navigatorName, localFile)**

Description

This function downloads a public navigator to the local machine. It will download the specified runtime or edit file.

Parameters:

← **navigatorName String** : The public navigator to download. It may have the suffix .stnav_-runtime or .stnav_edit but if omitted, then .stnav_runtime is assumed.

← **localFile String** : The local file to write. If the extension (.stnav_runtime or .stnav_edit) is omitted then the file saved will have the same extension as the navigator file. An exception will be raised if the local file extension is invalid (either totally invalid such as .txt or trying to save a .stnav_runtime navigator to a .stnav_edit local file). If a relative path is used then it is relative to the directory set by SetProjectDir().

Returns:

None

Example:

```
>> # download the runtime and edit files from a navigator called 'Box5'
>> DownloadPublicNavigator("Box5", "mynav")  # saves to mynav.stnav_runtime in current directory
>> DownloadPublicNavigator("Box5.stnav_edit", "mynav") # saves to mynav.stnav_edit in current directory
```

This function is available in release 3.3.3

2.3.2.14 `ListPublicNavigators` ()

Description

This function lists all available public navigators.

Parameters:

None

Returns:

List : List of strings of all public navigators. The full extension will be listed. The list is not sorted in any way. It is not guaranteed to be repeatable between successive calls to this function.

Example:

```
>> ListPublicNavigators()
>> ["Box5.stnav_runtime", "Box5.stnav_edit", "Nav6.stnav_runtime", ...]
```

This function is available in release 3.3.3

2.4 Power State API

These functions control the power state of the set top box.

Functions

- PowerOnSTB
- PowerOffSTB
- IsSlotPowerOn

2.4.1 Detailed Description

These functions control the power state of the set top box.

2.4.2 Function Documentation

2.4.2.1 `PowerOnSTB` (**timeDelay** = 30, **slotNo** = 0)

Description

This function powers on the DUT.

Parameters:

← **timeDelay Float:** (optional) The delay in seconds after the DUT is powered up and before the function returns, see note[1] .

← **slotNo Integer:** (optional) Powers on one specific slot instead of all reserved slots.

Returns:

None

Example:

```
>>PowerOnSTB()      # Power on, waiting the default 30 seconds
or
>>PowerOnSTB(10,3)  # Power on waiting for 10 seconds the DUT in slot 3
```

Note:

1. This delay is used to avoid the damage to the DUT(s) caused by immediate power-off again.

If the *timeDelay* parameter is left out the default delay is 30 seconds.
If the *slotNo* parameter is omitted, the default is for all reserved slots.

2.4.2.2 `PowerOffSTB` (**timeDelay** = 30, **slotNo** = 0)

Description

This function powers off the DUT.

Parameters:

← **timeDelay Float:** (optional) The delay in seconds after the DUT is powered down and before the function returns, see note[1] .

← **slotNo Integer:** (optional) Powers off one specific slot instead of all reserved slots.

Returns:

None

Example:

```
>>PowerOffSTB()     # Power off using the default 30 sec
or
>>PowerOffSTB(10,3) # power off and delay by 10 seconds the DUT in slot 3
```

Note:

1. This delay is used to avoid the damage to the DUT(s) caused by immediate power-on again.

If the *timeDelay* parameter is left out the default is 30 seconds.
If the *slotNo* parameter is omitted, the default is for all reserved slots.

2.4.2.3 `IsSlotPowerOn` (**slotNo** = 0)

Description

This function returns whether a slot is powered on. The CheckSlots() function has a limitation in that if a slot is reserved, you cannot find out if the slot is powered on or off. This function resolves that limitation. This function is supported in release 3.3.4.

Parameters:

← **slotNo Integer:** (optional) Specifies which slot to return data for. Use 0 for all reserved slots. return *Bool* : The value is True or False for slot powered on or off. If the server has no RPS installed then None is returned. When the slotNo is 0 the return is a list as [[slot, status], [slot, status]] with one item in the list for each slot and the status of True, False or None.

Example:

```
>>IsSlotPowerOn(5)
>>True
>>IsSlotPowerOn()
>>[[1, True], [2, False]]
```

2.5 Serial Port API

These functions are to access serial data on each StormTest slot.

Functions

- AddObserve
- RemoveObserve
- GetObserveLog
- GetSerialLog
- StartSerialLog
- StopSerialLog
- SetExtParser
- ChangeSerialParameters
- SendCommandViaSerial
- SetRawDataReceiver
- SendRawData
- CommentToLog

2.5.1 Detailed Description

These functions are to access serial data on each StormTest slot.

In order to access the serial data, the script should pass the serial port parameters to the ReserveSlot() function. This will enable the serial data stream.

Note that it is possible for the user to monitor the serial stream while a test is running by using a standard telnet program to telnet to the StormTest server on a port that is a slot based increment off the base port. The base port is configured using the web based Admin Console.

For example, if the base port is 12001 (the default), then the serial stream for slot 5 is on port 12005. The data from the serial port is inserted into the main html log file and for compatibility in a text file with a auto generated name - the exact filename can be found by a script by calling GetSerialLog()

2.5.2 Function Documentation

2.5.2.1 `AddObserve` (**observingString**, **callbackFunction** = `None`, **slotNo** = `0`, **waitForEOL** = `False`)

Description

This function allows the user to observe the serial output for a specific text string. If that string is seen in the output, then StormTest will log the entire line containing the string to a new log file. This new log file will have the same name as the main log file, but with an **_mon** suffix.
A callback can also be passed to StormTest and this callback will be executed every time the string is observed.
Please note that StormTest assumes that the output from the DUT is encoded using the latin1 (ISO-8859-1) character set. The observe string that is passed into this function should also be encoded with latin1. If this is not the case, the function will not work as expected.

Parameters:

← **observingString String:** The string, which the serial port output will be checked for. This string should use latin1 (ISO-8859-1) character encoding.

← **callbackFunction Function** (optional) The callback function that should be called in case the string occurs.

← **slotNo Integer:** (optional) Used to specify the slot number to start observing on.

← **waitForEOL Bool:** (optional) If this parameter is True, in case of a matching string, StormTest will wait for an end-of-line character before writing the received text to the **_mon** log file. Default is False.

Returns:

None

Example:

```
>>AddObserve("ERR") # The serial output will be monitored for the string ERR
```

Note:

If *callbackFunction* is not specified the default callback handler will be used (logging to a file).
The callback function is defined as follows:

callbackFunction(slotNo, subSlot, a_key, a_string).

Parameters:

← **slotNo Integer:** The slot number the string was found on.

← **subSlot Integer:** The sub slot number the string was foun on.

← **a_key String:** The substring in the debug line as defined via the *observingString*.

← **a_string String:** The debug line as a string from the serial port.

Note:

the subSlot parameter is needed for HS64 systems. However scripts written with 3 parameters in the callback will work perfectly. StormTest detects the signature and adjusts accordingly. HS64 scripts can also use 3 parameters but won't know which sub slot has found the observed string.

2.5.2.2 `RemoveObserve` (**observingString**, **slotNo** = 0)

Description

This function is used to stop observing the string set in AddObserve.

Parameters:

← **observingString String:** The string to stop observing. This string should use latin1 (ISO-8859-1) character encoding.

← **slotNo Integer:** (optional) Used to specify the slot number to call RemoveObserve on.

Returns:

None

Example:

```
>>RemoveObserve("ERR") # Monitoring for string ERR will be stopped
```

2.5.2.3 `GetObserveLog` ()

Description

A test script can monitor the serial output for a number of strings by calling the AddObserve() function. StormTest will then record any instances of these strings in a separate log file with an **_mon** suffix. This function is then used to check if any of the strings that the test script is monitoring for have been found by StormTest by looking at the size of the **_mon** log file.

Parameters:

None

Returns:

see Debug Log Information

Example:

```
>>print GetObserveLog()
>>[(11, 0L, 'C:\\test\\Prefix_11_Timestamp_mon.log'),     # 0 bytes in the monitor log
                                                          # for slot 11

>> (12, 25L, 'C:\\test\\Prefix_12_Timestamp_mon.log')]    # 25 bytes in the monitor log
                                                          # for slot 12
```

2.5.2.4 `GetSerialLog` ()

Description

This function is used to check logging data from the serial port.

Parameters:

None

Returns:

see Debug Log Information

Example:

```
>>print GetSerialLog()
>>[(11, 0L, 'C:\\test\\Prefix_11_Timestamp.log'),     # 0 bytes in the monitor log
                                                      # for slot 11

>> (12, 25L, 'C:\\test\\Prefix_12_Timestamp.log')]    # 25 bytes in the monitor log
                                                      # for slot 12
```

2.5.2.5 `StartSerialLog` (**serialParam, slotNo** = 0)

Description

This function is used to begin logging data from the serial port. Note that this function should **only** be used if serial port logging has previously been stopped by a call to the StopSerialLog() function call. All subsequent serial data will be logged to a new file

Parameters:

← **serialParam Serial** Port Setup Parameters:**Parameters** to use for communicating with the serial port on the DUT (see Serial Port Setup Parameters).

← **slotNo Integer:** (optional) The slot number to start monitoring for serial data.

Returns:

None

Example:

```
>>StartSerialLog([19200,8,"none",1,"none"],3)   # Start logging on slot number 3
```

Note:

If the *slotNo* parameter is omitted the default is to start logging for all slots reserved.

2.5.2.6 `StopSerialLog` **(slotNo** = 0**)**

Description

This function is used to stop logging data from the serial port.

Parameters:

← **slotNo Integer:** (optional) The slot number to stop monitoring for serial data.

Returns:

None

Example:

```
>>StopSerialLog()
```

Note:

If the *slotNo* parameter is omitted the default is to stop logging for all slots reserved.

2.5.2.7 `SetExtParser` (**callbackFunction, slotNo** = 0)

Description

This function allows a user to develop an external parser for the serial data. The callback function passed to this API will receive the raw serial data from the DUT in each reserved slot. This should be used when the data is line based text data as complete lines are sent to the callback. For parsing binary data or data without line feed characters, SetRawDataReceiver should be used.

Parameters:

← **callbackFunction Function:** The definition of the function can be found below.

← **slotNo Integer:** (optional) Only installs the external parser for serial data from one slot.

Returns:

None

Example:

```
>>def callbackFunction(slot, data):
>>  print "Received",data,"on slot",slot
>>
>>SetExtParser(callbackFunction)
```

Note:

If the *slotNo* parameter is omitted the default is to install the parser for all slots reserved. The callback function is defined as follows:

callbackFunction(slotNo, subSlot, data)

Parameters:

← **slotNo Integer:** The slot number the data is coming from.

← **subSlot Integer:** The sub slot number the data is coming from.

← **data String:** Data received from the DUT in that slot.

Note:

the subSlot parameter is needed for HS64 systems. However scripts written with 2 parameters in the callback will work perfectly. StormTest detects the signature and adjusts accordingly. HS64 scripts can also use 2 parameters but won't know which sub slot has found the observed string.

2.5.2.8 `ChangeSerialParameters` (**serialParam**, **slotNo** = 0)

Description

This function is used to change the parameters used to communicate with the serial port on the DUT. Serial port logging must have been enabled previously. Subsequent serial port data is logged into the same file as before.

Parameters:

← **serialParam Serial Port Setup Parameters:** to use for communicating with the serial port on the DUT (see Serial Port Setup Parameters).

← **slotNo Integer:** (optional) The slot number to change the serial parameters on.

Returns:

None

Example:

```
>>ChangeSerialParameters([19200,8,"none",1,"none"],3)
# Change serial parameters on slot number 3
```

Note:

If the *slotNo* parameter is omitted the default is to change serial parameters for all slots reserved.

2.5.2.9 `SendCommandViaSerial` (**commandString**, **expectString**, **timeDelay**, **slotNo** = `0`, **waitForEOL** = `False`)

Description

This function can be used to interact with a DUT via its serial port.
If a DUT is running an application that takes input via the serial port, then this function can be used to send commands to that application. The function sends the *commandString* to the serial port and will wait up to *timeDelay* seconds for the string *expectString* to appear on the serial output. An example of how this might work is that the user could send "4" to run test number 4 and then wait up to 60 seconds for "TEST PASSED" to appear on the serial output. The return value from the function will contain the line from the serial log that the 'expect string' was seen in and can then be parsed further by the test script if necessary.

Parameters:

← **commandString String:** The command string. This string should use latin1 (ISO-8859-1) character encoding. StormTest will append a line feed and a carriage return ('\r\n') to the end of this string.

← **expectString String:** The expected string. This string should use latin1 (ISO-8859-1) character encoding.

← **timeDelay Float:** Time in seconds.

← **slotNo Integer:** (optional) Used to specify the slot number to send the serial command on.

← **waitForEOL Bool:** (optional) If this parameter is True, at a matching string, wait for a linefeed character ('\n') before returning from the function. Default is False.

Returns:

Array see Returned Serial Port Data

Example:

```
>>ret=SendCommandViaSerial("4","TEST PASSED",60)
>>print ret
>>[(1, True, 'TEST PASSED', 'CMD> 4\nLine of text containing TEST PASSED and some
    text until a newline character\n'),
>> (2, False, 'TEST PASSED', 'CMD> 4\n')]                    # slot 1 success, slot 2 failure
```

2.5.2.10 `SetRawDataReceiver` (**callbackFunction, slotNo** = 0)

Description

This function allows a user to develop an external parser or a binary data receiver for the serial data. The callback function passed to this API will receive the raw serial data from the DUT in each reserved slot. If you know the data is text based with line feeds you can use either SetExtParser or SetRawDataReceiver

Parameters:

← **callbackFunction Function:** The definition of the function can be found below.

← **slotNo Integer:** (optional) Only installs the external parser for serial data from one slot.

Returns:

None

Example:

```
>>def callbackFunction(slot, data):
>>  print "Received",data,"on slot",slot
>>
>>SetRawDataReceiver(callbackFunction)
```

Note:

If the *slotNo* parameter is omitted the default is to install the parser for all slots reserved. The callback function is defined as follows:

callbackFunction(slotNo, data)

Parameters:

← **slotNo Integer:** The slot number the data is coming from.

← **subSlot Integer:** The sub slot number the data is coming form.

← **data String:** Data received from the DUT in that slot. The raw serial data is not decoded and it is up to a user.

Note:

the subSlot parameter is needed for HS64 systems. However scripts written with 2 parameters in the callback will work perfectly. StormTest detects the signature and adjusts accordingly. HS64 scripts can also use 2 parameters but won't know which sub slot has found the observed string.

2.5.2.11 `SendRawData` (**hexString, slotNo** = 0)

Description

This function can send binary raw data to a DUT via its serial port. While sending data, other serial port related APIs will be blocked.

Parameters:

← **hexString String:** The data string. This string is not be encoded and it is up to a user.

← **slotNo Integer:** (optional) Used to specify the slot number to send the serial command on.

Returns:

Array

Example:

```
>>ret=SendRawData(chr(0xff)+chr(0xfa))
>>print ret
>>[(1, True, 2), (2, False, 0)]     # slot 1 success and sent 2 bytes, slot 2 failure
```

2.5.2.12 `CommentToLog` (**commentString**, **slotNo** = 0)

Description

This function is used to write a string to the serial log. This allows the user to write a test script that will automatically mark the point in the serial log that certain events in the test script occurred.

Parameters:

← **commentString String:** A string that will be written to the serial log. This string should use latin1 (ISO-8859-1) character encoding.

← **slotNo Integer:** (optional) If used, the comment string will only be written to the serial log of the slot specified.

Returns:

None

Example:

```
>>CommentToLog("Event A is about to happen")
# Write the string "Event A is about to happen" to the serial log file of all
# reserved slots
```

2.6 Audio and Video API

These are the audio and video capture and analysis functions.

Enums

- enum OverWriteAction

  OverWriteAction Enum.

- enum CompareAlgorithm
- enum SaveScreenCap
- enum HDMIColorSpace
- enum VideoStandard
- enum VideoSignalState

  VideoSignalState Enum.

- enum CoordConverter

Functions

- SetVideoCpuUsage
- ShowVideo
- CloseVideo
- HideVideo
- DetectMotionEx
- CaptureImageEx
- CompareImageEx
- CompareIconEx
- WaitImageMatch
- WaitImageNoMatch
- WaitIconMatch
- WaitIconNoMatch
- CompareColorEx
- WaitColorMatch
- WaitColorNoMatch
- SetVideoOffset
- GetVideoOffset
- CalibrateVideoOffset
- StartVideoLog
- StopVideoLog
- EnableRollingRecord
- DisableRollingRecord
- SaveRecording
- BookmarkVideoLog
- GetAudioLevel
- WaitAudioPresence
- WaitAudioAbsence
- ResyncAudio

- IsAudioResyncNeeded
- SetResolution
- GetResolution
- GetRawInputInfo
- SetMaxBitRate
- GetMaxBitRate
- SetFrameRate
- GetFrameRate
- SetVideoStandard
- ResetDefaultParameters
- RegisterVideoLossCallback
- SetDesignToWorldTransform
- GetDesignToWorldTransform
- SetWorldToDesignTransform
- GetWorldToDesignTransform

2.6.1 Detailed Description

These are the audio and video capture and analysis functions.

To use all these functions, the video stream to the client must be enabled (it is enabled by default). The video stream is enabled/disabled by a flag passed to the ReserveSlot() function.

2.6.2 Function Documentation

2.6.2.1 `SetVideoCpuUsage` (**cpuUsage**, **slotNo** = 0)

Description

This function controls the level of CPU used by the client to decode and display the video streamed to the StormTest client. CPU Usage can either be high (1) or low (0).
Note that changing this setting to low has no impact on the ability of a StormTest script to run. Image comparisons etc will still operate as normal. It's only effect is to make the video playback look less smooth to the human eye.
If low CPU usage is selected, the CPU usage per test is reduced by approximately 50%. For this reason, it is recommended that low CPU usage is selected when running tests on a PC where you wish to run multiple tests at once (ie if you are running your tests on the StormTest server itself).
Note that this is a client side function only - it has no effect on the contents of the video stream being sent from the server and stored to disk by the StartVideoLog() function.

Parameters:

← **cpuUsage Integer:** Setting to 1 enables high CPU usage. Setting to 0 enables low CPU usage.

← **slotNo Integer:** (optional) Used to specify the slot number to call SetVideoCpuUsage() on.

Returns:

None

Example:

```
>>SetVideoCpuUsage(1)    # Use as much CPU as is necessary to decode the video stream.
```

Note:

The default value if the test does not call SetVideoCpuUsage() is **high CPU usage / high quality** video decoding.

2.6.2.2 `ShowVideo` **(slotNo** = 0**)**

Description

This function opens a video stream and displays the video in a monitoring window This function should be used INSTEAD of OpenVideoWindow(), eliminating the need for a callback video handler handler

Parameters:

← **slotNo Integer:** (optional) Used to specify the slot number to call ShowVideo() on.

Returns:

None

Example:

```
>>ShowVideo()
```

2.6.2.3 `CloseVideo` **(slotNo** = 0**)**

Description

This function stops streaming video from the Storm Test server which was opened with the ShowVideo() API call and closes the associated video window. This function can only be used on video streams opened with ShowVideo() and INSTEAD of CloseVideoWindow()

Parameters:

← **slotNo Integer:** (optional) Used to specify the slot number to call CloseVideo() on.

Returns:

None

Example:

```
>>CloseVideo()
```

2.6.2.4 `HideVideo` **(slotNumber** = 0**)**

Description

This function hides an open video window opened with the ShowVideo() API call. The video stream remains open allowing video analysis This function can only be used on video streams opened with the ShowVideo() API call

Parameters:

← **slotNumber Integer:** (optional) Used to specify the slot number to call HideVideo() on.

Returns:

None

Example:

```
>>HideVideo()
```

2.6.2.5 `DetectMotionEx` (**rect**, **timeout**, **percent** = 5, **slotNo** = 0)

Description

This function performs detect motion on a video stream

Parameters:

← **rect Tuple:** The rect is a crop area. In Release 3.1 and later this may be a NamedRegion.

← **timeout Float:** Number of seconds to wait for motion to be detected.

← **percent Integer:** The limit for pass/fail. Default 5, so if there is more than 5% difference between frames, the return value is true.

← **slotNo Integer:** (optional) Used to specify the slot number to call DetectMotionEx on.

Returns:

List: A list of lists consisting of slot number,true/false,last image captured as StormTestImageObject ,actual percentage, number of captures performed, total time taken. A single list if the exact slot number is given

Example:

```
# look for motion in top right corner of the screen (an area 104 x 75 pixels starting at 604 from left and 0 from t
# wait for 5 seconds.
ret = DetectMotionEx((600,0,104,75),5,percent=5,slotNo = 8)
print ret
>>>[8, False, <StormTestImageObject instance>,0.36330969109354783, 42, 5.0009999999999994]
ret = DetectMotionEx((600,0,104,75),5,percent=5)
print ret
>>>[[8, False, <StormTestImageObject instance>,0.36330969109354783, 42, 5.0009999999999994]]
```

2.6.2.6 `CaptureImageEx` (**rect**, **filename**, **jpegQuality** = 80, **overwriteAction** = `OverWriteAction.NewName`, **slotNo** = 0)

Description

This function captures an image, crops and saves it if those details are specified

Parameters:

← **rect Tuple:** The rect is a crop area. In Release 3.1 and later this may be a NamedRegion.

← **filename String:** The filename specifies where to save the file and allows d t to substitute the date/time. Extension of filename determines the type save, if none, then .png. User can specify Python 'None' for filename and then the file is not saved, just captured to memory. In Release 3.1 and later this may be a NamedImage.

← **jpegQuality Integer:** The jpegQuality specifies the quality of a jpeg image being saved. Default 80, range is 1 - 100.

← **overwriteAction OverWriteAction:** overwriteAction allows user to control behavior on file conflict (overwrite, error, newname). Default OverWriteAction.NewName.

← **slotNo Integer:** (optional) Used to specify the slot number to call CompareImage on.

Returns:

List: If slotNo is 0 or ommitted: A list of lists consisting of slot number,true/false,StormTestImageObject ,full path of saved file, if any. If slotNo is an explicit number then just a simple list of slot number, true/false etc is returned.

Note:

The true/false refers to whether an image was captured or not, true meaning it was captured without error.
If ReserveSlot is called with VideoFlag False, the capture is performed on the server and the saved file is always JPEG format. This has always been the case with StormTest and is not new behavior. It may be fixed in a future release so you should use .JPG if videoFlag was false.

Example:

```
ret = CaptureImageEx (None, "file.bmp")
print ret
  >>>[[8, True, <StormTestImageObject instance>, u'file.jpg']]
```

Note:

This function ignores the effect of SetVideoOffset when the connected server is an HD server. It accounts for the offset when the connected server is Standard Definition with the captured image being offset as per SetVideoOffset

2.6.2.7 `CompareImageEx` (**refImage**, **testImage**, **percent** = `98`, **includedAreas** = `None`, **excludedAreas** = `None`, **algorithm** = `CompareAlgorithm.Compare1`)

Description

This function performs a single compare

Parameters:

← **refImage String** or StormTestImageObject : file or image from CaptureImageEx(). In Release 3.1 and later this may be a NamedImage.

← **testImage String** or StormTestImageObject : file or image from CaptureImageEx(). In Release 3.1 and later this may be a NamedImage.

← **percent Integer:** The limit for pass/fail. Default 98. When the detected similarity is above this value, the comparison succeeds.

← **includedAreas List:** Included areas for comparision, currently only one can be specified. This is either a simple rectangle OR a list of rectangles (each a list of 4) and ONLY 1 element currently supported. In Release 3.1 and later the rectangel may be a NamedRegion.

← **excludedAreas List:** Excluded areas for comparision, NOT currently supported.

← **algorithm CompareAlgorithm:** Comparison algorithm to use, only one allowed

Returns:

List: true/false, actual percentage match

Example:

```
ret = CompareImageEx(refImage, testImage, percent)
print ret
>>>[False, 45.076345304474223]
# using a crop area
ret = CompareImageEx(refImage, testImage, percent, [0,0,50,50])
# using a list of crop areas
ret = CompareImageEx(refImage, testImage, percent, [[0,0,50,50]])
```

Note:

If a named image or rectangle is used then only the general 'all slot' path will be searched because this function does not take a slot number.

2.6.2.8 `CompareIconEx` (**refIcon, testImage, percent** = 98, **location** = `None`, **algorithm** = `CompareAlgorithm.Compare1`)

Description

This function performs a single compare on an icon at a specified location

Parameters:

← **refIcon String** or StormTestImageObject : file or image from CaptureImageEx(). In Release 3.1 and later this may be a NamedImage.

← **testImage String** or StormTestImageObject : file or image from CaptureImageEx(). In Release 3.1 and later this may be a NamedImage.

← **percent Integer:** The limit for pass/fail. Default 98. When the detected similarity is above this value, the comparison succeeds.

← **location Tuple:** A tuple (x,y) where icon is within the testImage. In Release 3.1 and later this may be a NamedRegion (but only the left and top parts are used)

← **algorithm CompareAlgorithm:** Comparison algorithm to use, only one allowed

Returns:

List: true/false, actual percentage match

Example:

```
ret = CompareIconEx(refIcon, testImage , location=(100,100))
print ret
>>>[True, 99.076345304474223]
```

Note:

If a named image or rectangle is used then only the general 'all slot' path will be searched because this function does not take a slot number.

2.6.2.9 `WaitImageMatch` **(** **refImage**, **percent** = `98`, **includedAreas** = `None`, **excludedAreas** = `None`, **algorithm** = `CompareAlgorithm.Compare1`, **timeToWait** = `5`, **waitGap** = `1`, **slotNo** = `0`**)**

Description

Waits for image compare to pass.

Parameters:

← **refImage String** or StormTestImageObject : file or image from CaptureImageEx(). In Release 3.1 and later this may be a NamedImage.

← **percent Integer:** The limit for pass/fail. Default 98. When the detected similarity is above this value, the comparison succeeds.

← **includedAreas List:** Included areas for comparision, currently only one can be specified. See CompareImageEx for details on format and usage. In Release 3.1 and later the rectangle may be a NamedRegion.

← **excludedAreas List:** Excluded areas for comparision, NOT currently supported.

← **algorithm CompareAlgorithm:** Comparison algorithm to use, only one allowed

← **timeToWait Integer:** Number of Seconds to wait for a pass

← **waitGap Float:** Interval between subsequent comparisons

← **slotNo Integer:** (optional) Used to specify the slot number to call WaitImageMatch on.

Returns:

Array: An array of tuples consisting of slot number,true/false,last image captured as StormTestImageObject ,actual percentage, number of captures performed, total time taken

Example:

```
ret = WaitImageMatch(testImage, percent, includedAreas, excludedAreas, algorithm, timeToWait, waitGap)
print ret
>>>[[8, False, <StormTestImageObject instance>,3.7219811921117305, 5, 4.5529999732971191]]
```

Note:

This function ignores the effect of SetVideoOffset when the connected server is an HD server. It accounts for the offset when the connected server is Standard Definition.

2.6.2.10 WaitImageNoMatch (**refImage**, **percent** = 98, **includedAreas** = None, **excludedAreas** = None, **algorithm** = CompareAlgorithm.Compare1, **timeToWait** = 5, **waitGap** = 1, **slotNo** = 0)

Description

Waits for image compare to fail.

Parameters:

← **refImage String** or StormTestImageObject : file or image from CaptureImageEx(). In Release 3.1 and later this may be a NamedImage.

← **percent Integer:** The limit for pass/fail. Default 98, so if the returned percentage match is below this it is a passing result

← **includedAreas List:** Included areas for comparision, currently only one can be specified. In Release 3.1 and later the recttangle may be a NamedRegion. See CompareImageEx for details on format and usage.

← **excludedAreas List:** Excluded areas for comparision, NOT currently supported.

← **algorithm CompareAlgorithm:** Comparison algorithm to use, only one allowed

← **timeToWait Integer:** Number of Seconds to wait for a pass

← **waitGap Float:** Interval between subsequent comparisons

← **slotNo Integer:** (optional) Used to specify the slot number to call WaitImageNoMatch on.

Returns:

Array: An array of tuples consisting of slot number,true/false,last image captured as StormTestImageObject ,actual percentage, number of captures performed, total time taken

Example:

```
ret = WaitImageNoMatch(testImage, percent, includedAreas, excludedAreas, algorithm, timeToWait, waitGap)
print ret
>>>[[8, False, <StormTestImageObject instance>,99.40149751270468, 10, 10.073000192642212]]
```

Note:

This function ignores the effect of SetVideoOffset when the connected server is an HD server. It accounts for the offset when the connected server is Standard Definition.

2.6.2.11 `WaitIconMatch` (**refIcon**, **percent** = 98, **location** = None, **algorithm** = `CompareAlgorithm.Compare1`, **timeToWait** = 5, **waitGap** = 1, **slotNo** = 0)

Description

Waits for icon compare of a ref icon against a live stream to pass

Parameters:

← **refIcon String** or StormTestImageObject : file or image from CaptureImageEx(). In Release 3.1 and later this may be a NamedImage.

← **percent Integer:** The limit for pass/fail. Default 98. When the detected similarity is above this value, the comparison succeeds.

← **location Tuple:** A tuple (x,y) where icon is within screen area being compared. In Release 3.1 and later this may be a NamedRegion but only the left and top values are used.

← **algorithm CompareAlgorithm:** Comparison algorithm to use, only one allowed

← **timeToWait Integer:** Number of Seconds to wait for a pass

← **waitGap Float:** Interval between subsequent comparisons

← **slotNo Integer:** (optional) Used to specify the slot number to call WaitImageNoMatch on.

Returns:

Array: An array of tuples consisting of slot number,true/false,last image captured as StormTestImageObject ,actual percentage, number of captures performed, total time taken

Example:

```
ret = WaitIconMatch(refIcon, percent, location, algorithm, timeToWait, waitGap)
print ret
>>>[[8, True, <StormTestImageObject instance>, 99.794884604363105, 1, 0.1119999885559082]]
```

Note:

This function ignores the effect of SetVideoOffset when the connected server is an HD server. It accounts for the offset when the connected server is Standard Definition.

2.6.2.12 `WaitIconNoMatch` (**refIcon**, **percent** = `98`, **location** = `None`, **algorithm** = `CompareAlgorithm.Compare1`, **timeToWait** = `5`, **waitGap** = `1`, **slotNo** = `0`)

Description

Waits for icon compare of a ref icon against a live stream to fail.

Parameters:

- ← **refIcon String** or StormTestImageObject : file or image from CaptureImageEx(). In Release 3.1 and later this may be a NamedImage.
- ← **percent Integer:** The limit for pass/fail. Default 98, so if the returned percentage match is below this it is a passing result
- ← **location Tuple:** A tuple (x,y) where icon is within screen area being compared. In Release 3.1 and later this may be a NamedRegion but only the left and top values are used.
- ← **algorithm CompareAlgorithm:** Comparison algorithm to use, only one allowed
- ← **timeToWait Integer:** Number of Seconds to wait for a pass
- ← **waitGap Float:** Interval between subsequent comparisons
- ← **slotNo Integer:** (optional) Used to specify the slot number to call WaitImageNoMatch on.

Returns:

Array: An array of tuples consisting of slot number,true/false,last image captured as StormTestImageObject ,actual percentage, number of captures performed, total time taken

Example:

```
ret = WaitIconNoMatch(refIcon, percent, location, algorithm, timeToWait, waitGap)
print ret
>>>>[[8, False, <StormTestImageObject instance>,99.756356898506851, 5, 4.5179998874664307]]
```

Note:

This function ignores the effect of SetVideoOffset when the connected server is an HD server. It accounts for the offset when the connected server is Standard Definition.

2.6.2.13 `CompareColorEx` **(image, color, tolerances** = (4,4, **flatness** = 98, **peakError** = 0, **includedAreas** = None, **excludedAreas** = None**)**

Description

This function performs a single color compare on an image. It calculates the average color in each RGB component and compares that with the specified reference color. A tolerance is applied to each value so that if the calculated color is within the tolerance (in either direction) then it is considered a match. Thereafter a 'flatness' is calculated and a 'peak error' is calculated of tolerance. The 'flatness' is how 'flat' the color is - whether there is much variation across the area of interest. The rectangle could be, on average gray but have a lot of red and cyan in it which cancel out - the average would be right but the flatness would be very low (100% flat means perfect single true color). The 'peak error' measures how many pixels (as a %) differ from the reference color by more than the given tolerance. We could get a rectangle where the average color matches and the flatness is high but the color difference for large parts of the rectangle exceeds the tolerance by a small amount. This would show up as a high peak error even though the flatness could be high. A high value of flatness and a low value of peak error makes the color comparer very sensitive to variations and is more likely to fail even if, to the human eye, the color matches. It is recomended that you use the Developer Suite trainer to experiment with the settings - it will save a lot of time and experimentation. A match occurs IF the average color is within the tolerance limits AND the calculated flatness is above or equal to the specified value AND the calculated peak error is below or equal to the specified value.

Parameters:

← **image String** or StormTestImageObject : file or image from CaptureImageEx(). In Release 3.1 and later this may be a NamedImage.

← **color Tuple** : red green and blue components of a color that is used as the reference for the comparison. In Release 3.1 and later this may be a NamedColor - in this case the tolerances, flatness and peakError parameters are ignored because the NamedColor has its own values for these parameters.

← **tolerances Tuple** : red green and blue component tolerances. The values are numbers and represent how far from the reference color each RGB component may be and still be considered a match.

← **flatness Double** : the degree of flatness required within the region where the color is calculated. 0 - 100 is the valid range

← **peakError Double** : the peak error (see above description) required within the region where the color is calcualted. 0 - 100 is valid range

← **includedAreas List:** Included areas for comparision, currently only one can be specified. See CompareImageEx for details on format and usage. In Release 3.1 and later the rectangle may be a NamedRegion.

← **excludedAreas List:** Excluded areas for comparision, NOT currently supported.

Returns:

List: true/false, actual color, actual flatness, actual peak error

Example:

```
# validate color at 100x20 pixel rect with top left at (10,10) has a mid gray color (128,128,128) with a
# flatness better than 95%, peak error 1% and allow a tolerance of 8 on each component
ret = CompareColorEx(testImage, (128,128,128),(8,8,8), 95, 1, includedAreas=[[10,10,100,20]])
print ret
>>>[True, (127,129,128), 97, 0]
```

Note:

If a named image, color or region is used then only the general 'all slot' path will be searched because this function does not take a slot number.

2.6.2.14 `WaitColorMatch` (color, tolerances = `(4,4,` flatness = `98`, peakError = `0`, includedAreas = `None`, excludedAreas = `None`, timeToWait = `5`, waitGap = `1`, slotNo = `0`)

Description

Waits for color compare to pass. It tries multiple times until a pass or the timeout is reached. It calculates the average color in each RGB component and compares that with the specified reference color. A tolerance is applied to each value so that if the calculated color is within the tolerance (in either direction) then it is considered a match. Thereafter a 'flatness' is calculated and a 'peak error' is calculated of tolerance. The 'flatness' is how 'flat' the color is - whether there is much variation across the area of interest. The rectangle could be, on average gray but have a lot of red and cyan in it which cancel out - the average would be right but the flatness would be very low (100% flat means perfect single true color). The 'peak error' measures how many pixels (as a %) are away from the average. We could get a rectangle where the average color matches and the flatness is high but a small corner is dramatically different color. This would show up as a high peak error even though the flatness could be high. A high value of flatness and a low value of peak error makes the color comparer very sensitive to variations and is more likely to fail even if, to the human eye, the color matches. It is recomended that you use the Developer Suite trainer to experiment with the settings - it will save a lot of time and experimentation. A match occurs IF the average color is within the tolerance limits AND the calculated flatness is above or equal to the specified value AND the calculated peak error is below or equal to the specified value.

Parameters:

← **color Tuple** : red green and blue components of a color that is used as the reference for the comparison. # In Release 3.1 and later this may be a NamedColor - in this case the tolerances, flatness and peakError parameters are ignored because the NamedColor has its own values for these parameters.

← **tolerances Tuple** : red green and blue component tolerances. The values are numbers and represent how far from the reference color each RGB component may be and still be considered a match.

← **flatness Double** : the degree of flatness required within the region where the color is calculated. 0 - 100 is the valid range

← **peakError Double** : the peak error (see above description) required within the region where the color is calculated. 0 - 100 is valid range

← **includedAreas List:** Included areas for comparision, currently only one can be specified. In Release 3.1 and later the rectangle may be a NamedRegion

← **excludedAreas List:** Excluded areas for comparision, NOT currently supported.

← **timeToWait Integer:** Number of Seconds to wait for a pass

← **waitGap Float:** Interval between subsequent comparisons

← **slotNo Integer:** (optional) Used to specify the slot number to call WaitColorMatch on.

Returns:

List: A list of lists consisting of slot number,true/false,last image captured as StormTestImageObject, actual color, actual flatness, actual peak error, number of captures performed, total time taken

Example:

```
ret = WaitColorMatch((128,128,128),(8,8,8), 95, 1, includedAreas, excludedAreas, timeToWait, waitGap)
print ret
>>>[[8, False, <StormTestImageObject instance>,[129,130,126], 98, 1, 5, 4.5529999732971191]]
```

Note:

This function ignores the effect of SetVideoOffset when the connected server is an HD server. It accounts for the offset when the connected server is Standard Definition.

2.6.2.15 `WaitColorNoMatch` (**color**, **tolerances** = `(4,4,` **flatness** = `98`, **peakError** = `0`, **includedAreas** = `None`, **excludedAreas** = `None`, **timeToWait** = `5`, **waitGap** = `1`, **slotNo** = `0`)

Description

Waits for color compare to fail - that is for an area of the screen to be NOT equal to the color. It tries multiple times until a color mismatch occurs or the timeout is reached. It calculates the average color in each RGB component and compares that with the specified reference color. A tolerance is applied to each value so that if the calculated color is within the tolerance (in either direction) then it is considered a match. Thereafter a 'flatness' is calculated and a 'peak error' is calculated of tolerance. The 'flatness' is how 'flat' the color is - whether there is much variation across the area of interest. The rectangle could be, on average gray but have a lot of red and cyan in it which cancel out - the average would be right but the flatness would be very low (100% flat means perfect single true color). The 'peak error' measures how many pixels (as a %) are away from the average. We could get a rectangle where the average color matches and the flatness is high but a small corner is dramatically different color. This would show up as a high peak error even though the flatness could be high. A high value of flatness and a low value of peak error makes the color comparer very sensitive to variations and is more likely to fail even if, to the human eye, the color matches. It is recomended that you use the Developer Suite trainer to experiment with the settings - it will save a lot of time and experimentation. A match occurs IF the average color is within the tolerance limits AND the calculated flatness is above or equal to the specified value AND the calculated peak error is below or equal to the specified value. This function waits for a match to NOT occur.

Parameters:

← **color Tuple** : red green and blue components of a color that is used as the reference for the comparison. In Release 3.1 and later this may be a NamedColor - in this case the tolerances, flatness and peakError parameters are ignored because the NamedColor has its own values for these parameters.

← **tolerances Tuple** : red green and blue component tolerances. The values are numbers and represent how far from the reference color each RGB component may be and still be considered a match.

← **flatness Double** : the degree of flatness required within the region where the color is calculated. 0 - 100 is the valid range

← **peakError Double** : the peak error (see above description) required within the region where the color is calculated. 0 - 100 is valid range

← **includedAreas List:** Included areas for comparision, currently only one can be specified. See CompareImageEx for details on format and usage. In Release 3.1 and later the rectangle may be a NamedRegion

← **excludedAreas List:** Excluded areas for comparision, NOT currently supported.

← **timeToWait Integer:** Number of Seconds to wait for a pass

← **waitGap Float:** Interval between subsequent comparisons

← **slotNo Integer:** (optional) Used to specify the slot number to call WaitColorNoMatch on.

Returns:

Array: An array of tuples consisting of slot number,true/false,last image captured as StormTestImageObject,actual color, actual flatness, actual peak error, number of captures performed, total time taken A return value of True means that the color on the screen does NOT equal the specified color - that is the function has successfully detected a change of color from that specified.

Example:

```
ret = WaitColorNoMatch((128,128,128),(8,8,8), 95, 1, includedAreas, excludedAreas, timeToWait, waitGap)
print ret
>>>[[8, False, <StormTestImageObject instance>,(129,130,126), 98, 1, 5, 4.5529999732971191]]
```

Note:

This function ignores the effect of SetVideoOffset when the connected server is an HD server. It accounts for the offset when the connected server is Standard Definition.

2.6.2.16 `SetVideoOffset` (**offset**, **slotNo** = `0`, **saveToDatabase** = `False`)

Description

This function is designed to allow a test written with reference images captured from one box to be run on a box with different hardware characteristics, but the same software application, i.e. it could be a different model or from a different manufacturer. In this case, the analogue video capture images will most likely be offset relative to the reference box. The SetVideoOffset() function can be then be used by the test writer to tell StormTest what this offset is. Then StormTest will apply this offset to any image captures (both directly and indirectly that are part of an API call). All image based operations such as OCR, image comparison, navigate will use the offset image.
The actual values that are passed to the function are the number of pixels on each axis that the live video picture needs to be translated by in order for it to match the reference box
For example, if a screen capture from a box is shifted 7 pixels to the left and 2 pixels down relative to a reference box then its top left hand corner is at (-7,2) relative to the reference. It must therefore have 7 added to the x co-ordinate and 2 subtracted from the y co-ordinate in order to bring it in line with the reference.
So, in this example, the user would call SetVideoOffset((7,-2))

Parameters:

← **offset** **Tuple:** This is in the form (x,y) where (x,y) is the amount to translate the live video so that it matches the reference box. In Release 3.1 and later this may be a NamedRegion but only the left and top values will be used.

← **slotNo** **Integer:** (optional) Used to specify the slot number to set an offset. If not used, then the offset will apply to all reserved slots.

← **saveToDatabase** @ Bool: (optional) If true then the value(s) are saved to the database and automatically used by all scripts on that type of DUT thereafter. Use with caution!

Returns:

offsetResult **List:** A list of tuples consisting of slot number and the result. Just a single tuple if explicit slot is used.

Example:

```
>>ret=SetVideoOffset((10, -10))
>>print ret
>>[[1, True], [5, False]]     # result from slot 1 success and from slot 5 fail
```

Note:

This function is deprecated when the connected server is an HD server and has no effect.

2.6.2.17 `GetVideoOffset` (**slotNo** = 0)

Description

This function returns the value of the video offset.

Parameters:

← **slotNo** **Integer:** (optional) Used to specify the slot number to get an offset.

Returns:

videoOffset **List:** A list of tuples consisting of slot number and the offset value.

Example:

```
>>ret=GetVideoOffset()
>>print ret
>>[[1, (10, -10)], [5, (0, 0)]]
```

Note:

This function is deprecated when the connected server is an HD server and always returns (0,0).

2.6.2.18 `CalibrateVideoOffset` **(filename, maxX** = 10, **maxY** = 10, **threshold** = 95, **saveToDatabase** = False, **slotNo** = 0**)**

Description

This function calibrates the current DUT video offset with reference to a reference image It assumes that the DUT is showing the same image as the reference and analyzes the best combination of X & Y offsets and calls SetVideoOffset() internally to set the video offset

Parameters:

← **filename String:** the reference file. It should be same resolution as the DUT is showing. In Release 3.1 and later this may be a NamedImage

← **maxX Integer:** (optional) the maximum possible offset in X direction. The function searches no more than this number of pixels left & right looking for a match

← **maxY Integer:** (optional) the maximum possible offset in Y direction. The function searches no more than this number of pixels up and down looking for a match

← **threshold Integer:** (optional) the % match that is considered acceptable. Too low a figure will cause inaccurate offsets, too high figure may prevent any offset being accepted even though there is a valid value.

← **saveToDatabase Bool** (optional) whether the discovered offset is saved to the database

← **slotNo Integer:** (optional) Used to specify the slot number to calibrate. If not used, then the calibration will occur on all reserved slots sequentially.

Returns:

calibrationArray **Array:** An array of tuples consiting of slot number, status, offset, error. Just a tuple if explicit slot is used.

Example:

```
>>ret = CalibrateVideoOffset("guide.png",10,10,95,False,7)
>>print ret
>>(7, True,(-4,0),3.412593)
```

Note:

this function is deprecated when the connected server is HD

2.6.2.19 `StartVideoLog` (**prefix** = `'Vid'`, **slotNo** = `0`)

Description

This function starts saving the video capture to disk.

Parameters:

← **prefix String:** (optional) Prefix to add to the auto-generated filename.

← **slotNo Integer:** (optional) The slot number to start video logging on.

Returns:

see Multi-dimensional array of results

Example:

```
>>ret=StartVideoLog()
>>print ret
>>[[1, True, "Vid_Slot1_Timestamp.264"], [9, True, "Vid_Slot9_Timestamp.264"]]
```

Note:

If the *slotNo* parameter is omitted the default is to start logging for all slots reserved.

2.6.2.20 `StopVideoLog` (**slotNo** = 0)

Description

This function stops saving the video capture to disk.

Parameters:

← **slotNo** **Integer:** (optional) The slot number to stop video logging on.

Returns:

see Multi-dimensional array of results

Example:

```
>>StopVideoLog()
```

Note:

If the *slotNo* parameter is omitted the default is to stop logging for all slots reserved.

2.6.2.21 `EnableRollingRecord` (**maxTime** = `600`, **location** = `None`, **slotNo** = `0`)

Description

This function enables the rolling record buffer for a slot. The rolling record buffer is saved by using the SaveRecording call. It is disabled by DisableRollingRecord which will reclaim disk space used. The function does some checking for disk space and file creation so may take a few seconds or so to execute.

Parameters:

← **maxTime Integer:** (optional) The max length of the buffer in seconds. There is a hard coded limit of 10 minutes. If you need more than 10 minutes we suggest Start/StopVideoLog is a better option.

← **location String:** (optional) The location of a folder to store the temporary files. If set to None (the default) then a sub folder of the log folder for the slot will be used. The location should be on a disk with very fast access due to the amount of data being written. It need not (and should not) be the same as you intend to save any such buffers.

← **slotNo Integer:** (optional) The slot number to enable the rolling record buffer on.

Returns:

Bool. If slot is zero then a list of lists, each sub list being [slot, bool] where the bool is the status.

Example:

```
>>EnableRollingRecord(60,"c:\\temp")
>>[[1,True],[7,True]]
>>EnableRollingRecord(60, None, 5)
>>True
```

Note:

Temporary files created will be deleted when the slot is freed (FreeSlot or ReleaseServerConnection)

2.6.2.22 `DisableRollingRecord` (**slotNo** = 0)

Description

This disables the rolling record buffer for a slot. The return is True if the rolling record buffer is successfully disabled. All temporary files created by a prior EnableRollingRecord are deleted. It is not an error to call DisableRollingRecord on a slot which has no rolling record buffer. The return will be True so this is a safe call at any time.

Parameters:

← **slotNo Integer:** (optional) The slot number to enable the rolling record buffer on.

Returns:

Bool. If slot is zero then a list of lists, each sub list being [slot, bool] where the bool is the status.

Example:

```
>>DisableRollingRecord()
>>[[1,True],[7,True]]
>>DisableRollingRecord(5)
>>True
```

2.6.2.23 `SaveRecording` (**time**, **fileName**, **slotNo** = 0)

Description

This saves part of the rolling record buffer to disk in the specified file. It is not an error to call this function with a time larger then the maximum time specified in EnableRollingRecord or a value greater than the amount of video so far captured. The function will attempt to save the amount requested but report the actual amount saved.

Parameters:

← **time Integer** : the number of seconds of video and audio to save.

← **fileName String** : the file name to save the video and audio to. It can be relative (to the folder specified in SetDirectories/SetProjectDir) or an absolute path. The extension will be 'corrected' by the API to be .264 (Standard Def systems) or .avi (HD Systems). You are advised to use a path relative to the log folder. From release 3.2.5, logfiles will have a hyperlink to the recording so you can easily view then from a browser. However, for this to work, you must use a path relative to the log folder. You could, in theory, use an absolute path if you were to only run the test from a local file. This cannot work under the daemon as the final resolved path must be in the FTP area of the daemon for hyperlinks to work. Therefore, if the test is run under the daemon and the path is not relative to the log folder (the fileName is either absolute or starts with .. to indicate a parent folder) the hyperlink is not written to the log file but a warning is written instead.

← **slotNo Integer:** (optional) The slot number of the video/audio to save.

Returns:

Tuple or list of lists. If slotNo is non zero then a tuple of (status, fullFileName, seconds) is returned. The status is a bool (True if all OK), fullFileName is the full path to the file saved. If slotNo is zero then the return is a list of lists. The list items are [slotNo, tuple] where tuple is the standard tuple as if slotNo was non zero

Example:

```
>>SaveRecording(10,"myfirsttest.ran",9)
>>(True, 8.5, "c:\\mytests\\mytest_20150503_143407_35\\myfirsttest.264")
>>SaveRecording(30,"myfirsttest.avi")
>>[[5,(True, 30, "c:\\mytests\\mytest_20150503_144510_22\\myfirsttest.avi")],[7,(True, 28, "c:\\mytests\\mytest_2015050
```

Note:

Under the daemon, the log folder is mapped by the client daemon and relative paths for the video file are treated in the same manner as for saving images from the StormTestImageObject's Save() method.

2.6.2.24 `BookmarkVideoLog` ()

Description

When a test script calls StartVideoLog(), a 'bookmark' is automatically created for every call to a StormTest API function. This allows the person reviewing the logs to quickly jump to the point in the video that a particular API was called. However, there are no bookmarks for non-StormTest functions.
This function allows the test writer to create user-defined bookmarks in the video log.

Parameters:

None

Returns:

None

Example:

```
>>BookmarkVideoLog()
```

2.6.2.25 `GetAudioLevel` (**slotNo** = 0)

Description

This function gets the immediate audio level in dBu.

Note:

You cannot call this while running an audio analysis on any channel.

Parameters:

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

Array: The return type is structured as an array of tuples. Each tuple consists of the slot number, status and the audio level in dBu.

Example:

```
>>GetAudioLevel()
>>[[1, True, -20], [2, True, -12]]
```

2.6.2.26 `WaitAudioPresence` (**threshold**, **timeout**, **slotNo** = 0)

Description

This function detects the presence or absence of audio.

Note:

You cannot call this while running an audio analysis on any channel.

Parameters:

← **threshold Float:** threshold value above which will be deemed as audio presence.

← **timeout Float** : The maximum time in seconds to wait for the audio to exceed the threshold.

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

Array: The return type is structured as an array of tuples. Each tuple consists of the slot number, the audio detection result (i.e. True/False), the actual audio level and total time taken.

Example:

```
>>WaitAudioPresence(-20, 2)
>>[[1, True, -8.961853637816823, 0.19401939999999998]]
>>WaitAudioPresence(-20, 2, 1)
>>[True, -8.961853637816823, 0.19401939999999998]
```

2.6.2.27 `WaitAudioAbsence` (**threshold**, **timeout**, **slotNo** = 0)

Description

This function detects the presence or absence of audio.

Note:

You cannot call this while running an audio analysis on any channel.

Parameters:

← **threshold Float:** threshold value above which will be deemed as audio presence.

← **timeout Float** : The maximum time in seconds to wait for the audio to be below the threshold.

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

Array: The return type is structured as an array of tuples. Each tuple consists of the slot number, the audio detection result (i.e. True/False), the actual audio level and total time taken.

Example:

```
>>WaitAudioAbsence(-60, 2)
>>[[1, False, -8.9549860644472332, 1.9291928999999999]]
>>WaitAudioAbsence(-60, 2, 1)
>>[False, -8.9549860644472332, 1.9291928999999999]
```

2.6.2.28 `ResyncAudio` (**slotNo** = 0)

Description

This function resynchronises the HDMI audio signals. Occasionally, DUTs can change audio sampling rate several times in a few seconds. This can cause the hardware to lose audio sync. The symptom is the audio levels are incorrectly detected and there is a 'hissing' sound from any speakers. Should this happen, then calling this function will reset the audio channel hardware - there will be a period of several seconds of silence before the audio recovers. This function is slow in operation - 5 seconds or more is typical and should be used as a last resort. Many DUTs do not change audio rates and for those DUTs, this function is unnecessary. This function does nothing on SD systems. This function requires the server to be at version 3.2.5 or later.

Parameters:

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

None

Exceptions:

StormTestExceptions is thrown on an error, for example if the server is earlier than release 3.2.5.

Note:

You should **not** call this function while running Audio Analysis unless absolutely necessary - it will interfere with the results of the analysis.

2.6.2.29 `IsAudioResyncNeeded` (**slotNo** = 0)

Description

This function determines whether a resync of the HDMI audio signals may be needed. If you see strange audio levels then you may need to call ResyncAudio function. The IsAudioResyncNeeded will indicate whether the system believes you should call ResyncAudio. You can call ResyncAudio even if IsAudioResyncNeeded returns False. This function always returns False on SD systems and if the server version is less than 3.2.5 (because such servers can not perform an audio resync).

Parameters:

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

List : if slotNo is 0 then a list of lists, each list being [slotNo, status]. If slotNo is non zero then a simple Boolean is returned of the status - True if ResyncAudio should be called.

2.6.2.30 `SetResolution` (**format**, **slotNo** = 0)

Description

This function sets the format (i.e. the video resolution) for the video. It should be noted that as the video resolution is increased, it may be necessary to increase the maximum bitrate in order to ensure that video quality is not compromised.

Parameters:

← **format String:** see Supported Video Formats

← **slotNo Integer:** (optional) The slot number to set the resolution on

Returns:

None

Example:

```
>>SetResolution("QCIF",1)  # Sets the resolution to QCIF for slot number 1 only
or
>>SetResolution("QCIF")    # Sets the resolution to QCIF for all slots
```

Note:

If the *slotNo* parameter is omitted the default is to set the resolution for all slots reserved
It is not possible to set the resolution for HD sources connected to HD servers using Hardware Compression and this API call is ignored. For such servers the streamed video is at the same resolution as the raw input video. If GPUCompression has been enabled on an HD server (indicated in GetCapabilities) then this API calls works as for 4K systems.
For sources connected to the 4K server, you can set the resolution to one of the resolutions listed in Supported Video Formats. For optimum CPU performance on the server and lowest bandwidth use, you should use the lowest possible resolution consistent with your needs. The resolution only affects the log files and the quality of the popup window used to view the DUT video. It has no effect on the other test api functions such as CaptureImageEx etc. GPU Compression was added in release 3.3.0 and the server must be at 3.3.0 or later.

2.6.2.31 `GetResolution` **(slotNo** = 0**)**

Description

This function returns the resolution of the encoded video on reserved slots.

Parameters:

← **slotNo Integer:** (optional) Allows the user to get the resolution on just one slot rather than all slots reserved.

Returns:

slotResolution **String Array:** An array of tuples consisting of slot number and resolution. See Supported Video Formats. If a specific slot number is passed to the function, then the return value will be a string indicating the video resolution on that slot.

Example:

```
# Get resolution of all reserved slots
>>ret=GetResolution()
>>print ret
>>[[1, "QCIF"], [3, "CIF"]]   # Slot 1 is at QCIF resolution, slot 3 is at CIF
                              # resolution

# Get resolution of one specific slot
>>ret=GetResolution(3)
>>print ret
>>"CIF"                       # Slot 3 is at CIF resolution
```

Note:

If the *slotNo* parameter is omitted the resolutions returned are for all slots reserved.

2.6.2.32 `GetRawInputInfo` **(slotNo** = 0**)**

Description

This function returns information about the raw input of the reserved slot(s). For some servers there may be a difference between the streamed resolution and the input resolution. The return here returns information about the raw video and, for HD servers, raw audio input parameters. The function returns the most recent set of parameters (audio, in particular can change frequently).

Parameters:

← **slotNo Integer:** (optional). The slot number for which values are wanted. Use 0 for all reserved slots.

Returns:

slotInfo **Dictionary:** A dictionary of keys (each key being a string) and values. If slotNo is zero then a list of slot number, dictionary is returned. The keys supported are:
Width (width in pixels of video)
Height (height in pixels of video)
Interlaced (True for interlaced modes, false for progressive)
FrameRate (number of frames per second. For interlaced mode, this is the field rate - this matches general marketing material where 1080i50 is 50 Hz but really 25 completer frames/second.)
HardwareCaptureRate (number of frames per second that the hardware captures images for high speed timer and video quality analysis purposes) This is never above the FrameRate but for some hardware it may be below the frame rate.
ColorSpace (the color space as VideoColorSpace enumerated type). Only for 4K systems - Unknown for HD systems. SD is analog so no color space.
PixelClock (the pixel clock frequency in Hz). Only for 4K systems - Unknown for all other systems.
AudioRawBitDepth (the number of bits per audio sample at the hardware level. Only for 4K systems, all others systems return None as it is unknown)
AudioRate (the audio bit rate in Hz)
AudioBitDepth (the bits per sample for audio that are used in StormTest for all purposes on the server. For compatibility purposes, the 4K system uses 16 bits even if raw input was 20 or 24 bits)
AudioChannels (the number of audio channels in the raw audio stream 1 or 2)

2.6.2.33 `SetMaxBitRate` (**bitRate**, **slotNo** = 0)

Description

This function sets the maximum bit rate to be used when encoding the video from a particular slot. Note that this function does not set the *actual* bit rate that is used, it is simply defining a maximum ceiling that the encoder should not cross.

Parameters:

← **bitRate Integer:** This is the bitrate to be used in bits per second, e.g. a value of 512000 would set the bit rate to 512 kbps. For Standard Definition systems a minimum value of 30000 is allowed and for HD systems using hardware compression the minimum is 128000. For 4K systems and HD systems using GPU compression the minimum is 1/500 of the raw uncompressed bit rate. So, if you set the resolution to be 720 x 576 and frame rate of 30 then the raw bit rate is 720 x 576 x 30 x 12 (12 bits/pixel always) leading to a raw rate of 149 Mbit/sec so a minimum compressed rate of 300 kbps. No error is reported on setting an inappropriate rate but GetMaxBitRate() can be used to retrieve the setting. GPU compression was added in release 3.3.0 and requires the server to be at release 3.3.0 or later. There is no maximum value defined as the encoder will simply not be able to reach the maximum bit rate beyond a certain level. Standard Definition systems have a usable maximum of 4Mbit/sec and HD systems can use up to 12MBit/sec.

← **slotNo Integer:** (optional) The slot number to set the bitrate on.

Returns:

None

Example:

```
>>SetMaxBitRate(512000,1) # Sets the maximum bit rate to 512000bps on slot 1
  or
>>SetMaxBitRate(512000)   # Sets the maximum bit rate to 512000bps on all slots
```

Note:

If the *slotNo* parameter is omitted the default is to set the maximum bit rate for all slots reserved.

2.6.2.34 `GetMaxBitRate` (**slotNo** = 0)

Description

This function returns the maximum bit rate that the video encoder can use when encoding the video.

Parameters:

← **slotNo Integer:** (optional) Allows the user to get the maximum bit rate on just one slot rather than all slots reserved.

Returns:

bitRate **Integer Array:** An array of tuples consisting of slot number and maximum encoding bit rate. If a specific slot number is passed to the function, then the return value will be an integer indicating the max bit rate on that slot.

Example:

```
# Get maximum bit rate of all reserved slots
>>ret = GetMaxBitRate()
>>print ret
>>[[1, 512000], [4, 128000]]    # Slot 1 is using a max of 512kbps, slot 4 is using
                                # a max of 128kbps

# Get maximum bit rate of one specific slot
>>ret = GetMaxBitRate(4)
>>print ret
>>128000                        # Slot 4 is using a max of 128kbps
```

Note:

If the *slotNo* parameter is omitted the maximum bit rates for all slots reserved are returned.

2.6.2.35 `SetFrameRate` (**frameRate, slotNo** = 0)

Description

This function sets the frame rate for the streaming content.

Parameters:

← **frameRate Integer:** The new frame rate. The valid frame rate depends on the server and slot. For Standard Definition, the value is between 1 and 25 for PAL DUTs, 1 and 30 for NTSC DUTs. For HD systems, the valid range is 1 to 60. However, the actual frame rate is limited to 30 on HD systems if the source is 1920 x 1080.

← **slotNo Integer:** (optional) The slot number to set the frame rate on.

Returns:

None

Example:

```
>>SetFrameRate(25,1)  # Sets the frame rate to 25 on slot 1
  or
>>SetFrameRate(25)    # Sets the frame rate to 25 on all slots
```

Note:

If the *slotNo* parameter is omitted the default is to set the frame rate for all slots reserved.

2.6.2.36 `GetFrameRate` (**slotNo** = 0)

Description

This function returns the frame rate that the video is being encoded with on reserved slots.

Parameters:

← **slotNo Integer:** (optional) Allows the user to get the frame rate on just one slot rather than all slots reserved.

Returns:

frameRate **Integer Array:** An array of tuples consisting of slot number and frame rate. If a specific slot number is passed to the function, then the return value will be an integer indicating the frame rate on that slot.

Example:

```
# Get frame rate of all reserved slots
>>ret=GetFrameRate()
>>print ret
>>[[1, 25], [5, 12], [6, 12]]      # Slot 1 running at 25 fps, slot 5 and 6 at 12 fps

# Get frame rate of one specific slot
>>ret=GetFrameRate(5)
>>print ret
>>12                               # Slot 5 is running at 12 fps
```

Note:

If the *slotNo* parameter is omitted the frame rates for all slots reserved are returned.

2.6.2.37 `SetVideoStandard` (**videoStandard, slotNo** = 0)

Description

This function sets the video standard for the streaming content.

Parameters:

← **videoStandard VideoStandard:** The new video standard.

← **slotNo Integer:** (optional) The slot number to set the video standard on.

Returns:

Status **Boolean:** True if the call succeeded on all slots, else False.

Example:

```
>>SetVideoStandard(VideoStandard.NTSC)  # Sets the video standard to NTSC on all reserved slots
>>True
  or
>>SetVideoStandard(VideoStandard.NTSC,1)  # Sets the video standard to NTSC on slot 1
>>True
```

Note:

If the *slotNo* parameter is omitted the default is to set the frame rate for all slots reserved.

2.6.2.38 `ResetDefaultParameters` (**slotNo** = 0)

Description

This function returns the video settings to the default values. These include the frame rate, the resolution and the bit rate.

Parameters:

← **slotNo Integer:** (optional) The slot number to return to original values.

Returns:

Error indication. If the client fails to communicate with the server (either ConnectToServer has not been called or server has died), then None is returned. If the slot number is invalid or not reserved then 0 is returned. On success, the result is a list of 2 elements, the first element is 1 (always) and the second element is the list of slots for which the parameters were reset (this has 1 element if a single slot is reserved)

Example:

```
>>ret = ResetDefaultParameters()
>>print ret
>>[1,[6]]
```

Note:

If the *slotNo* parameter is omitted, the parameters of all slots reserved are set to the default values.

2.6.2.39 `RegisterVideoLossCallback` (callbackFunction)

Description

This function allows the user to register to be notified if video loss occurs on a slot. The video state is checked every 5 seconds and a callback generated upon a state change.

Parameters:

← **callbackFunction Function:** The callback function that should be called in case video loss occurs.

Returns:

Bool: true if the callbackFunction has successfully been registered. false if a server does not support this feature.

Note:

The callback function is defined as follows:

callbackFunction(slotNo, state)

Parameters:

← **slotNo Integer:** The slot number the video state change occurred on.

← **state VideoSignalState:** The state of the video.

2.6.2.40 `SetDesignToWorldTransform` (**transform**, **slotNo** = 0)

Description

Sets the transformation to use between design and world coordinates for the specified slot. This function overrides any other transform, including SetVideoOffset(). After this call, most functions which were affect by SetVideoOffset now use the specified transform. The value is not saved to the database. To set a transform in the database, please see GetDutInstanceInSlot() and UpdateDutModel().

Parameters:

← **transform CoordConverter** or List: transform to use. It may be a CoordConverter object OR a List/Tuple that can be passed directly to the constructor of a CoordConverter. Only 1 transform is stored internally as world to design is the inverse of design to world.

← **slotNo Integer:** The slot number where the transform is set. If slotNo is zero then the transform is applied to all reserved slots.

Returns:

None

Note:

The CaptureImageEx is not altered by the transform - this allows you to capture the raw image for you own processing even when a severe transform is set. The older deprecated (and no longer documented) functions are also not affected by the transform.

Example:

```
>>>cc = CoordConverter()
>>>cc.SetFromRectangles([0,0,704,576], [-10,-10, 1930, 1090])
>>>SetDesignToWorldTransform(cc)
```

2.6.2.41 `GetDesignToWorldTransform` (**slotNo** = 0)

Description

Gets current design to world transform for the specified slot. Such a transform always exists - if user has done nothing special then this is the Identity transform which does nothing.

Parameters:

← **slotNo Integer:** the slot number to retrieve the transform for. If 0 then the transforms for all slots are returned

Returns:

CoordConverter: The transform. If slotNo is 0 then a List of [slot, transform] is returned, one for each reserved slots.

Example:

```
>>>print GetDesignToWorldTransform(5)
>>><stormtest.ClientAPI.CoordConverter object>
>>>cc = CoordConverter()
>>>cc.SetFromRectangles([0,0,704,576], [-10,-10, 1930, 1090])
>>>SetDesignToWorldTransform(cc)
>>>print GetDesignToWorldTransform()
>>>[[5, <stormtest.ClientAPI.CoordConverter object>]]
```

2.6.2.42 `SetWorldToDesignTransform` (**transform**, **slotNo** = 0)

Description

Sets the transformation to use between world and design coordinates for the specified slot. This function overrides any other transform, including SetVideoOffset(). After this call, most functions which were affect by SetVideoOffset now use the specified transform. The value is not saved to the database. To set a transform in the database, please see GetDutInstanceInSlot() and UpdateDutModel().

Parameters:

← **transform CoordConverter** or List: transform to use. It may be a CoordConverter object OR a List/Tuple that can be passed directly to the constructor of a CoordConverter. Only 1 transform is stored internally as design to world is the inverse of world to design.

← **slotNo Integer:** The slot number where the transform is set. If slotNo is zero then the transform is applied to all reserved slots.

Returns:

None

Note:

The CaptureImageEx is not altered by the transform - this allows you to capture the raw image for you own processing even when a severe transform is set. The older deprecated (and no longer documented) functions are also not affected by the transform.

Example:

```
>>>cc = CoordConverter()
>>>cc.SetFromRectangles([0,0,704,576], [-10,-10, 1930, 1090])
>>>SetWorldToDesignTransform(cc)
```

2.6.2.43 `GetWorldToDesignTransform` **(slotNo** = 0**)**

Description

Gets current world to design transform for the specified slot. Such a transform always exists - if user has done nothing special then this is the Identity transform which does nothing.

Parameters:

← **slotNo Integer:** the slot number to retrieve the transform for. If 0 then the transforms for all slots are returned

Returns:

CoordConverter: The transform. If slotNo is 0 then a List of [slot, transform] is returned, one for each reserved slots.

Example:

```
>>>print GetWorldToDesignTransform(5)
>>><stormtest.ClientAPI.CoordConverter object>
>>>cc = CoordConverter()
>>>cc.SetFromRectangles([0,0,704,576], [-10,-10, 1930, 1090])
>>>SetDesignToWorldTransform(cc)
>>>print GetWorldToDesignTransform()
>>>[[5, <stormtest.ClientAPI.CoordConverter object>]]
```

2.7 Directory Path API

The functions in this section are used to configure the directories used by StormTest.

Functions

- SetProjectDir
- SetDirectories
- GetProjectDir
- GetDirectories
- GetLogFileDirectory

2.7.1 Detailed Description

The functions in this section are used to configure the directories used by StormTest.

The user is recommended to call these function before calling ReserveSlot()

2.7.2 Function Documentation

2.7.2.1 `SetProjectDir` **(projRoot)**

Description

This function sets the project root directory. The directories specified in the SetDirectories() function are relative to the directory passed into this function. If a test does not call this function, the project directory defaults to '.' (i.e. the current working directory). If this is called after ConnectToServer() but before ReserveSlot() there will be some debug output in directory relative to the current working directory and then the rest in the directory specified by SetProjectDir(). This is because ConnectToServer() can generate debug outut. It is recommended therefore to call this function prior to ConnectToServer()

Parameters:

← **projRoot String:** The path to the project root.

Returns:

None

Example:

```
>>SetProjectDir(os.path.join("C:\\","my_project_dir"))
```

2.7.2.2 `SetDirectories` (**logDir** = `None`, **refImgDir** = `None`, **autoCreateSubDir** = `True`)

Description

This function sets the location of the logfiles and the reference files relative to the root defined in SetProjectDir(). If SetProjectDir() is not called, then they will be relative to the current working directory (i.e. where the test is run from). If this function is not called by a test, then the reference images must be in the same directory as the test.
Note that the logDir **must** be set before ReserveSlot() is called (as this is when logging commences). Any attempt to change the logDir after ReserveSlot() has been called will be ignored. The refImgDir can be altered at any time and the new value will apply for all subsequent image comparisons.

Parameters:

← **logDir String:** The path to storage location of the log files. This is ignored when the test is run under the Client Daemon and the log folder is determined by the Client Daemon.

← **refImgDir String:** The path to storage location of the reference images.

← **autoCreateSubDir String:** (optional) DEPRECATED. Should be set to True. A unique directory is always created for log files in releases later than 2.0.

Returns:

None

Example:

```
>>SetDirectories( os.path.join("logs","test1"), os.path.join("reference","images") )
```

2.7.2.3 `GetProjectDir` ()

Description

This function gets the project root directory. This may have been set previously by a call to SetProjectDir(). This is an optional function.

Parameters:

None

Returns:

projPath **String:** The path to the project root.

Example:

```
>>ret=GetProjectDir()
>>print ret
>>C:\my_project_dir
```

2.7.2.4 `GetDirectories` ()

Description

This function gets the location of the logfiles and the reference files relative to the root defined in SetProjectDir(). If SetProjectDir() was not called, then they will be relative to the current working directory (i.e. where the test is run from).

Parameters:

None

Returns:

dirPath **List:** A list of directory paths. The first path is the output directory where all the log files are stored. The second is the input directory, where the reference images are loaded from.

Example:

```
>>dirPaths=GetDirectories()
>>print dirPath
>>['logs\\test1', 'reference\\images']
```

2.7.2.5 `GetLogFileDirectory` ()

Description

This function gets the exact full path of the log file directory. StormTest creates a unique directory for each script run based on date and time. The root directory of these log file directories can be retrieved by GetDirectories(), however, the exact directory being used cannot be got from that. This function will provide the exact path.

Parameters:

None

Returns:

dirPath **String:** A string with full absolute path of log file directory

Example:

```
>>dirPath=GetLogFileDirectory()
>>print dirPath
>>c:\users\some_user\tests\detect_guide_20110621_101534
```

Note:

If GetLogFileDirectory() is called before SetDirectories() or ConnectToServer() then the log file directory will not exist but the path returned is correct. If SetDebugLevel() is called to disable logging, GetLogFileDirectory() will still return a valid path but the path will not exist.

2.8 Named Resources API

The functions in this section provide access to named resources.

Functions

- SetResourcePath
- GetResourcePath
- SetNamedResourceMatchCase
- GetNamedResourceMatchCase
- FindNamedRegion
- FindNamedColor
- FindNamedString
- FindNamedImage
- FindNamedScreenDefinition
- WriteNamedRegion
- DeleteNamedRegion
- ReadNamedRegion
- RenameNamedRegion
- WriteNamedImage
- DeleteNamedImage
- ReadNamedImage
- RenameNamedImage
- WriteNamedString
- DeleteNamedString
- ReadNamedString
- RenameNamedString
- WriteNamedColor
- DeleteNamedColor
- ReadNamedColor
- RenameNamedColor
- WriteNamedScreenDefinition
- DeleteNamedScreenDefinition
- ReadNamedScreenDefinition
- RenameNamedScreenDefinition
- CreateNamedFolder
- DeleteNamedFolder
- RenameNamedFolder
- ListNamedResources

2.8.1 Detailed Description

The functions in this section provide access to named resources.

The named resources are an extension to the 3.0 use of resources within a StormTest Navigator. They allow test scripts to be written in a more robust manner making maintenance and updates easier.

A resource is a region, image, color, string or screen definition. It is identified by name. Resources are organised into namespaces - these are analogous to folders on a disk filing system (but since the resource is stored in the database not in a file, the term namespace is more accurate). Both name and namespace are case sensitive but this may be overriden.

In release 3.2 model specific resources are supported. All of the Write/Delete/Read calls were defined with a dutModelId parameter which was marked as 'reserved' in earlier releases. It is no longer reserved. The full explanation is supplied only on the Region API calls - all others are analogous and you should read the Region description to understand how the other resources work.

We define 2 types of resource: a generic and a model specific. For all resources, it is impossible for a model specific variant to exist without also a generic existing - if you just try to create a model specific then a generic will also be created with the same value. Likewise, if you delete the generic then all model specific variants are also deleted. To fully use the power of model specific resources requires that your model names (as defined in Admin Console) are unique across the facility. If this is not the case then the behavior is undefined - when maintaining resources where a model name is specified that is not unique, you will get an exception. However, there is nothing to stop you creating a new model with the same name as another and getting confused during lookup of resources. Internally, all models are stored as a unique database id not by name and that is unique.

All resources created using versions of StormTest prior to 3.2 are generic and code written for earlier versions will work but operate on the generic version (and in the case of delete, will delete all versions of the resource).

The example under WriteNamedRegion shows how model specific resources for multiple models are managed. It is a long example showing all the combinations of use cases.

2.8.2 Function Documentation

2.8.2.1 `SetResourcePath` (**path**, **slotNo** = `'all'`)

Description

This function sets the base path for named resources for a slot. Each slot can have a path for its resources. By default, it is empty which means that resources need to be referenced by the full path. This function sets the base path for a slot and then later code can use a relative name. Setting slotNo = 'all' sets the path for all slots (both reserved and yet to be reserved). It overrides any prior path with an explicit slot. A slot does not need to be reserved to use this function. The slot number is always a physical slot. The path may be a string, or a list/tuple of strings. If a list/tuple of strings then the list/tuple is searched for relative lookups in order until a match is found.

Parameters:

← **path object:** A string, tuple or list specifying the path to the resources

← **slotNo integer:** (optional): The slot number to which the path should refer. The string 'all' may be used to mean all slots.

Returns:

success **Bool:** True if the path was set successfully

Example:

```
>>SetResourcePath("projects/my team/new box")   # all slots will use resources in 'projects/my team/new box' namespace
```

2.8.2.2 `GetResourcePath` **(slotNo =** `'all'`**)**

Description

Gets the base path for named resources for a slot. This may be called before reserving a slot. slotNo has same meaning as for SetResourcePath()

Parameters:

← **slotNo integer:** (optional) The slot number to which the path should refer. The string 'all' may be used to mean all slots.

Returns:

path **object:** The path. A string or tuple matching whatever was passed in for the SetResourcePath.

Example:

```
>>SetResourcePath("projects/my team/new box")
>>SetResourcePath(["project/his team/old box", "projects/nowhere/old box"], 5)
>>print GetResourcePath()
>>projects/my team/new box
>>print GetResourcePath(5)
>>['project/his team/old box', 'projects/nowhere/old box']
>>print GetResourcePath(7)
>>None                      # no specific path for slot 7.
```

2.8.2.3 `SetNamedResourceMatchCase` (**caseSensitive** = `True`)

Description

Sets the case matching for named resource lookup. By default, the API is case sensitive. Use this function with caseSensitive = False to make the API case insensitive. The setting applies to all later calls which look up the names. The whole lookup, including all paths, is performed either in case sensitive or case insensitive manner – not just the name portion.

Parameters:

← **caseSensitive bool:** (optional) Set the case sensitive. If omitted, the functions are case sensitive.

Returns:

None

Example:

```
>>>SetNamedResourceMatchCase(False)
```

2.8.2.4 `GetNamedResourceMatchCase` ()

Description

This function returns the current state of case sensitive match. Default is true if SetNamedResourceMatchCase() has not been called.

Parameters:

None

Returns:

caseSensitive **bool:** true if current mode is case sensitive

Example:

```
>>>print GetNamedResourceMatchCase()
>>>True
```

2.8.2.5 `FindNamedRegion` (name)

Description

This function gets a named region object corresponding to the name item. The name may be an absolute name (starting with /) or relative name (not starting with /). An absolute object ignores the path(s) set by SetResourcePath. A relative object will use the path set by SetResourcePath plus the slot number to resolve the final Region. This returns a lightweight object that does not contain the real region at this point. A method needs to be called to resolve the final region but this object can be passed to many functions needing a rectangle and the function will use the slot number to find the actual rectangle.

Parameters:

← **name string:** The name of a region

Returns:

object **NamedRegion** - the NamedRegion object.

Note:

The object tracks changes in the SetResourcePath but also that it caches each resolution of the Region per slot.
There is no requirement for the named region to actually exist at the time this function is called, however, it must exist at the time of the final use of the named region.

Example:

```
>>>print FindNamedRegion("myregion")
>>><stormtest.stormtest_client.NamedRegion object>
```

2.8.2.6 `FindNamedColor` (name)

Description

This function gets a named color object corresponding to the name item. The name may be an absolute name (starting with /) or relative name (not starting with /). An absolute object ignores the path(s) set by SetResourcePath. A relative object will use the path set by SetResourcePath plus the slot number to resolve the final color. This returns a lightweight object that does not contain the real color at this point. A method needs to be called to resolve the final color but this object can be passed to many functions needing a color and the function will use the slot number to find the actual color.

Parameters:

← **name string:** The name of a color

Returns:

object **NamedColor** - the NamedColor object.

Note:

The object tracks changes in the SetResourcePath but also that it caches each resolution of the color per slot.
There is no requirement for the named color to actually exist at the time this function is called, however, it must exist at the time of the final use of the named color.

Example:

```
>>>print FindNamedColor("my projects/new project/mycolor")
>>><stormtest.stormtest_client.NamedColor object>
```

2.8.2.7 `FindNamedString` (name)

Description

This function gets a named string object corresponding to the name item. The name may be an absolute name (starting with /) or relative name (not starting with /). An absolute object ignores the path(s) set by SetResourcePath. A relative object will use the path set by SetResourcePath plus the slot number to resolve the final string. This returns a lightweight object that does not contain the real string at this point. A method needs to be called to resolve the final string but this object can be passed to many functions needing a string and the function will use the slot number to find the actual string.

Parameters:

← **name string:** The name of a string

Returns:

object **NamedString** - the NamedString object.

Note:

The object tracks changes in the SetResourcePath but also that it caches each resolution of the string per slot.
There is no requirement for the named string to actually exist at the time this function is called, however, it must exist at the time of the final use of the named string.

Example:

```
>>>print FindNamedString("/my projects/new project/some string")
>>><stormtest.stormtest_client.NamedString object>
```

2.8.2.8 `FindNamedImage` (name)

Description

This function gets a named image object corresponding to the name item. The name may be an absolute name (starting with /) or relative name (not starting with /). An absolute object ignores the path(s) set by SetResourcePath. A relative object will use the path set by SetResourcePath plus the slot number to resolve the final image. This returns a lightweight object that does not contain the real image at this point. A method needs to be called to resolve the final image but this object can be passed to many functions needing a image and the function will use the slot number to find the actual image.

Parameters:

← **name string:** The name of a image

Returns:

object **NamedImage** - the NamedImage object.

Note:

The object tracks changes in the SetResourcePath but also that it caches each resolution of the image per slot.
There is no requirement for the named image to actually exist at the time this function is called, however, it must exist at the time of the final use of the named image.

Example:

```
>>>print FindNamedImage("my projects/new project/my good image")
>>><stormtest.stormtest_client.NamedImage object>
```

2.8.2.9 `FindNamedScreenDefinition` **(name)**

Description

This function gets a named screen definition object corresponding to the name item. The name may be an absolute name (starting with /) or relative name (not starting with /). An absolute object ignores the path(s) set by SetResourcePath. A relative object will use the path set by SetResourcePath plus the slot number to resolve the final image. This returns a lightweight object that does not contain the real screen definition at this point. A method needs to be called to resolve the final screen definition but this object can be passed to many functions needing a screen definition and the function will use the slot number to find the actual screen definition.

Parameters:

← **name string:** The name of a screen definition

Returns:

object **NamedScreenDefinition** - the _namedSdo object.

Note:

The object tracks changes in the SetResourcePath but also that it caches each resolution of the screen definition per slot.
There is no requirement for the named screen definition to actually exist at the time this function is called, however, it must exist at the time of the final use of the named screen definition.

Example:

```
>>>print FindNamedScreenDefinition("my projects/new project/my first sdo")
>>><stormtest.stormtest_client.NamedScreenDefinition object>
```

2.8.2.10 `WriteNamedRegion` (**regionPath**, **regionData**, **comment** = `None`, **dutModelId** = `0`)

Description

Writes the regionData to the database as regionPath. regionPath is always case sensitive and SetNamedResourceMatchCase is ignored.

Parameters:

← **regionPath string:** the full absolute path to the region (may begin with a / but not necessary), path elements separated by /. Final element is the region name.

← **regionData tuple:** (left, top, width, height) of the region.

← **comment string:** (optional) Free form text of a user defined comment.

← **dutModelId Integer**, String or List : The model for which the data is to be written. The integer value of 0 means the generic model only will be created/updated. It can also be an explicit model number, a string representing a single model or a list of strings representing multiple models.

Returns:

None: An exception is thrown on an error (for example bad parameters, no communication).

Example:

```
# this example shows all the possible use cases of the model specific resources.
# to simplify the code, we define a variable for the path and ignore the comment field, setting it to None
# each time (you can set it to any string you like in practice)
>>>regionName = "/my projects/title"
# create a generic region:
>>>WriteNamedRegion(regionName,[0,0,100,20])
# Now create a specific region for a Model with id 999 (looked up from ListDutModels).
>>>WriteNamedRegion(regionName,[10,10,100,20], None, 999)
# Now create another region for models 'A' and 'B' (neither are id 999)
>>>WriteNamedRegion(regionName,[20,20,120,30], None, ['A', 'B'])
# Now create another region but for models 'B' and 'C'. After this call we will have
# a generic, one for model 999, one for model 'A' and one for models 'B' and 'C' - 4 items in database
>>>WriteNamedRegion(regionName,[30,30,100,30], None, ['B', 'C'])
# update just model 'C'
>>>WriteNamedRegion(regionName,[40,40,100,30], None, 'C')
>>>print ReadNamedRegion(regionName)
>>>[[0, (0, 0, 100, 20)]]
>>>print ReadNamedRegion(regionName, 999)
>>>[[u'SomeModel', (10, 10, 100, 20)]]
>>>print ReadNamedRegion(regionName, 'A')
>>>[[u'A', (20, 20, 120, 30)]]
>>>print ReadNamedRegion(regionName, 'B')
>>>[[u'B', (30, 30, 100, 30)]]
>>>print ReadNamedRegion(regionName, 'C')
>>>[[u'C', (40, 40, 100, 30)]]
>>>WriteNamedRegion(regionName,[9,9,800,10],None, ['A' ,'B', 'C'])
>>>print ReadNamedRegion(regionName, 'A')
>>>[[u'A', (9, 9, 800, 10)]]
>>>print ReadNamedRegion(regionName, 'B')
>>>[[u'B', (9, 9, 800, 10)]]
>>>print ReadNamedRegion(regionName, 'C')
>>>[[u'C', (9, 9, 800, 10)]]
>>>print ReadNamedRegion(regionName, 'D')
>>>Traceback (most recent call last):
...
StormTestExceptions: ReadNamedRegion: No model specific region exists for model D
```

2.8.2.11 `DeleteNamedRegion` (**regionPath**, **dutModelId** = `0`)

Description

Deletes the named region regionPath.

Parameters:

← **regionPath string:** the full absolute path to the region (may begin with a / but not necessary), path elements separated by /. Final element is the region name.

← **dutModelId integer**, String or List: The model for which the region is to be deleted. The integer value of 0 means the generic model and all variants will be deleted. It can also be an explicit model number, a string representing a single model or a list of strings representing multiple models. It is not an error to delete a region with a model where there is no model specific region for that region - just a waste of CPU cycles and network bandwidth.

Returns:

None: An exception is thrown on an error (for example bad parameters, no communication).

Example:

```
# This example expands the WriteNamedRegion to perform some deletion
>>>regionName = "/my projects/title"
>>>WriteNamedRegion(regionName,[0,0,100,20])
>>>WriteNamedRegion(regionName,[20,20,120,30], None, ['A', 'B'])
>>>WriteNamedRegion(regionName,[30,30,100,30], None, ['B', 'C'])
>>>print ReadNamedRegion(regionName)
>>>[[0, (0, 0, 100, 20)]]
>>>print ReadNamedRegion(regionName, 'A')
>>>[[u'A', (20, 20, 120, 30)]]
>>>print ReadNamedRegion(regionName, 'B')
>>>[[u'B', (30, 30, 100, 30)]]
>>>print ReadNamedRegion(regionName, 'C')
>>>[[u'C', (30, 30, 100, 30)]]
>>>DeleteNamedRegion(regionName, 'C')
>>>print ReadNamedRegion(regionName, 'B')
>>>[[u'B', (30, 30, 100, 30)]]
>>>print ReadNamedRegion(regionName, 'C')
>>>StormTestExceptions: ReadNamedRegion: No model specific region exists for model C
>>>DeleteNamedRegion(regionName)
>>>print ReadNamedRegion(regionName, 'B')
>>>StormTestExceptions: ReadNamedRegion: ReadNamedRegion failed: The named entity does not exist
```

2.8.2.12 `ReadNamedRegion` (**regionPath**, **dutModelId** = 0)

Description

Reads the named region regionPath

Parameters:

← **regionPath string:** the full absolute path to the region (may begin with a / but not necessary), path elements separated by /. Final element is the region name.

← **dutModelId integer** or String: The model to read. If 0 then the generic verion is read. It can also be the model name or an id retrieved from ListDutModels()

Returns:

List: Each list item of the list has 2 elements, the first is the dutModelId and the second is a tuple of the region (left, top, width, height). The dutModelId is 0 for the generic region and the model name in Unicode for others. If a dutModelId is supplied as an integer, the return is the model name, not the numeric id. It was intended at one stage to return all variants in one function but that was changed due to potential network overhead but the format of a list of lists has been retained. An exception is thrown on an error (for example bad parameters, no communication).

Example:

```
>>>print ReadNamedRegion(regionName)
>>>[[0, (0, 0, 100, 20)]]
>>>print ReadNamedRegion(regionName, 'A')
>>>[[u'A', (20, 20, 120, 30)]]
```

2.8.2.13 `RenameNamedRegion` **(regionPath, newName)**

Description

Renames the NamedRegion regionPath to newName. The NamedRegion must exist and the newName must not exist.

Parameters:

← **regionPath string:** the full absolute path to the region (may begin with a / but not necessary), path elements separated by /. Final element is the region name.

← **newName string:** just the name portion and cannot include path components.

Returns:

None

Exceptions:

StormTestExceptions is thrown on an error (for example bad parameters, no communication).

Note:

This cannot be used to move regions, just to change the name. All model specific versions are renamed so this function does not need a dutModelId parameter.

Example:

```
>>>RenameNamedRegion("myproject/region","title")
```

2.8.2.14 `WriteNamedImage` (**imagePath**, **imageData**, **comment** = `None`, **dutModelId** = 0)

Description

Writes the imageData to the database as imagePath. imagePath is always case sensitive and SetNamedResourceMatchCase is ignored.

Parameters:

← **imagePath string:** the full absolute path to the image (may begin with a / but not necessary), path elements separated by /. Final element is the image name.

← **imageData tuple:** (binary data, width, height) of the image. The binary data is of a form that can be read/written to a file and is understood to be an image (eg a PNG file).

← **comment string:** (optional) Free form text of a user defined comment.

← **dutModelId Integer**, String or List : The model for which the data is to be written. The integer value of 0 means the generic model only will be created/updated. It can also be an explicit model number, a string representing a single model or a list of strings representing multiple models.

Returns:

None: An exception is thrown on an error (for example bad parameters, no communication).

Note:

see WriteNamedRegion for an exhaustive example.

Example:

```
>>>fp = open('c:\\users\\me\\sample.png', 'rb')
>>>image = fp.read()
>>>fp.close()
# we know this image is 100x60 but you could use PIL to find it.  The size is for guidance only because in most
# uses of the image, the system will check the image size.
>>>WriteNamedImage('myprojects/logoicon',(image, 100,60))
```

2.8.2.15 `DeleteNamedImage` (**imagePath, dutModelId** = 0)

Description

Deletes the named image imagePath.

Parameters:

← **imagePath string:** the full absolute path to the image (may begin with a / but not necessary), path elements separated by /. Final element is the image name.

← **dutModelId integer**, String or List: The model for which the image is to be deleted. The integer value of 0 means the generic model and all variants will be deleted. It can also be an explicit model number, a string representing a single model or a list of strings representing multiple models. It is not an error to delete an image with a model where there is no model specific image for that image - just a waste of CPU cycles and network bandwidth.

Returns:

None: An exception is thrown on an error (for example bad parameters, no communication).

Example:

```
>>>DeleteNamedImage('myprojects/logoicon')
```

2.8.2.16 `ReadNamedImage` (**imagePath**, **dutModelId** = 0)

Description

Reads the named image imagePath

Parameters:

← **imagePath string:** the full absolute path to the image (may begin with a / but not necessary), path elements separated by /. Final element is the image name.

← **dutModelId integer** or String: The model to read. If 0 then the generic verion is read. It can also be the model name or an id retrieved from ListDutModels()

Returns:

List: Each list item of the list has 2 elements, the first is the dutModelId and the second a tuple of the image (binary data, width, height). The binary data can be saved direct to a file. The dutModelId is 0 for the generic image and the model name in Unicode for others. If a dutModelId is supplied as an integer, the return is the model name, not the numeric id. It was intended at one stage to return all variants in one function but that was changed due to potential network overhead but the format of a list of lists has been retained. An exception is thrown on an error (for example bad parameters, no communication).

Example:

```
>>>result = ReadNamedImage('myprojects/logoicon')
>>>print result
>>>[[0, ('\x89PNG...', 100, 60)]]
>>>fp = open('c:\\users\\me\\res.png','wb')
>>>fp.write(result[0][1][0])
>>>fp.close()
```

2.8.2.17 `RenameNamedImage` (imagePath, newName)

Description

Renames the Named image imagePath to newName. The Named image must exist and the newName must not exist.

Parameters:

← **imagePath string:** the full absolute path to the image (may begin with a / but not necessary), path elements separated by /. Final element is the image name.

← **newName string:** just the name portion and cannot include path components.

Returns:

None. An exception is thrown on an error (for example bad parameters, no communication).

Note:

This cannot be used to move images, just to change the name. All model specific versions are renamed so this function does not need a dutModelId parameter.

Example:

```
>>>RenameNamedImage("myproject/image","icon")
```

2.8.2.18 `WriteNamedString` (**stringPath**, **stringData**, **comment** = `None`, **dutModelId** = `0`)

Description

Writes the stringData to the database as stringPath. stringPath is always case sensitive and SetNamedResourceMatchCase is ignored.

Parameters:

← **stringPath string:** the full absolute path to the string (may begin with a / but not necessary), path elements separated by /. Final element is the string name.

← **stringData string:** the string.

← **comment string:** (optional) Free form text of a user defined comment.

← **dutModelId integer**, String or List : The model for which the data is to be written. The integer value of 0 means the generic model only will be created/updated. It can also be an explicit model number, a string representing a single model or a list of strings representing multiple models.

Returns:

None: An exception is thrown on an error (for example bad parameters, no communication).

Note:

see WriteNamedRegion for an exhaustive example.

Example:

```
>>>WriteNamedString('myprojects/titletext',"My Cool DUT")
```

2.8.2.19 `DeleteNamedString` (**stringPath**, **dutModelId** = 0)

Description

Deletes the named string stringPath.

Parameters:

← **stringPath string:** the full absolute path to the string (may begin with a / but not necessary), path elements separated by /. Final element is the string name.

← **dutModelId integer**, String or List: The model for which the string is to be deleted. The integer value of 0 means the generic model and all variants will be deleted. It can also be an explicit model number, a string representing a single model or a list of strings representing multiple models. It is not an error to delete a string with a model where there is no model specific string for that string - just a waste of CPU cycles and network bandwidth.

Returns:

None: An exception is thrown on an error (for example bad parameters, no communication).

Example:

```
>>>DeleteNamedString('myprojects/titletext')
```

2.8.2.20 `ReadNamedString` (**stringPath**, **dutModelId** = 0)

Description

Reads the named string stringPath

Parameters:

← **stringPath string:** the full absolute path to the string (may begin with a / but not necessary), path elements separated by /. Final element is the string name.

← **dutModelId integer** or String: The model to read. If 0 then the generic verion is read. It can also be the model name or an id retrieved from ListDutModels()

Returns:

List: Each list item of the list has 2 elements, the first is the dutModelId and the second is the string. The dutModelId is 0 for the generic string and the model name in Unicode for others. If a dutModelId is supplied as an integer, the return is the model name, not the numeric id. It was intended at one stage to return all variants in one function but that was changed due to potential network overhead but the format of a list of lists has been retained. An exception is thrown on an error (for example bad parameters, no communication).

Example

```
>>>print ReadNamedString('myprojects/titletext')
>>>[[0,'My Cool DUT']]
```

2.8.2.21 `RenameNamedString` (stringPath, newName)

Description

Renames the Named string stringPath to newName. The Named string must exist and the newName must not exist.

Parameters:

← **stringPath string:** the full absolute path to the string (may begin with a / but not necessary), path elements separated by /. Final element is the string name.

← **newName string:** just the name portion and cannot include path components.

Returns:

None. An exception is thrown on an error (for example bad parameters, no communication).

Note:

This cannot be used to move strings, just to change the name. All model specific versions are renamed so this function does not need a dutModelId parameter.

Example:

```
>>>RenameNamedString('myprojects/titletext'.'no text')
```

2.8.2.22 `WriteNamedColor` (**colorPath**, **colorData**, **comment** = `None`, **dutModelId** = `0`)

Description

Writes the colorData to the database as colorPath. colorPath is always case sensitive and SetNamedResourceMatchCase is ignored.

Parameters:

← **colorPath string:** the full absolute path to the color (may begin with a / but not necessary), path elements separated by /. Final element is the color name.

← **colorData tuple:** (color tuple, tolerance tuple, flatness, peak error) of the color.

← **comment string:** (optional) Free form text of a user defined comment.

← **dutModelId integer**, String or List : The model for which the data is to be written. The integer value of 0 means the generic model only will be created/updated. It can also be an explicit model number, a string representing a single model or a list of strings representing multiple models.

Returns:

None: An exception is thrown on an error (for example bad parameters, no communication).

Note:

see WriteNamedRegion for an exhaustive example.

Example:

```
>>>WriteNamedColor('myprojects/hilight color',((255,0,0),(10,10,10),98.3,1.5))
```

2.8.2.23 `DeleteNamedColor` (**colorPath**, **dutModelId** = 0)

Description

Deletes the named color colorPath.

Parameters:

← **colorPath string:** the full absolute path to the color (may begin with a / but not necessary), path elements separated by /. Final element is the color name.

← **dutModelId integer**, String or List: The model for which the color is to be deleted. The integer value of 0 means the generic model and all variants will be deleted. It can also be an explicit model number, a string representing a single model or a list of strings representing multiple models. It is not an error to delete a color with a model where there is no model specific color for that color - just a waste of CPU cycles and network bandwidth.

Returns:

None: An exception is thrown on an error (for example bad parameters, no communication).

Example:

```
>>>DeleteNamedColor('myprojects/hilight color')
```

2.8.2.24 `ReadNamedColor` (**colorPath**, **dutModelId** = 0)

Description

Reads the named color colorPath

Parameters:

← **colorPath** **string:** the full absolute path to the color (may begin with a / but not necessary), path elements separated by /. Final element is the color name.

← **dutModelId** **integer** or String: The model to read. If 0 then the generic verion is read. It can also be the model name or an id retrieved from ListDutModels()

Returns:

List: Each list item of the list has 2 elements, the first is the dutModelId and the second a tuple of the color (color tuple, tolerance tuple, flatness, peak error). The dutModelId is 0 for the generic string and the model name in Unicode for others. If a dutModelId is supplied as an integer, the return is the model name, not the numeric id. It was intended at one stage to return all variants in one function but that was changed due to potential network overhead but the format of a list of lists has been retained. An exception is thrown on an error (for example bad parameters, no communication).

Example:

```
>>>print ReadNamedColor('myprojects/hilight color')
>>>[[0,((255,0,0),(10,10,10),98.3,1.5)]]
```

2.8.2.25 `RenameNamedColor` (colorPath, newName)

Description

Renames the Named color colorPath to newName. The Named color must exist and the new-Name must not exist.

Parameters:

← **colorPath string:** the full absolute path to the color (may begin with a / but not necessary), path elements separated by /. Final element is the color name.

← **newName string:** just the name portion and cannot include path components.

Returns:

None. An exception is thrown on an error (for example bad parameters, no communication).

Note:

This cannot be used to move colors, just to change the name. All model specific versions are renamed so this function does not need a dutModelId parameter.

Example:

```
>>>RenameNamedColor('myprojects/hilight color', 'lolite')
```

2.8.2.26 `WriteNamedScreenDefinition` **(screenPath, screenDefinition, comment** = `None`**, dutModelId** = `0`**)**

Description

Writes the screenDefinition to the data base as screenPath. screenPath is always case sensitive and SetNamedResourceMatchCase is ignored. There is no requirement for the screenPath to end in the same name as the embedded name of the screenDefinition. Likewise the comment is not related to the embedded comment property.

Parameters:

← **screenPath string:** the full absolute path to the screen definition (may begin with a / but not necessary), path elements separated by /. Final element is the screen definition name in the database.

← **screenDefinition ScreenDefinition:** the ScreenDefinition object.

← **comment string:** (optional) Free form text of a user defined comment.

← **dutModelId integer**, String or List : The model for which the data is to be written. The integer value of 0 means the generic model only will be created/updated. It can also be an explicit model number, a string representing a single model or a list of strings representing multiple models.

Returns:

None: An exception is thrown on an error (for example bad parameters, no communication).

Note:

see WriteNamedRegion for an exhaustive example.

Example:

```
>>>sdo = ScreenDefinition()
>>>WriteNamedScreenDefinition('myprojects/empty sdo',sdo)
```

2.8.2.27 `DeleteNamedScreenDefinition` (**screenPath, dutModelId** = 0)

Description

Deletes the named screen definition screenPath.

Parameters:

← **screenPath string:** the full absolute path to the screen definition (may begin with a / but not necessary), path elements separated by /. Final element is the screen definition name in the database.

← **dutModelId integer**, String or List: The model for which the screen definition is to be deleted. The integer value of 0 means the generic model and all variants will be deleted. It can also be an explicit model number, a string representing a single model or a list of strings representing multiple models. It is not an error to delete a screen definition with a model where there is no model specific screen definition for that screen definition - just a waste of CPU cycles and network bandwidth.

Returns:

None: An exception is thrown on an error (for example bad parameters, no communication).

Example:

```
>>>DeleteNamedScreenDefinition('myprojects/empty sdo')
```

2.8.2.28 `ReadNamedScreenDefinition` (**screenPath, dutModelId** = 0)

Description

Reads the named screen definition screenPath

Parameters:

← **screenPath string:** the full absolute path to the screen definition (may begin with a / but not necessary), path elements separated by /. Final element is the screen definition name.

← **dutModelId integer** or String: The model to read. If 0 then the generic verion is read. It can also be the model name or an id retrieved from ListDutModels()

Returns:

List: Each list item of the list has 2 elements, the first is the dutModelId and the second the screen definition. The dutModelId is 0 for the generic string and the model name in Unicode for others. If a dutModelId is supplied as an integer, the return is the model name, not the numeric id. It was intended at one stage to return all variants in one function but that was changed due to potential network overhead but the format of a list of lists has been retained. An exception is thrown on an error (for example bad parameters, no communication).

Example

```
>>>print ReadNamedScreenDefinition('myprojects/empty sdo')
>>>[[0,<stormtest.ClientAPI.ScreenDefinition object>]]
@endocde
```

2.8.2.29 `RenameNamedScreenDefinition` (**screenPath, newName)**

Description

Renames the Named Screen Definition screenPath to newName. The Named screen definition must exist and the newName must not exist.

Parameters:

← **screenPath string:** the full absolute path to the screen definition (may begin with a / but not necessary), path elements separated by /. Final element is the screen definition name.

← **newName string:** just the name portion and cannot include path components.

Returns:

None. An exception is thrown on an error (for example bad parameters, no communication).

Note:

This cannot be used to move screen definitions, just to change the name. All model specific versions are renamed so this function does not need a dutModelId parameter.

Example:

```
>>>RenameNamedScreenDefinition('myprojects/empty sdo', 'another sdo')
```

2.8.2.30 `CreateNamedFolder` (folderPath)

Description

Creates an empty folder for storing named resources. This is for completeness and allows scripts to create folders and entities separately, possibly from an external database. It is not necessary to create all intermediate paths – this function will create intermediate folders as needed.

Parameters:

← **folderPath string:** the full absolute path (may begin with a / but not necessary), path elements separated by /.

Returns:

None. An exception is thrown on an error (for example bad parameters, no communication).

Example:

```
>>>CreateNamedFolder('/my projects/backup')
```

2.8.2.31 `DeleteNamedFolder` (folderPath)

Description

Deletes a folder and all contents (including sub folders).

Parameters:

← **folderPath string:** the full absolute path (may begin with a / but not necessary), path elements separated by /.

Returns:

None. An exception is thrown on an error (for example bad parameters, no communication).

Note:

This is a very dangerous function to use. You can delete everything by using the empty string and there is no undelete option.

Example:

```
>>>DeleteNamedFolder('/my folder/my project/temp')
```

2.8.2.32 `RenameNamedFolder` **(folderPath, newName)**

Description

Renames the Named folder folderPath to newName. The Named folder must exist and the newName must not exist.

Parameters:

- ← **folderPath string:** the full absolute path (may begin with a / but not necessary), path elements separated by /.
- ← **newName string:** just the name portion and cannot include path components.

Returns:

None. An exception is thrown on an error (for example bad parameters, no communication).

Note:

This cannot be used to move folder, just to change the name.

Example:

```
>>>RenameNamedFolder('/my projects/backup','old backup')
```

2.8.2.33 `ListNamedResources` (**path**, **whatToList** = 63, **recurseLevel** = 0, **dutModelId** = 0)

Description

Returns a list of Resources, recursing as necessary.

Parameters:

← **path string:** the full absolute path (may begin with a / but not necessary), path elements separated by /. Can be empty for the root.

← **whatToList integer:** made up of bit wise OR to limit the type of data to return. The values are: 1 => namespaces, 2 => regions, 4 => colors, 8 => images, 16 => Screen Definition Objects, 32 => strings. So setting whatToList = 63 will return everything (0 returns nothing), 1 would return just name spaces.

← **recurseLevel string:** determines how many levels to recurse through namespaces. 0 is just one level down from path.

← **dutModelId integer**, String or List: The models to include in the return list. The integer value of 0 means the generic model only will be returned. It can also be an explicit model number, a string representing a single model or a list of strings representing multiple models. It can also be the empty list [] meaning all possible models. It is not an error to include a model here which has no variants in the folder.

Returns:

List: Each item in the list is a named resource. The named resource is a list of 8 elements. These are [path relative to input path, type of resource as whatToList, comment, dutModelId, creation date, creating user, modification date, modifying user]. The dates are epoch seconds (seconds since 1/1/1970 00:00:00 UTC). the dutModelId is 0 for the generic and a unicode string for all model specific resources. The order within the list is not guaranteed and is simply the fastest way to retrieve from the database.

Example:

```
# from the WriteNamedRegion example
>>>print ListNamedResources('/my projects', recurseLevel = 0, dutModelId='A')
>>>[[u'title', 2, u'', u'A', 1406040427, u'me', 1406040988, u'me']]
>>>print ListNamedResources('/my projects', recurseLevel = 0, dutModelId=[])
>>>[[u'title', 2, u'', 0, 1406034434, u'me', 1406040987, u'me'], [u'title', 2, u'', u'SomeModel', 1406034533, u'me', 14
[u'title', 2, u'', u'A', 1406040427, u'me', 1406040988, u'me'], [u'title', 2, u'', u'B', 1406040988, u'me', 1406040988,
[u'title', 2, u'', u'C', 1406040988, u'me', 1406040988, u'me']]
```

2.9 MetaData API

Some objects in the system allow the addition of metadata.

Enums

- enum MetaDataType

Functions

- AddMetaData
- RemoveMetaData
- ReadMetaData

2.9.1 Detailed Description

Some objects in the system allow the addition of metadata.

This was introduced in StormTest version 3.2. Each item of metadata is a name, value pair where both are strings. Some of the metadata is interpreted by StormTest but most of it is not. To avoid confusion, all metadata items that have meaning to StormTest have names starting with an underscore '_'. Users should not create or modify items beginning with the underscore. This will avoid unexpected StormTest behavior. All names not beginning with an underscore are for the end user. It should be noted that adding meta data to items may prevent some of them being deleted until the meta data has been removed.

2.9.2 Function Documentation

2.9.2.1 `AddMetaData` (**objType, objInstance, metaName, metaValue, allowUpdate** = `True`)

Description

Adds meta data to an object. Meta data is a keyword = value pair of strings. Only a sub set of objects can support meta data.

Parameters:

← **objType String** or Integer: the type of object. It must be one of the MetaDataType enumeration. To support adding meta data to model specific resources, the objType can also have a suffix of ".modelname" where modelname is the case sensitive name of a model

← **objInstance String:** the instance of the object. For the named resources, this is the full path. For other items it is the id, either as a string or as an integer

← **metaName String:** the name of meta data to add. Users should not use names beginning with an underscore (_) - these are reserved for StormTest system meta data.

← **metaValue String:** the value of meta data to add.

← **allowUpdate String:** (optional). If the meta data already exists and this flag is true then the existing meta data is updated. If this flag is False and the meta data exists then this function fails

Returns:

None On error an exception is thrown

Note:

The API checks that the specified item exists before adding the meta data.

Example:

```
# add "Stable Build" = "0" to dut model 3750. Meaning known only to user code
>>>AddMetaData(MetaDataType.DutModel,3750,"Stable Build","0")
# adds "Stable Build" = "0" to a named region /Tests/MyRegion but only to the version for 'MyModel'
>>>AddMetaData(MetaDataType.NamedRegion + ".MyModel","/Tests/MyRegion","Stable Build","0")
```

2.9.2.2 `RemoveMetaData` (objType, objInstance, metaName)

Description

Removes meta data to an object. Meta data is a keyword = value pair of strings. Only a sub set of objects can support meta data.

Parameters:

← **objType String** or Integer: the type of object. It must be one of the MetaDataType enumeration. To support adding meta data to model specific resources, the objType can also have a suffix of ".modelname" where modelname is the case sensitive name of a model

← **objInstance String:** the instance of the object. For the named resources, this is the full path. For other items it is the id, either as a string or as an integer

← **metaName String:** the name of meta data to remove. Use the empty string ("") to remove all user meta data. Use "_" to remove all system meta data - this is unwise in general but is provided for advanced users.

Returns:

None On error an exception is thrown.

Note:

The API checks that the specified item exists before removing the meta data. Removing non exsitent meta data is NOT an error - it is always safe to call RemoveMetaData() with metaName as empty so long as the objType and objInstance exist.

Example:

```
# remove "Stable Build" from dut model 3750.
>>>RemoveMetaData(MetaDataType.DutModel,3750,"Stable Build","0")
# remove all user meta data from the named region /Tests/MyRegion and model "MyModel". System meta data is not affected
>>>RemoveMetaData(MetaDataType.NamedRegion + ".MyModel","/Tests/MyRegion","")
```

2.9.2.3 `ReadMetaData` (objType, objInstance, metaName)

Description

Read meta data on an object. Meta data is a keyword = value pair of strings. Only a sub set of objects can support meta data.

Parameters:

← **objType String** or Integer: the type of object. It must be one of the MetaDataType enumeration. To support adding meta data to model specific resources, the objType can also have a suffix of ".modelname" where modelname is the case sensitive name of a model

← **objInstance String:** the instance of the object. For the named resources, this is the full path. For other items it is the id, either as a string or as an integer

← **metaName String:** the name of meta data to read. Use the empty string ("") to read all meta data, including system meta data.

Returns:

List: Each item in the list is a pair of strings: the meta data name and the meta data value. This is true even when an explicit item is requested. If no meta data exists then an empty list is returned. On error an exception is thrown.

Note:

The API checks that the specified item exists before reading the meta data. Reading non exsitent meta data is NOT an error - it is always safe to call ReadMetaData() with metaName as empty so long as the objType and objInstance exist.

Example:

```
>>>ReadMetaData(MetaDataType.DutModel,3750)
>>>[["Stable Build","0"],["Key1","value1"]]
```

2.10 Debug output API

This section is used to configure the level of output from the StormTest client and where that output is sent to.

Enums

- enum OutputTo
- enum DebugLevel

Functions

- SetDebugLevel
- SetLogPageSize
- WriteDebugLine
- SetHtmlThumbnailSize
- GetHtmlThumbnailSize

2.10.1 Detailed Description

This section is used to configure the level of output from the StormTest client and where that output is sent to.

2.10.2 Function Documentation

2.10.2.1 `SetDebugLevel` (**trace**, **level**, **serialLog** = 1)

Description

This function sets the verbosity of the output from a script. The default is to print out error, warning and info messages.

Parameters:

← **trace Integer:** Turn on/off function entry/exit trace, see note [1] .

← **level Integer:** Sets the verbosity level, see note [2] . In Release 2.8 and later, an enumeration from DebugLevel can also be used.

← **serialLog Integer:** (optional) Turn on/off serial logging in the main log file, see note [3] .

Returns:

None

Example:

```
>>SetDebugLevel(1,4)
# Error, warning, information and trace messages turned on
```

Note:

1. The variable *trace* can be set to 0 or 1 for OFF or ON
2. The variable *level* can be set to the following values:
   1: Prints out no messages.
   2: Prints out error messages.
   3: Prints out error and warning messages.
   4: Prints out error, warning and information messages.
   5: Prints out error, warning, info and verbose messages.
3. The variable *serialLog* can be set to 0 or 1 for OFF or ON. The default is 1 (i.e. ON). Calling SetDebugLevel(0, 0) will prevent any log file from being created.

2.10.2.2 `SetLogPageSize` (pageSize)

Description

This function sets the size of pages of the HTML log file. For a long running test, the log file may grow so large that it cannot be loaded into a browser or the StormTest Developer Suite. From Release 3.3.0 onwards, the HTML log file is broken into pages (with links between them) to solve this problem. The default size of each page is 20MB. However, you can control this size by using SetLogPageSize() as the first function in your script. A small page size will make loading of each page much faster and the browser/log viewer more responsive. However, review of the overall test log will require more clicks and automated log scanning tools would need changing. Set the pageSize to None to disable paging totally and restore the pre 3.3.0 behavior for large log files (so they are suitable for automated tools but not for loading into a browser) This funtion will throw an exception if called after the log file starts.

Parameters:

← **pageSize Integer** : Size, in bytes of each page of the generated HTML log file. This value is approximate as the code must write HTML closing tags after the size limit is reached. Also, it will not break a line of log across files. Use pageSize = None to disable paging. The API imposes no maximum value on pageSize and a minimum of 30000 - if you set a value below this an exception is raised because such a low value is likely to lead to a file per line of log output.

Returns:

None

Example:

```
>>SetLogPageSize(5000000)        # set 5MB pages
>>SetLogPageSize(None)           # no paging at all
```

2.10.2.3 `WriteDebugLine` (**commentToLog**, **debugLevel** = `None`)

Description

This function writes a comment to the log

Parameters:

← **commentToLog String:** The comment to be written to the log

← **debugLevel DebugLevel:** The level to use for the comment. If omitted, the comment is always put into the log file. This parameter requires release 2.8 and later.

Returns:

None

Example:

```
>>WriteDebugLine("This is a comment")
# Writes the debug messages to the to the log file
```

2.10.2.4 `SetHtmlThumbnailSize` (**scaleFactor, minSize)**

Description

This function sets the thumbnail size in the HTML log file. All images saved as thumbnails after this call will use these values. This function may be called at any time - however it should NOT be called at the very start of the script if the next line is SetDebugLevel(0,0) to turn off logging. This function is available in release 3.4

Parameters:

← **scaleFacotr float:** The amount to scale the raw image by for the thumbnail. The value must be less than 1 (you can't make a thumbnail bigger than the raw image).

← **minSize Tuple:** A tuple or list of 2 integers specifying the minimum dimensions in the horizontal and vertical dimensions respectively. Whatever the scale factor, the thumbnail will be no smaller than the minSize.

Returns:

None

Exceptions:

StormTestExceptions is thrown if the parameters are invalid.

Example:

```
>>SetHtmlThumbnailSize(0.25, (80,60))
# Thumbnails are 25% (1/4) of raw size but no smaller than 80 x 60
```

Note:

The default system scale is 0.125 and (32,24) for the minimum size.

2.10.2.5 `GetHtmlThumbnailSize` ()

Description

This function returns the current scale factor and minimum size of the thumbnails that will be written to the HTML log file. This function may be called at any time. This function is available in release 3.4

Parameters:

None

Returns:

Tuple: A tuple of scale factor, minimum size as described by SetHtmlThumbnailSize

Example:

```
>>GetHtmlThumbnailSize()
>>(0.125, (32, 24))
```

2.11 Screen Definition Object API

This section contains the functions and classes that allow a script to use Screen Definition Objects.

Enums

- enum ReturnScreen
- enum OCRStringCorrection
- enum OCRRegion
- enum ImageRegion
- enum ColorRegion
- enum MotionRegion
- enum AudioRegion
- enum Color
- enum RectangleMatrix
- enum CustomRegion
- enum AlgorithmDescriptor
- enum ScreenDefinition

Functions

- WaitScreenDefMatch
- WaitScreenDefNoMatch
- ScreenDefMatch
- ScreenDefMatchImage
- TestRawZapFrames
- FindRawZapMatch
- FindRawZapNoMatch

2.11.1 Detailed Description

This section contains the functions and classes that allow a script to use Screen Definition Objects.

This was introduced in version 3.2 of StormTest. It is a higher level of control than the underlying Image compare, color compare, OCR string, audio detection and motion detection.

The easiest way to create Screen Definition Objects is by using the graphical StormTest Developer Suite Tool. However, they can be created in code in Python.

2.11.2 Function Documentation

2.11.2.1 `WaitScreenDefMatch` (**screenDef**, **timeToWait** = 5, **waitGap** = 1, **slotNo** = 0)

Description

Evaluates the screenDef. It is implemented as a call to ScreenDefinition.WaitMatch.

WaitScreenDefMatch(sdo, waitTime, waitGap, slotNo) is identical to sdo.WaitMatch(waitTime, waitGap, slotNo) including the return values and types.

Parameters:

← **screenDef ScreenDefinition** : The screen definition object to evaluate

← **timeToWait Float** : Optional. The maximum time to wait in seconds for the VerifyStatus of the screen definition to become true. Default is 5 seconds.

← **waitGap Float** : Optional. The time between successive attempts at verification in seconds. This is the time to wait after a failure before trying again.

← **slotNo Integer** : Optional. The slot to use for the evaluation of the screen definition object. If slotNo is non zero then all reserved slots are used.

Returns:

List : If slotNo is non zero then the returned list is [screen definition, time waited, number of captures made]. If slotNo is zero then the return is a list of lists, each sub list being [slotNo, screen definition, time waited, number of captures made] :

```
>>>WaitScreenDefMatch(sdo, 4, 0.1)
>>>[[5, <ClientAPI.ScreenDefinition object>, 1.2388, 4]]
>>>WaitScreenDefMatch(sdo, slotNo=5)
>>>[<ClientAPI.ScreenDefinition object>, 1.2388, 6]
```

2.11.2.2 `WaitScreenDefNoMatch` (**screenDef, timeToWait** = 5, **waitGap** = 1, **slotNo** = 0)

Description

Evaluates the screenDef. It is implemented as a call to ScreenDefinition.WaitNoMatch.

WaitScreenDefNoMatch(sdo, waitTime, waitGap, slotNo) is identical to sdo.WaitNoMatch(waitTime, waitGap, slotNo) including the return values and types.

Parameters:

← **screenDef ScreenDefinition** : The screen definition object to evaluate

← **timeToWait Float** : Optional. The maximum time to wait in seconds for the VerifyStatus of the screen definition to become false. Default is 5 seconds.

← **waitGap Float** : Optional. The time between successive attempts at non verification in seconds. This is the time to wait after a success before trying again.

← **slotNo Integer** : Optional. The slot to use for the evaluation of the screen definition object. If slotNo is non zero then all reserved slots are used.

Returns:

List : If slotNo is non zero then the returned list is [screen definition, time waited, number of captures made]. If slotNo is zero then the return is a list of lists, each sub list being [slotNo, screen definition, time waited, number of captures made]

Example:

```
>>>WaitScreenDefNoMatch(sdo, 4, 0.1)
>>>[[5, <ClientAPI.ScreenDefinition object>, 1.2388, 4]]
>>>WaitScreenDefNoMatch(sdo, slotNo=5)
>>>[<ClientAPI.ScreenDefinition object>, 1.2388, 6]
```

2.11.2.3 `ScreenDefMatch` **(screenDef, slotNo** = 0**)**

Description

Captures an image from the slot and executes the verification stage of screenDef once. It is implemented as a call to ScreenDefinition.Match.

ScreenDefMatch(sdo, slotNo) is identical to sdo.Match(0, slotNo) including the return values and types.

Parameters:

← **screenDef ScreenDefinition** : The screen definition object to evaluate

← **slotNo Integer** : Optional. The slot number to use for evaluation.

Returns:

ScreenDefinition or List : If slotNo is non zero then a single ScreenDefinition is returned, otherwise a List is returned. Each item in the list is a list of [slotNo, ScreenDefinition] where slotNo is the reserved slot. In all cases the returned ScreenDefinition is a copy and the original screen definition is unchanged.

Example:

```
>>>sdo = ScreenDefinition()
>>>print ScreenDefMatch(4)
>>><ClientAPI.ScreenDefinition object>
>>>print ScreenDefMatch()
>>>[[4, <ClientAPI.ScreenDefinition object>], [5, <ClientAPI.ScreenDefinition object>]]
```

2.11.2.4 `ScreenDefMatchImage` (**screenDef, image, slotNo** = 0)

Description

Executes the verification stages of screenDef on the image and if that passes, executes the read stages. The image may be a StormTestImageObject, PIL image, NamedImage or String. The screenDef cannot include any MotionRegions or AudioRegions as they cannot be evaluated on an image. A StormTestExceptions exception is thrown if either of those region types are present. It is implemented as a call to ScreenDefinition.Match.

ScreenDefMatchImage(sdo, im, slotNo) is identical to sdo.Match(im, slotNo) including the return values and types.

Parameters:

← **screenDef ScreenDefinition** : The screen definition object to evaluate

← **image PIL** Image, StormTestImageObject, NamedImage or String : The image to use for matching. If a string is used then it is assumed to be a file and will be loaded using the same semantics as the StormTestImageObject.FromFile() method. This option is available on Release 3.3.3 and later.

← **slotNo Integer** : The slotNo is used to determine how resources are resolved. If slotNo is 0 then generic resources are used, otherwise resources for the specific model in the specified slot are used. There is no requirement here that the specified slot is reserved when using this method on an image (however some slot must be reserved for the server to accept the request).

Returns:

ScreenDefinition : The processed ScreenDefinition as a copy and the original screen definition is unchanged.

Example:

```
>>>sdo = ScreenDefinition()
>>>im = Image.open("c:\\users\\me\\myimg.png")
>>>print ScreenDefMatchImage(sdo, im)          # use generic lookup of resources
>>><ClientAPI.ScreenDefinition object>
>>>print ScreenDefMatchImage(sdo, im, 7)       # use resources for dut model in slot 7
>>><ClientAPI.ScreenDefinition object>
>>>print ScreenDefMatchImage(sdo, "c:\\users\\me\\myimg.png")
>>><ClientAPI.ScreenDefintion object>
```

2.11.2.5 `TestRawZapFrames` (**screenDef, frameIndex, slotNo** = 0)

Description

Evaluates the screen definition over one or more arbitrary frames captured from the high speed timer. The screen definition cannot contain a MotionRegion or an AudioRegion and a StormTestExceptions exception is raised if it does.

It is implemented as a call to ScreenDefinition.TestRawZapFrames.

TestRawZapFrames(sdo, frameIndex, slotNo) is identical to sdo.TestRawZapFrames(frameIndex, slotNo) including the return values and types.

Parameters:

← **screenDef ScreenDefinition** : The screen definition object to evaluate

← **frameIndex Integer** or List : The frame of frames to test. It can be a single integer for a single frame test or a list of integers. If a list then multiple tests are performed. Frame indexes are zero based.

← **slotNo Integer** : Optional. The slot number. There must have been a prior execution of the high speed timer on this slot with frames captured.

Returns:

ScreenDefinition or List : see example code for the combinations of return types and input parameters.

Note:

There is no requirement for any particular order of frames with specifying a list: [6, 10, 56] is as valid as [99, 3, 78].

Example:

```
>>>ReserveVideoTimer()
>>>EnableCaptureZapFrames()
>>>TriggerHighSpeedTimer()
>>>WaitSec(5)
>>>StopHighSpeedTimer()
# now we can work on the zap frames
>>>sdo = ScreenDefinition()
>>>sdo.Load("c:\\users\\me\\mysdo.stscreen")
>>>print TestRawZapFrames(sdo, 5, slotNo=1)
>>><ClientAPI.ScreenDefinition object>
>>>print TestRawZapFrames(sdo, [6,7,8], slotNo=1)
>>>[(6, <ClientAPI.ScreenDefinition object>), (7, <ClientAPI.ScreenDefinition object>),(8, <ClientAPI.ScreenDefinition
>>>print TestRawZapFrames(sdo, 5)
>>>[[1,<ClientAPI.ScreenDefinition object>]]
>>>print TestRawZapFrames(sdo, [6,7,8])
>>>[[1, [(6, <ClientAPI.ScreenDefinition object>), (7, <ClientAPI.ScreenDefinition object>),(8, <ClientAPI.ScreenDefini
```

2.11.2.6 `FindRawZapMatch` (**screenDef**, **startIndex**, **maxFrames**, **slotNo** = 0)

Description

Evaluates the screen definition over a range of contiguous previously captured frames from the high speed timer looking for a match of the screen definition (VerifyStatus returned as True). A MotionRegion may be used in the screen definition - it will only examine frames up to the last specified in the range. It will calculate the timeout based on the frame rate of the original capture not on real time to do the testing.

It is implemented as a call to ScreenDefinition.FindRawZapMatch.

FindRawZapMatch(sdo, startIndex, maxFrames, slotNo) is identical to sdo.FindRawZapMatch(startIndex, maxFrames, slotNo) including the return values and types.

Parameters:

← **screenDef ScreenDefinition** : The screen definition object to evaluate

← **startIndex Integer** : Zero based start frame to scan for screen definition match.

← **maxFrames Integer** : The maximum number of frames to scan for a match.

← **slotNo Integer** : Optional. The slot number. There must have been a prior execution of the high speed timer on this slot with frames captured. If omitted or 0 then the scan occurs on all reserved slots.

Returns:

Tuple : If slotNo is non zero then a tuple of ScreenDefinition that matched and index where a match was found. If slotNo is zero then a List of slot, tuples. If no match is found then the matched frame is None instead of an Integer.

Example:

```
>>>ReserveVideoTimer()
>>>EnableCaptureZapFrames()
>>>TriggerHighSpeedTimer()
>>>WaitSec(5)
>>>StopHighSpeedTimer()
# now we can work on the zap frames
>>>sdo = ScreenDefinition()
>>>sdo.Load("c:\\users\\me\\mysdo.stscreen")
>>>print FindRawZapMatch(sdo, 0, 25, slotNo=6)
>>>(<ClientAPI.ScreenDefinition object>, 12)
>>>print FindRawZapMatch(sdo, 25, 100)
>>>[[6, (<ClientAPI.ScreenDefinition object>, None)]]
```

2.11.2.7 `FindRawZapNoMatch` (**screenDef**, **startIndex**, **maxFrames**, **slotNo** = 0)

Description

Evaluates the screen definition over a range of contiguous previously captured frames from the high speed timer looking for a NON match of the screen definition (VerifyStatus returned as False). A MotionRegion may be used in the screen definition - it will only examine frames up to the last specified in the range. It will calculate the timeout based on the frame rate of the original capture not on real time to do the testing.

It is implemented as a call to ScreenDefinition.FindRawZapNoMatch.

FindRawZapNoMatch(sdo, startIndex, maxFrames, slotNo) is identical to sdo.FindRawZapNoMatch(startIndex, maxFrames, slotNo) including the return values and types.

Parameters:

← **screenDef ScreenDefinition** : The screen definition object to evaluate

← **startIndex Integer** : Zero based start frame to scan for screen definition non match.

← **maxFrames Integer** : The maximum number of frames to scan for a non match.

← **slotNo Integer** : Optional. The slot number. There must have been a prior execution of the high speed timer on this slot with frames captured. If omitted or 0 then the scan occurs on all reserved slots.

Returns:

Tuple : If slotNo is non zero then a tuple of ScreenDefinition that failed to match and index where the match was not found. If slotNo is zero then a List of slot, tuples. If all frames match then the matched frame is None instead of an Integer.

Example:

```
>>>ReserveVideoTimer()
>>>EnableCaptureZapFrames()
>>>TriggerHighSpeedTimer()
>>>WaitSec(5)
>>>StopHighSpeedTimer()
# now we can work on the zap frames
>>>sdo = ScreenDefinition()
>>>sdo.Load("c:\\users\\me\\mysdo.stscreen")
>>>print FindRawZapNoMatch(sdo, 0, 25, slotNo=6)
>>>(<ClientAPI.ScreenDefinition object>, 12)
>>>print FindRawZapNoMatch(sdo, 25, 100)
>>>[[6, (<ClientAPI.ScreenDefinition object>, None)]]
```

2.12 Optical Character Recognition API

This section contains the functions that allow a script to use the optical character recognition abilities of StormTest.

Enums

- enum ProcessImage

Functions

- OCRSlot
- WaitOCRMatch
- WaitOCRNoMatch
- OCRFile
- OCRImage
- OCRGetRemainingChars
- OCRSetLanguage
- OCRGetLanguage
- OCRSetDefaultImageProcessing
- OCRGetDefaultImageProcessing
- OCRSetImageFiltering
- OCRGetImageFiltering
- CompareStrings
- ComputeStringsDistance
- StringsAutoCorrection

2.12.1 Detailed Description

This section contains the functions that allow a script to use the optical character recognition abilities of StormTest.

StormTest allows a test script to perform OCR on the live stream or on an image file. As OCR works best on high resolution images, the OCRSlot() function always operates on images at the native input resolution (704x576 for PAL, 704 x 480 for NTSC and the detected HD resolution in HD).

2.12.2 Function Documentation

2.12.2.1 `OCRSlot` (**rect** = `None`, **slotNo** = `0`)

Description

This function returns the strings found when OCR has been performed on a capture from a live stream. **Important:** When OCR is performed on the live stream, the resolution used is *always* **4CIF** for Standard Definition servers. This means you must use a 704x576 reference image when calculating what co-ordinates to use for your rectangle. For HD, the resolution is automatically selected based on the input to the server.

Parameters:

← **rect List:** Restricts the OCR to the part of the screen bounded by the rectangular co-ordinates supplied. If no bounding rectangle is required, use *None* here. See note ^{1} for more information. In Release 3.1 and later this may be a NamedRegion or a ScreenDefinition Object. If a ScreenDefinition Object is used then the return parameters change.

← **slotNo Integer:** (optional) The slot number to capture strings on.

Returns:

ocrResult **List:** Contains the OCR statistics and list of strings found. See note ^{2} for more information. If the OCR fails for some reason then None is returned instead of a List. If a ScreenDefinition Object is used as the first parameter then the string return is replaced by the original ScreenDefinition Object from which the results can be simply read. The returned ScreenDefinition object is NOT the same as the input object.

Note:

The ScreenDefinition object may only contain OCRRegions. It **cannot** contain other types of region. To use ScreenDefinitions in version 3.2, it is recommended to use the dedicated Screen Definition API and NOT OCRSlot even when just OCRRegion objects are contained within the ScreenDefinition.

Example:

```
# Perform OCR on all reserved slots
>>ret = OCRSlot(None)
>>print ret
>>(0, 0, 39, 11,                           # Statistics
>> [[2,'This is a string from slot 2'],     # Slot 2 result
>>  [4,'This is the string from slot 4']])  # Slot 4 result

# The above strings are returned from the two slots which are reserved
# The statistics indicate the combined totals for both slots

# Perform OCR on just one slot
>>ret = OCRSlot(None, 4)
>>[0, 0, 30, 7, 'This is the string from slot 4']
```

Note:

If the *slotNo* parameter is omitted, the OCR is performed on all reserved slots.

1. When defining the rectangular area you need to specify the top left hand corner and the width and height of the box, i.e. (x, y, w, h). Remember that the frame of reference is the native screen size not the streaming size (Standard Definition allows downscaling of the streaming video).

2. The statistics returned are:
   (a) Total number of suspicious symbols.
   (b) Total number of unrecognised symbols.
   (c) Total number of all symbols.
   (d) The total number of all words found.
   (e) A list of strings[4] found per slot.
3. If the OCR fails to perform any OCR then None is returned.
4. Where the returned text contains one or more non-ASCII characters, the string will be returned as a unicode type, otherwise it will be a str type. Unicode will need to be encoded to UTF-8 if it is to be used with other StormTest APIs.

This function ignores the effect of SetVideoOffset when the connected server is an HD server. It accounts for the offset when the connected server is Standard Definition.

2.12.2.2 `WaitOCRMatch` (**text**, **rect** = `None`, **timeToWait** = `5`, **waitGap** = `1`, **slotNo** = `0`, **maxDistance** = `0`, **autoCorrect** = `OCRStringCorrection.NoCorrection`)

Description

This function Waits until the specified slot passes the OCR comparison with the OCRed string found when OCR has been performed on a capture from a live stream. **Important:** When OCR is performed on the live stream, the resolution used is *always* **4CIF** for standard definition Servers. This means you must use a 704x576 reference image when calculating what co-ordinates to use for your rectangle. For HD, the resolution is automatically selected based on the input to the server.

Parameters:

← **text String** : Expected Text for comparison. In Release 3.1 and later this may be a NamedString.

← **rect List:** Restricts the OCR to the part of the screen bounded by the rectangular co-ordinates supplied. If no bounding rectangle is required, use *None* here. See note ^{1} for more information. In Release 3.1 and later this may be a NamedRegion.

← **timeToWait Integer:** Number of Seconds to wait for a pass

← **waitGap Float:** Interval between subsequent comparisons

← **slotNo Integer:** (optional) The slot number to capture strings on.

← **maxDistance Integer:** (optional) The maximum allowed Levenshtein distance between the expected text and the OCRed text. Wikipedia can give more details.

← **autoCorrect OCRStringCorrection:** (optional) The type of auto correction to be performed on input string. This is done before the distance is calculated

Returns:

resultList **List:** Contains two lists. The Result List is a list of slot numbers and result for each of the reserved slots. The OCR Result List is of the same format as the return values of the OCRSlot function.

Example:

```
# Wait for 'TV Guide', waiting no more than 3 seconds
>>ret=WaitOCRMatch("TV Guide",[40,20,100,25],timeToWait=3)
>>print ret
>>[[[11, False]], [0, 0, 0, 0, [[11, '']]]]
>>ret=WaitOCRMatch("TV Guide",[40,20,100,25],timeToWait=3, slotNo=11)
>>print ret
>>[[[11, False]], [0, 0, 0, 0, '']]
```

Note:

1. If the *slotNo* parameter is omitted, the OCR is performed on all reserved slots.
   (a) *resultList* in the return value is a list of the results of all reserved slots
2. When defining the rectangular area you need to specify the top left hand corner and the width and height of the box, i.e. (x, y, w, h). Remember that the frame of reference is the native screen size not the streaming size (Standard Definition allows downscaling of the streaming video).
3. The statistics returned are:

(a) Total number of suspicious symbols.
(b) Total number of unrecognised symbols.
(c) Total number of all symbols.
(d) The total number of all words found.
(e) A list of strings[4] found per slot.

4. If the OCR fails to perform any OCR then None is returned.
5. Where the returned text contains one or more non-ASCII characters, the string will be returned as a unicode type, otherwise it will be a str type. Unicode will need to be encoded to UTF-8 if it is to be used with other StormTest APIs.

This function ignores the effect of SetVideoOffset when the connected server is an HD server. It accounts for the offset when the connected server is Standard Definition.
Where autoCorrect is specified, the returned string is *always* the value prior to auto correction being applied so that you can see the raw string.

2.12.2.3 `WaitOCRNoMatch` (**text**, **rect** = `None`, **timeToWait** = `5`, **waitGap** = `1`, **slotNo** = `0`, **maxDistance** = `0`, **autoCorrect** = `OCRStringCorrection.NoCorrection`)

Description

This function Waits until the specified slot fails the OCR comparison with the OCRed string found when OCR has been performed on a capture from a live stream. **Important:** When OCR is performed on the live stream, the resolution used is *always* **4CIF** for standard definition Servers. This means you must use a 704x576 reference image when calculating what co-ordinates to use for your rectangle. For HD, the resolution is automatically selected based on the input to the server.

Parameters:

← **text String** : Expected Text for comparison. In Release 3.1 and later this may be a NamedString.

← **rect List:** Restricts the OCR to the part of the screen bounded by the rectangular co-ordinates supplied. If no bounding rectangle is required, use *None* here. See note 1 for more information. In Release 3.1 and later this may be a NamedRegion.

← **timeToWait Integer:** Number of Seconds to wait for a fail

← **waitGap Float:** Interval between subsequent comparisons

← **slotNo Integer:** (optional) The slot number to capture strings on.

← **maxDistance Integer:** (optional) The maximum allowed Levenshtein distance between the expected text and the OCRed text. Wikipedia can give more details.

← **autoCorrect OCRStringCorrection:** (optional) The type of auto correction to be performed on input string. This is done before the distance is calculated

Returns:

resultList **List:** Contains two lists. The Result List is a list of slot numbers and result for each of the reserved slots. The OCR Result List is of the same format as the return values of the OCRSlot function.

Example:

```
# Wait for 'TV Guide', waiting no more than 3 seconds
>>ret=WaitOCRNoMatch("TV Guide",[40,20,100,25],timeToWait=3)
>>print ret
>>[[[11, True]], [0, 0, 0, 0, [[11, '']]]]
>>ret=WaitOCRNoMatch("TV Guide",[40,20,100,25],timeToWait=3, slotNo=11)
>>print ret
>>[[[11, True]], [0, 0, 0, 0, '']]
```

Note:

1. If the *slotNo* parameter is omitted, the OCR is performed on all reserved slots.
   (a) *resultList* in the return value is a list of the results of all reserved slots
2. When defining the rectangular area you need to specify the top left hand corner and the width and height of the box, i.e. (x, y, w, h). Remember that the frame of reference is the native screen size not the streaming size (Standard Definition allows downscaling of the streaming video).
3. The statistics returned are:

(a) Total number of suspicious symbols.
(b) Total number of unrecognised symbols.
(c) Total number of all symbols.
(d) The total number of all words found.
(e) A list of strings[4] found per slot.

4. When the OCR fails then None is returned.
5. Where the returned text contains one or more non-ASCII characters, the string will be returned as a unicode type, otherwise it will be a str type. Unicode will need to be encoded to UTF-8 if it is to be used with other StormTest APIs.

This function ignores the effect of SetVideoOffset when the connected server is an HD server. It accounts for the offset when the connected server is Standard Definition.
Where autoCorrect is specified, the returned string is *always* the value prior to auto correction being applied so that you can see the raw string.

2.12.2.4 `OCRFile` (**rect**, **fileName**, **languages** = `None`, **process** = `None`, **filters** = `None`)

Description

This function returns the strings found when OCR has been performed on an image file.

Parameters:

← **rect List:** Restricts the OCR to the part of the screen bounded by the rectangular co-ordinates supplied. If no bounding rectangle is required, use *None* here. See note 1 for more information. In Release 3.1 and later this may be a NamedRegion.

← **fileName String** or StormTestImageObject: The local filename to perform OCR on. This may also be the image returned from CaptureImageEx or similar function which produces a StormTestImageObject object. Types of image file supported are: BMP, JPEG, PNG and TIFF. The function can read GIF files but note that StormTest cannot save files as GIF. In Release 3.1 and later this may be a NamedImage.

← **languages List:** (optional) May be one language or a list of languages. Each language is a string of three characters from the ISO639-3 code list. For a list of supported languages, please see Languages supported by the OCR engine. Defaults to 'eng' - English.

← **process ProcessImage:** (optional) List of pre-processing options to be applied to the image before OCR is carried out. See note 2 for more information. Defaults to (ProcessImage.Invert)

← **filters List:** (optional) List of filters as per OCRSetImageFiltering()

Returns:

ocrResult **List:** Contains the OCR statistics and a string containing all characters found. See note 3 for more information.

Example:

```
# Perform OCR on a bitmap file
>>ret=OCRFile((70,70,300,300),"myOCRImage.bmp")
>>print ret
>>[0,0,15,4,'This is the string found in the bitmap']
```

Note:

1. When defining the rectangular area you need to specify the top left hand corner and the width and height of the box, i.e. (x, y, w, h).
2. The user may choose what pre-processing should be done to the image before it is sent to the OCR engine. The user may choose no pre-processing, one method or a combination of methods. Available method is:
   (a) Invert image (ProcessImage.Invert)
3. The statistics returned are:
   (a) Total number of suspicious symbols.
   (b) Total number of unrecognised symbols.
   (c) Total number of all symbols.
   (d) The total number of all words found.
   (e) A string 4 containing all the characters found in the bitmap

4. The returned text string is always returned as unicode. This will need to be encoded to UTF-8 if it is to be used with other StormTest APIs.
5. If a named image or rectangle is used then only the general 'all slot' path will be searched because this function does not take a slot number.

2.12.2.5 `OCRImage` (**image**, **rect**, **languages** = `None`, **process** = `None`, **filters** = `None`)

Description

This function returns the strings found when OCR has been performed on an image.

Parameters:

← **image StormTestImageObject** or PIL Image: The image to perform OCR on. It can be a NamedImage.

← **rect List:** Restricts the OCR to the part of the screen bounded by the rectangular co-ordinates supplied. If no bounding rectangle is required, use *None* here. See note [1] for more information. This may be a NamedRegion or a ScreenDefinition object.

← **languages List:** (optional) May be one language or a list of languages. Each language is a string of three characters from the ISO639-3 code list. For a list of supported languages, please see Languages supported by the OCR engine. Defaults to 'eng' - English.

← **process ProcessImage:** (optional) List of pre-processing options to be applied to the image before OCR is carried out. See note [2] for more information. Defaults to (ProcessImage.Invert)

← **filters List:** (optional) List of filters as per OCRSetImageFiltering()

Returns:

ocrResult **List:** Contains the OCR statistics and a string containing all characters found. See note [3] for more information.

Example:

```
# Perform OCR on an image
>>ret=OCRImage(img, (70,70,300,300))
>>print ret
>>[0,0,15,4,'This is the string found in the bitmap']
```

Note:

1. When defining the rectangular area you need to specify the top left hand corner and the width and height of the box, i.e. (x, y, w, h).
2. The user may choose what pre-processing should be done to the image before it is sent to the OCR engine. The user may choose no pre-processing, one method or a combination of methods. Available method is:
   (a) Invert image (ProcessImage.Invert)
3. The statistics returned are:
   (a) Total number of suspicious symbols.
   (b) Total number of unrecognised symbols.
   (c) Total number of all symbols.
   (d) The total number of all words found.
   (e) A string [4] containing all the characters found in the bitmap
4. The returned text string is always returned as unicode. This will need to be encoded to UTF-8 if it is to be used with other StormTest APIs.
5. If a named image or rectangle is used then only the general 'all slot' path will be searched because this function does not take a slot number.

6. If a ScreenDefinition object is used, the languages, process and filters parameters, if not None, override the values in the ScreenDefinition object. A copy of the ScreenDefinition object is returned instead of the string (eg [0,0,15,4,<ScreenDefinitionObject>])

2.12.2.6 `OCRGetRemainingChars` (**server** = `None`)

Description

This function returns an integer denoting the number of characters remaining on the OCR license for this period (i.e. until the end of the month)
By default, the OCR license details for the currently connected StormTest server are returned. However, if no server is connected, or if the details for a different server are required, the server name can be passed as a parameter.
This Function does not require a client license to be called.

Parameters:

← **server String:** (optional) Specify the StormTest server to get the remaining OCR characters from. If the server does not use the standard port, then the server name must include the port number (i.e. 'server:port')

Returns:

remChars **Integer:** The number of remaining characters.

Example:

```
>>ret = OCRGetRemainingChars()
>>print ret
>>9939020
```

2.12.2.7 `OCRSetLanguage` (**languages** = `'eng'`, **slotNo** = `0`)

Description

This function is used to set the default language dictionaries that are used by the OCR engine when it comes across suspicious characters in a word. The default setting for this is english - *'eng'*. It is still possible to OCR other languages based on the latin character set without changing the default language.
However, it is very important to set the default language correctly if you wish to OCR text that is rendered in non-latin alphabets. The user may set the OCR language to be a list of languages. The language is specified by using the **ISO639-3** encoding for that language. For a list of supported languages, please see Languages supported by the OCR engine
Please note that the language(s) set by a call to this function apply to all OCR operations carried out by the OCR engine on the specified slot. The language is stored on the server and will be used by later scripts unless OCRSetLanguage is called by the script. This function has no effect on the language used by OCRFile()

Parameters:

← **languages List:** May be one language or a list of languages. Each language is a string of three characters from the ISO639-3 code list.

← **slotNo Integer:** (optional) Apply the language change to OCRs performed on just one of the reserved slots

Returns:

result **Bool:** True if the call was successful. False otherwise. If the language code used is not supported or invalid, then a StormTestException will be raised.

Example:

```
# Tell the OCR engine to use the German dictionary.
>>result = OCRSetLanguage('deu')
>>print result
>>True

# Tell the OCR engine to use the Spanish, Portugese and French dictionaries.
>>result = OCRSetLanguage(('spa','por','fra'))
>>print result
>>True

# Tell the OCR engine to use an invalid dictionary - this will raise an exception
>>result = OCRSetLanguage('abc')
 Exception in user code:
 StormTestExceptions: Invalid language: [abc]
```

2.12.2.8 `OCRGetLanguage` **(slotNo** = 0**)**

Description

Get the default language dictionary used by the StormTest OCR engine for the reserved slots

Parameters:

← **slotNo Integer:** (optional) Get the default language dictionary for just one of the reserved slots

Returns:

result **List:** A list of lists consisting of slot number and a list of strings representing the language dictionaries in use. A single list will be returned if the exact slot number is given.

Example:

```
# Get dictionary, 2 slots reserved
>>result = OCRGetLanguage()
>>print result
>>[ [4, ['ces', 'dan', 'nld', 'deu']],
>>  [6, ['ces', 'dan', 'nld', 'deu']]]

# Get dictionary, specify slot
>>result = OCRGetLanguage(4)
>>print result
>>[4, ['ces', 'dan', 'nld', 'deu']]
```

2.12.2.9 `OCRSetDefaultImageProcessing` (**process** = `None`, **slotNo** = `0`)

Description

This function sets the default image pre-processing that will be applied to all images for the specifed slot before sending them to the OCR engine. From 3.0, only the Invert option is supported. This Invert is for legacy applications. New applications should disable this Invert and use OCRSetFilters(). However, since 1.0 of StormTest, the default has been to apply this Invert and to avoid breaking scripts, this behaviour is retained.

Parameters:

← **process ProcessImage:** List of pre-processing options to be applied to the image before OCR is carried out. See note [1] for more information.

← **slotNo Integer:** (optional) Apply the image pre-processing to OCRs performed on just one of the reserved slots

Returns:

result **Bool:** True if the call was successful. False if there was an error in the parameter.

Example:

```
# Tell the OCR engine to do no pre-processing of images by default.
>>result = OCRSetDefaultImageProcessing(None)
>>print result
>>True

# Tell the OCR engine to invert the colors using legacy method by default.
>>result = OCRSetDefaultImageProcessing(ProcessImage.Invert)
>>print result
>>True
```

Note:

1. The user may choose what pre-processing should be done to the image before it is sent to the OCR engine. The user may choose no pre-processing, one method or a combination of methods. Available method is:
   (a) Invert image (ProcessImage.Invert)

2.12.2.10 `OCRGetDefaultImageProcessing` **(slotNo** = 0**)**

Description

This function gets the default image pre-processing that will be applied to all images before sending them to the OCR engine.

Parameters:

← **slotNo Integer:** (optional) Get the default image pre-processing for just one of the reserved slots

Returns:

result **Tuple** or Integer: If zero or more than one pre-processing methods that will be applied then this is a tuple of each method that will be applied before each OCR. If only one method will be applied, then just that value as an integer is returned.

Example:

```
>>result = OCRGetDefaultImageProcessing()
>>print result
>>[ [4, 0],    # (ProcessImage.Invert)
    [5, 0],    # (ProcessImage.Invert)
    [6, ()]]   # No image pre-processing
```

2.12.2.11 OCRSetImageFiltering (filterList, slotNo = 0)

Description

This function sets the default image filtering that will be applied to all images before sending them to the OCR engine. This function has no effect on the image processing applied to images used by OCRFile().

Parameters:

← **filterList list** of string arrays: List of filtering options to be applied to the image before OCR is carried out. See note [1] for more information.

← **slotNo Integer:** (optional) Apply the image pre-processing to OCRs performed on just one of the reserved slots

Returns:

result **Bool:** True if the call was successful. False if there was an error in the parameter.

Example:

```
# Tell the OCR engine to do no filtering of images by default.
>>result = OCRSetImageFiltering(None)
>>print result
>>True

# Tell the OCR engine to blur the images on a 3x3 processing block and convert to grayscale
>>result = OCRSetImageFiltering([("Blur","Gaussian3x3"),("Color","Gray")])
>>print result
>>True
```

Note:

1. The user may choose what pre-processing should be done to the image for the specified slot before it is sent to the OCR engine. The user may choose no pre-processing, one method or a combination of methods. Each method is a tuple of 2 items, the first is a class of filtering and the second is a precise algorithm within the class to use.Available methods are:
   (a) ("Blur", "Gaussian3x3") - blur the image using a Gaussian blur algorithm on a 3x3 pixel matrix
   (b) ("Blur", "Gaussian5x5") - blur the image using a Gaussian blur algorithm on a 5x5 pixel matrix (blurs more than 3x3)
   (c) ("Sharpen", "Sharpen3x3") - sharpen the image using a 3x3 pixel matrix
   (d) ("Sharpen", "Sharpen5x5") - sharpen the image using a 5x5 pixel matrix (sharper than the 3x3)
   (e) ("Contrast", "Contrast1") - Increase the contrast a bit
   (f) ("Contrast", "Contrast2") - Increase the contrast a bit more
   (g) ("Contrast", "Contrast3") - Increase the contrast quite a bit
   (h) ("Contrast", "Contrast4") - Increase the contrast by quite a lot
   (i) ("Contrast", "Contrast5") - Increase the contrast by a lot
   (j) ("Invert", "Invert") - Invert all colors
   (k) ("Color", "Red") - keep just the red component of the color
   (l) ("Color", "Green") - keep just the green component of the color
   (m) ("Color", "Blue") - keep just the blue component of the color

(n) ("Color", "RedGreen") - keep the red and green components of the color
(o) ("Color", "RedBlue") - keep the red and blue components of the color
(p) ("Color", "BlueGreen") - keep the blue and green components of the color
(q) ("Color", "RedGray") - keep the red component and then make the image gray
(r) ("Color", "GreenGray") - keep the green component and them make the image gray
(s) ("Color", "BlueGray") - keep the blue component and then make the image gray
(t) ("Color", "RedGreenGray") - keep the red and green components and then make the image gray
(u) ("Color", "RedBlueGray") - keep the red and blue components and then make the image gray
(v) ("Color", "BlueGreenGray") - keep the blue and green components and then make the image gray
(w) ("Color", "Gray") - convert to gray
(x) ("Color", "Monochrome") - convert to black and white

2.12.2.12 `OCRGetImageFiltering` (**slotNo** = 0)

Description

This function gets the default image filtering that will be applied to all images before sending them to the OCR engine.

Parameters:

← **slotNo Integer:** (optional) Get the default image filtering for just one of the reserved slots

Returns:

result **List:** List of the filtering methods that will be applied by default before each OCR

Example:

```
>>result = OCRGetImageFiltering()
>>print result
>>[ [4, [["Sharpen","Sharpen5x5"]],    # (Sharpen image using a 5x5 area)
    [5, [["Blur","Gaussian3x3"]],      # (Blur image using a Gaussian 3x3 area)
    [6, []],                           # No image filtering
    [7, [["Color","RedGray"],["Sharpen","Sharpen3x3)]] # (Drop Blue and green components thus keeping Red, convert to
```

2.12.2.13 `CompareStrings` (**string1**, **string2**, **percent** = 90)

Description

This is a utility function that can be used in conjunction with OCRSlot() and OCRFile(). It allows a test to compare an expected string with the actual string returned by the OCR function and to get a Pass even when the strings do not match exactly.
Given that the results from OCR are not always 100% accurate depending on the quality of the incoming image, this function allows a test writer to tune the string comparison in a similar way that image comparisons are tuned.

Parameters:

← **string1 String:** A string

← **string2 String:** Another string

← **percent Integer:** How closely the two strings must match for a pass. Range is from 0-100, where 100 means the strings match exactly.

Returns:

compareResult **Bool:** True for a match, False otherwise

Example:

```
# Perform OCR on all reserved slots
>>ret = OCRSlot(None)
>>for slot,txt in ret[4]:
>>    if CompareStrings("Expected text",txt,85) == False:
>>        WriteDebugLine("OCR comparison failed")
>>    else:
>>        WriteDebugLine("OCR comparison passed")
```

2.12.2.14 `ComputeStringsDistance` (**string1, string2)**

Description

This is a utility function that can be used to compute the Levenshtein distance between two strings. It is used in conjunction with OCRSlot() and OCRFile() to allow for less than perfect OCR string matching

Parameters:

← **string1 String:** A string

← **string2 String:** Another string

Returns:

distance **Integer:** Levenshtein distance between the two strings. Wikipedia can give more details.

Example:

```
>>>print ComputeStringsDistance("TV Guide", "Guide")
>>>3
```

2.12.2.15 `StringsAutoCorrection` (**string1**, **type** = `OCRStringCorrection.NoCorrection`)

Description

This is a utility function that automatically correct a given string. Two methods are defined: 1, correct a given string to all characters, so all similar looking numbers are converted to their character counterparts 2, correct a given string to all numbers, so all similar looking characters are converted to their number counterparts

Parameters:

← **string1 String:** A string

← **type OCRStringCorrection:** AutoCorrection type

Returns:

correctedString **String:** The automatically corrected string

Example:

```
>>>print StringsAutoCorrection("Options", OCRStringCorrection.AllCharacters)
>>>Options
>>>print StringsAutoCorrection("56SI096AH", OCRStringCorrection.AllNumbers)
>>>5651096AH
```

2.13 High Speed Video Timing API

These commands control the high speed video timer API.

Functions

- ReserveVideoTimer
- FreeVideoTimer
- StartZapMeasurement
- TriggerHighSpeedTimer
- StopHighSpeedTimer
- GetZapTimes
- GetRawZapMeasurements
- EnableCaptureZapFrames
- GetCountImagesSaved
- GetRawZapFrame
- TriggerOsdMask
- StopOsdMask
- TriggerOsdString
- StopOsdString

2.13.1 Detailed Description

These commands control the high speed video timer 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 and can detect this to an accuracy of one frame (i.e. 40ms). Since the zap measurement is resource intensive, and would be adversely affected if too many simultaneous measurements were happening, the zap time measurement is treated as a reservable resource. The client is therefore required to reserve the resource before using it and release it once the measurement is complete. Note that the high speed timer cannot be reserved at the same time as running a reference stream video analysis on the slot. You can run high speed timer at the same time as VQA analysis (subject to memory constraints).

The memory constraints are discussed under Audio Video Analysis API

2.13.2 Function Documentation

2.13.2.1 `ReserveVideoTimer` (**slotNo** = 0)

Description

This function reserves a video timer resource for the specified slot. The slot must have been previously reserved via ReserveSlot(). There is a limit to the number of video timers that can be reserved at any one time (4 video timers per 16 slot rack for SD servers, no limit for HD servers), so the script writer must deal with the case where the function call returns a fail. Also, a high speed timer cannot be reserved while a reference stream Video analysis is running on the slot. You can reserve a high speed timer at the same time as VQA analysis (subject to memory constraints).

Parameters:

← **slotNo Integer:** (optional) The slot number on which to reserve a video timer.

Returns:

Bool: True if the video timer on the slot has been reserved, false otherwise.

Example:

```
>>>print ReserveVideoTimer()
>>>True
```

2.13.2.2 `FreeVideoTimer` **(slotNo** = 0**)**

Description

This function frees a previously reserved video timer on the specified slot.

Parameters:

← **slotNo Integer:** (optional) The slot number on which to free a video timer.

Returns:

Bool: True if the video timer has been freed, else False.

Example:

```
>>>print FreeVideoTimer()
>>>True
```

2.13.2.3 `StartZapMeasurement` (**irKey**, **keyCount** = `0`, **rect** = `None`, **colorCount** = `1000`, **timeout** = `20`, **slotNo** = `0`, **triggerSlot** = `None`)

Description

This function starts the Zap Time measurement. The slot must have had a video timer previously reserved via a call to ReserveVideoTimer() - this is because it is not possible to simultaneously run Zap time measurements on all slots of a 16 slot SD rack (HD systems do NOT have this limitation) - to ensure the Zap measurement can run when desired, the measurement resource must previously have been reserved - if necessary, the client can wait in a loop until it is able to reserver the video timer.

Parameters:

← **irKey String:** The ir key stroke which initiates the zap time measurement - the zap time is measured from the end of this keystroke. In Release 3.1 and later this may be a NamedString.

← **keyCount Integer:** (optional) 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. keyCount is only of use when you need to send multiple instances of irKey prior to starting the zap measurement as in the example above where the measurement should start **with** the second '1' key.

← **rect List:** (optional) The area of the video to examine when determining zap time. This is based on the standardised coordinates of a 704 x 576 screen, irrespective of current video. In Release 3.1 and later this may be a NamedRegion. streaming format. The default of None means all of the screen

← **colorCount Integer:** (optional) The threshold for determing 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.

← **timeout Number:** (optional) A timeout to wait for the zap to complete. A zap consists of 2 phases, the first phase waits for the color count to drop below the threshold and phase 2 waits for the color count to rise again. This timeout parameter is divided equally between the 2 phases. So, the default value of 20 allocates 10 seconds to wait for phase 1 and 10 seconds to wait for phase 2. If either time is exceeded, the call returns. So, if the DUT fails to start the zap process within timeout/2 seconds, the call stops and returns immediately. Phase 1 starts starts from the end of the irKey occurrence. There is also an overall timeout on the function from StartZapMeasurement() of 2 minutes for SD systems - to save on server resources. From release 3.2.5, HD systems have no limitation. So from calling StartZapMeasurement() to completion of Zap there is a maximum time limit of 2 minutes on SD systems, even if the StartZapMeasurement() timeout is set higher. From version 3.3.1 onwards, the timeout is measured in frame time not real time (so a timeout of 2 seconds with 50 fps will capture and process 100 frames). There is a safety feature such that if no frames are being processed, no more than 1 minute will elapse before the 'real' timeout kicks in.

← **slotNo Integer:** (optional) The slot number on which to start the zap measurement

← **triggerSlot Integer:** (optional) The slot number which acts as the trigger for the zap. This can be used if the script reserves 2 slots. In this case the trigger is monitored on the triggerSlot, **not** the slotNo but the zap timing and any associated images are acquired on slotNo. This is most useful when the zap timing is being used just as a simple means to capture all raw frames. TriggerHighSpeedTimer would be prefferred for use in that case.

The default value of 0 means trigger and acquire on the same slot. This is the default behaviour and emulates the behavior prior to release 2.8.3. triggerSlot is available only on release 2.8.3 and later.

Returns:

List: A list of tuples. Each tuple is one slot and is (slotNo,Bool) where the bool is True if successfully primed.

Example

```
>>if ReserveVideoTimer():
>>    PressDigits(101, slot)
>>    zapstatus = ST.StartZapMeasurement("Channel+",0,(0,0,704,300), timeout=5.0)
>>    WaitSec(1)
>>    # Do channel + and wait 5 seconds
>>    PressButton("Channel+")
>>    WaitSec(5)
>>    # get the zap times and validate them
>>    zaptimes = ST.GetZapTimes()
```

2.13.2.4 TriggerHighSpeedTimer (**irKey** = None, **keyCount** = 0, **timeout** = 20, **slotNo** = 0, **triggerSlot** = None)

Descripton

This function allows you to use the high speed timer for purposes other than measuring zap times. It sets up a trigger point and can be used instead of the StartZapMeasurement(). The timer is triggered and runs until the timeout or until stopped. This is only useful if the frame capture is also enabled using EnableCaptureZapFrames(). Then you can use GetRawZapFrame() to perform later analysis of the sequence. This could be used, for example, to capture 50 frames after pressing the 'TV_Guide' button and analyzing each frame to determine when the screen showed some text or a color in a particular place. GetZapTimes() has no meaning in this case.

Parameters:

← **irKey String:** (optional) The ir key stroke which initiates the timer. If omitted or set to None then the timer starts immediately. Timing starts from the end of the key stroke.

← **keyCount Integer:** (optional) 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 time start on the second '1' instead of the first. Ignored if irKey is None. Default 0.

← **timeout Number:** (optional) The total time in seconds to run the timer. The time starts from the irKey occurrence or immediately if irKey is None. There is also an overall timeout on the function from TriggerHighSpeedTimer() of 2 minutes on SD systems - to save on server resources. From release 3.2.5 onwards, HD systems have no limit as all slots can be used simultaneously. From version 3.3.1 onwards, the timeout is measured in frame time not real time (so a timeout of 2 seconds with 50 fps will capture and process 100 frames). There is a safety feature such that if no frames are being processed, no more than 1 minute will elapse before the 'real' timeout kicks in.

← **slotNo Integer:** (optional) The slot number on which to start the timer.

← **triggerSlot Integer:** (optional) The slot number which acts as the trigger for the timer. This can be used if the script reserves 2 slots and irKey is not None. In this case the trigger is monitored on the triggerSlot, **not** the slotNo but the the associated images are acquired on slotNo. This could be used, for instance when DUTs access some shared data and you wish to find the timing on slotNo from some event on the DUT in triggerSlot. triggerSlot is available only on release 2.8.3 and later. A value of 0 means the trigger and acquiring slot are the same slot. This is the default and emulates the behavior prior to release 2.8.3. If IrKey is None then triggerSlot is ignored.

Returns:

List: A List of tuples constaining slot number and boolean status. The boolean status will be true if the slot has successfully started the timer

Example:

```
>>if ReserveVideoTimer():
>>    EnableCaptureZapFrames(100,0,2)
>>    TriggerHighSpeedTimer("1",1)
>>    PressDigits(101)
>>    WaitSec(5)
>>    StopHighSpeedTimer()
>>    if GetCountImagesSaved(0)[0][1]>=11:
>>      (slot,image) = GetRawZapFrame(10)[0] # return 11th captured frame which is (11 * 3) * 40mS after the 2nd '1' k
>>      image.Save("frame11.png") # save it to disk.
```

Note:

The server captures images to memory and saves them asynchronously to disk so you need to call GetCountImagesSaved() before attempting to retrieve an image.

2.13.2.5 `StopHighSpeedTimer` (**slotNo** = 0)

Description

Stops the high speed timer. It is intended to be used with TriggerHighSpeedTimer() but could be used with StartZapMeasurement() - but not usually needed.

Parameters:

← **slotNo Integer:** (optional) The slot number on which to stop the timer.

Returns:

status **List:** List of tuples. Each tuple is (slot, status) where the status is True if the slot has successfully stopped the timer

Example:

```
>>>print StopHighSpeedTimer()
>>>[(12, True)]
>>>print StopHighSpeedTimer(12)
>>>[(12, True)]
```

2.13.2.6 `GetZapTimes` (**slotNo** = 0, **wait** = 0)

Description

This function returns the zap times if the zap has completed.

Parameters:

← **slotNo Integer:** (optional) The slot number on which to get the results.

← **wait Integer:** (optional) The time in seconds to wait for completion.

Returns:

List: Contains the Zap times, array of tuples, the 1st element is the slot number, the second is a bool indicating whether the zap was successful third is time (in seconds) to start of zap, fourth is time (in seconds) to end of zap. If zap is still in progress, then 3rd and 4th values are 0. If a timeout occurred, the 2nd parameter is False and the 3rd and 4th will reflect either the correct time or the timeout (depending upon whether the timeout occurred waiting for the zap start or zap end).

2.13.2.7 `GetRawZapMeasurements` (**slotNo** = 0)

Description

This function returns the raw measurements performed - this can be quite a lot as 25 measurements (30 on NTSC systems) are performed every second.

Parameters:

← **slotNo Integer:** (optional) The slot number on which to get the results.

Returns:

List: The return is a list of tuples, 1st is frame number relative to the irKey, second is the number of YUV colors in the rectangle of interest.

2.13.2.8 `EnableCaptureZapFrames` (**maxFrames** = 100, **startFrame** = 0, **skipCount** = 0, **slotNo** = 0)

Description

This function enables the zap timer code to capture the raw video frames during the measurement. These can be used for later analysis. They are captured on the server and can be retrieved using the GetRawZapFrame() function. They frames are overwritten by the next zap measurement on the same slot. So you can free the zap timer but so long as you do not free the Slot then you can still get the frames.

Parameters:

← **maxFrames Integer:** (optional) The max number of frames to capture. This is limited to 250 due to memory constraints (approx 10 seconds at 25 frames/sec). Default is 100.

← **startFrame Integer:** (optional) The frame to use as the first captured frame. Frame 0 is the first frame after the Zap starts. You can choose to ignore some frames by setting the value to be greater than zero. Less than zero is an error. Default is 0.

← **skipCount Integer:** (optional) The number of frames to skip between captures. A value of 0 will capture each frame, but a value of 1 will skip 1 frame between each and thus effectively capture at 1/2 the real frame rate (12.5 fps on a 25 fps PAL system). Useful to extend the elapsed time period of captured frames if it is not necessary to capture each one. Default 0.

← **slotNo Integer:** (optional) The slot number on which to capture the frames. The zap timer must be reserved on the slot prior to calling this value.

Returns:

List: The return is a list of tuples, 1st is slot number, 2nd is true/false

2.13.2.9 `GetCountImagesSaved` **(slotNo** = 0**)**

Description

This function returns the number of images actually saved if EnableCaptureZapFrames() is enabled This will be zero if the system is still saving them after the run has ended (there can be a delay). It must be called before calling FreeVideoTimer()

Parameters:

← **slotNo Integer:** (optional) The slot number for which the information is wanted.

Returns:

List: The return is a list of tuples, 1st is slot number, 2nd is number of images.

Note:

The High Speed Timer captures images coming from the hardware. For HD systems, it is possible for the Device Under Test to stop producing data on the HDMI link (eg when going into power save). This fact cannot be determined instantaneously so there is a delay of up to 10 seconds before the low level hardware concludes that there is no video from the hardware. This means that capturing frames as the DUT goes into power save will likely report fewer frames than you expect.

2.13.2.10 `GetRawZapFrame` (**frameIndex, slotNo** = 0)

Description

This function returns the raw video frame from a zap capture. It is an error to call this if the capture was not enabled prior to starting the zap.

Parameters:

← **frameIndex Integer:** The frame to retrieve. The 1st captured frame is frame 0. Note that if you specified a non zero skip in EnableCaptureZapFrames(), this is ignored in numbering the frame to be returned. Frame 1 will be the second frame captured - not necessarily the 2nd frame after the trigger point. The application code will need to do its own timing calculations.

← **slotNo Integer:** (optional) The slot number on which to get the frame.

Returns:

List: The retrun is a list of tuples, 1st is slot number, 2nd is the frame as a StormTestImageObject.

Note:

The server captures images to memory and saves them asynchronously to disk so you need to call GetCountImagesSaved() and, if necessary, wait for the images to be saved, before attempting to retrieve an image.

2.13.2.11 `TriggerOsdMask` (**rgb** = `(0,0,` **rect** = `(0,0,` **irKey** = `None`, **keyCount** = `0`, **autoStop** = `True`, **slotNo** = `0`)

Description

This function allows you to apply a mask on the video stream with a given color.

Parameters:

← **rgb Tuple:** (optional) The RGB color to use to mask the video. The default mask is pure black i.e. RGB(0,0,0)

← **rect List:** (optional) The area of the video to mask. The width must be multiple of 16. The height must be multiple of 8. The default rect is (0,0,32,32)

← **irKey String:** (optional) The ir key stroke which triggers the mask

← **keyCount Integer:** (optional) The number of irKey's to ignore before starting the mask. For example, setting irKey = '1' and keyCount = '1' would allow the client to send the ir command '101' and have time start on the second '1' instead of the first. Ignored if irKey is None. Default 0.

← **autoStop Boolean:** (optional) If set to True, the mask will only be applied to the current frame. Otherwise, the mask will be applied until StopOsdMask is called. The default is True, i.e. the mask will be automatically stopped.

← **slotNo Integer:** (optional) The slot number on which to apply the mask.

Returns:

List: Each element is a list of slot number, status. The status is true if the slot has successfully applied the mask. The return is always a list even if the slot number is explicitly given.

Example:

```
>>#the following example shows how TriggerOsdMask can be used with zap timing APIs, though it can be used independentl
>>if ReserveVideoTimer():
>>    EnableCaptureZapFrames(100,0,2)
>>    TriggerOsdMask((20,255,255), (16,16,48,48), "1", 1, True) # set a blue-ish color mask on the video upon the same
>>    TriggerHighSpeedTimer("1",1)
>>    PressDigits(101)
>>    WaitSec(5)
>>    StopHighSpeedTimer()
>>    (slot,image) = GetRawZapFrame(10)[0] # return 11th captured frame which is (11 * 3) * 40mS after the 2nd '1' key
>>    image.Save("frame11.png") # save it to disk.
```

2.13.2.12 `StopOsdMask` (**slotNo** = 0)

Description

Stops the OSD mask. It is intended to be used with TriggerOsdMask when its autoStop parameter is set to False

Parameters:

← **slotNo Integer:** (optional) The slot number on which to stop the OSD mask.

Returns:

List: Each element is a list of slot number, status. The status is true if the slot has successfully stopped the OSD mask. The return is always a list even if the slot number is explicitly given.

2.13.2.13 `TriggerOsdString` (**osdStr**, **x** = 0, **y** = 0, **irKey** = None, **keyCount** = 0, **autoStop** = True, **slotNo** = 0)

Description

This function allows you to display a string on the video.

Parameters:

← **osdStr String:** The string to display

← **x Integer:** x coordinate of the display position. x must be multiple of 8

← **y Integer:** y coordinate of the display position. y must be multiple of 4

← **irKey String:** (optional) The ir key stroke which triggers the OSD string

← **keyCount Integer:** (optional) The number of irKey's to ignore before starting the OSD string. For example, setting irKey = '1' and keyCount = '1' would allow the client to send the ir command '101' and have time start on the second '1' instead of the first. Ignored if irKey is None. Default 0.

← **autoStop Boolean:** (optional) If set to True, the OSD string will only be applied to the current frame. Otherwise, the string will be displayed until StopOsdString is called. The default is True, i.e. the OSD string will be automatically cleared

← **slotNo Integer:** (optional) The slot number on which to display the OSD string.

Returns:

Bool: true if the slot has successfully displayed the OSD string

Example

```
>>#the following example shows how TriggerOsdString can be used with zap timing APIs, though it can be used independen
>>if ReserveVideoTimer():
>>    EnableCaptureZapFrames(100,0,2)
>>    TriggerOsdString("Press 101 Done", 16,16, "1", 1, True)
>>    TriggerHighSpeedTimer("1",1)
>>    PressDigits(101)
>>    WaitSec(5)
>>    StopHighSpeedTimer()
>>    (slot,image) = GetRawZapFrame(10)[0] # return 11th captured frame which is (11 * 3) * 40mS after the 2nd '1' key
>>    image.Save("frame11.png") # save it to disk.
```

2.13.2.14 `StopOsdString` **(slotNo** = 0**)**

Description

Stops the OSD string. It is intended to be used with TriggerOsdString when its autoStop parameter is set to False

Parameters:

← **slotNo Integer:** (optional) The slot number on which to stop the OSD string.

Returns:

List of Lists. One List per slot reserved. The List per slot is [slot number, Bool] where the Bool is true if the slot has successfully stopped the OSD string

2.14 Feed Select API

These commands allow a test script to dynamically select the feed to a DUT in a particular slot.

Functions

- SelectInputFeed
- GetInputFeed

2.14.1 Detailed Description

These commands allow a test script to dynamically select the feed to a DUT in a particular slot.

Note that dynamic feed select is an optional feature on a StormTest rack and is not installed by default.

2.14.2 Function Documentation

2.14.2.1 `SelectInputFeed` (**feed**, **force** = `False`, **persistent** = `False`)

Description

This function is used to select an input feed for a slot. Note that the feeds to slots in a StormTest rack are generally arranged in groups. The normal configuration is that slots are grouped into groups of four. This means that switching the feed on one slot in a group will switch the feed for all slots in the group.
For this reason, the SelectInputFeed() function will only suceed if all slots in a group are reserved by the test script. If the tester wishes to over-ride this behaviour then he should use the **force** parameter to force a switch.

Parameters:

← **feed Integer:** Only 1 and 2 are valid values.

← **force Bool:** (optional) If set to True, then force the feed switch even if all slots in a group are not reserved by the test.

← **persistent Bool:** (optional) If set to False, then the feed to all slots reserved by the test will reset back to the default value when the test ends. If True, the setting will persist across multiple test runs.

Returns:

Integer/List: Returns 0 if the rack does not support this functionality. Otherwise returns a list of tuples, with one for each slot and a True/False result

Example:

```
>># Functionality not supported
>>print SelectInputFeed(1)
>>0

>># Slots 1,2,3 and 4 are reserved
>>print SelectInputFeed(1)
>>[[1,True], [2,True], [3,True], [4,True]]

>># Slots 1,2 and 3 are reserved
>>print SelectInputFeed(1)
>>[[1,False], [2,False], [3,False]] # Fails because one of the group is not reserved

>># Force the switch
>>print SelectInputFeed(1,True)
>>[[1,True], [2,True], [3,True]]
```

2.14.2.2 `GetInputFeed` ()

Description

This function returns the currently selected input feed for each reserved slot

Parameters:

None

Returns:

feedStatus **List:** A list of lists. For each slot, the slot number and the feed number is returned.

Example:

```
>># Slots 1-6 are reserved
>># Force all slots to feed 1
>>print SelectInputFeed(1,force = True)
>>[[1,True], [2,True], [3,True], [4,True], [5,True], [6,True]]
>>
>># Select feed 2, but don't force if all slots in a group are not reserved
>>print SelectInputFeed(2)
>>[[1,True], [2,True], [3,True], [4,True], [5,False], [6,False]]
>>
>># Now slots 1-4 will be on feed 2 and slots 5-6 on feed 1
>>print GetInputFeed()
>>[[1,2], [2,2], [3,2], [4,2], [6,1], [6,1]]
```

2.15 Audio/Video Switch API

These commands allow a test script to dynamically select the audio and video source that is captured by StormTest.

Enums

- enum AudioChannel

Functions

- SetAVInput
- GetAVInput
- GetCapsVideoInput
- GetCapsAudioInput
- GetRfFormats
- GetRfTunerConfiguration
- ConfigureRfTuner
- ConfigureHdDownConverter

2.15.1 Detailed Description

These commands allow a test script to dynamically select the audio and video source that is captured by StormTest.

Note that dymamic audio/video switching is an optional feature on a StormTest rack that requires extra hardware and it is not installed by default.

2.15.2 Function Documentation

2.15.2.1 `SetAVInput` **(videoSource, audioSource, audioChannel, slotNo** = 0**)**

Description

This function sets the video and audio input to StormTest. This function is only valid for StormTest racks that have been equipped with an audio/video switch device that allows the test script to select which of the a/v inputs to use.

Parameters:

← **videoSource string** : The video source to use.

← **audioSource string** : The audio source to use.

← **audioChannel AudioChannel** : The audio channel to use. StormTest only support mono audio, so the test can use this to switch between left and right audio channels.

← **slotNo Integer:** (optional) The slot number to set the AV input on

Returns:

List: First entry indicates if the call suceeded. The second entry is a message. Third entry is a list of slots the command was enacted on.

Example:

```
>>ret = SetAVInput("cvbs1", "analog1", AudioChannel.Left, 1)  # Set AV input for slot 1
>>print ret
>>[1, 'OK', [1]]
or
>>ret = SetAVInput("cvbs1", "analog1", AudioChannel.Left)   # Sets AV input for all reserved slots
>>print ret
>>[1, 'OK', [1, 2]]

>># Using the command when no AV switch is configured
>>ret = SetAVInput("cvbs1", "analog1", AudioChannel.Left)
>>print ret
>>[0, 'No AV Switch Available', [1, 2]]
```

Note:

If the *slotNo* parameter is omitted the default is to set AV input for all slots reserved

2.15.2.2 `GetAVInput` (**slotNo** = 0)

Description

This function returns the AV input

Parameters:

← **slotNo Integer:** (optional) Allows the user to get the AV input on just one slot rather than all slots reserved.

Returns:

Array: An array of array consisting of slot number and AV input. 0 is returned if there is no AV switch configured in the rack.

Example:

```
# Get AV inputs of all reserved slots
>>ret=GetAVInput()
>>print ret
>>[ [1, ['cvbs1', 'analog', 'Left']],
>>  [3, ['cvbs1', 'analog', 'Left']] ]

# Get AV input of one specific slot
>>ret=GetAVInput(3)
>>print ret
>>['cvbs1', 'analog', 'Left']
```

Note:

If the *slotNo* parameter is omitted the AV inputs returned are for all slots reserved.

2.15.2.3 GetCapsVideoInput (slotNo = 0)

Description

This function returns the available video inputs that can be passed to the SetAVInput() function.

Parameters:

← **slotNo Integer:** (optional) Allows the user to get the available video inputs on just one slot rather than all slots reserved.

Returns:

Array: An array of tuples consisting of slot number and available video inputs. 0 is returned if there is no AV switch configured in the rack.

Example:

```
# Get available video inputs of all reserved slots
>>ret=GetCapsVideoInput()
>>print ret
>>[[1, ['cvbs1', 'cvbs2', 'cvbs3', 'cvbs4', 'svideo', 'ypbpr', 'hdmi', 'dvi', 'vga', 'vrf1', 'vrf2']],
>>[3, ['cvbs1', 'cvbs2', 'cvbs3', 'cvbs4', 'svideo', 'ypbpr', 'hdmi', 'dvi', 'vga', 'vrf1', 'vrf2']]]

# Get available video inputs of one specific slot
>>ret=GetCapsVideoInput(3)
>>print ret
>>['analog1', 'analog2', 'analog3', 'analog4', 'analog5', 'hdmi', 'optical', 'coax', 'arf1', 'arf2']
```

Note:

If the *slotNo* parameter is omitted the available video inputs returned are for all slots reserved.

2.15.2.4 `GetCapsAudioInput` (**videoSource**, **slotNo** = 0)

Description

This function returns the available audio inputs for a given video input

Parameters:

← **slotNo Integer:** (optional) Allows the user to get the available audio inputs on just one slot rather than all slots reserved.

← **videoSource String:** The video source for which the audio capabilies are wanted.

Returns:

Array: An array of tuples consisting of slot number and available audio inputs. 0 is returned if there is no AV switch configured in the rack.

Example:

```
# Get available audio inputs of all reserved slots
>>ret=GetCapsAudioInput(VideoSource.Scart1CVBS)
>>print ret
>>[[1, ['analog1', 'analog2', 'analog3', 'analog4', 'analog5', 'hdmi', 'optical','coax', 'arf1', 'arf2']],
>>[3, ['analog1', 'analog2', 'analog3', 'analog4', 'analog5', 'hdmi', 'optical','coax', 'arf1', 'arf2']]]

# Get available audio inputs of one specific slot
>>ret=GetCapsAudioInput('cvbs1', 3)
>>print ret
>>['analog1', 'analog2', 'analog3', 'analog4', 'analog5', 'hdmi', 'optical', 'coax', 'arf1', 'arf2']
```

Note:

If the *slotNo* parameter is omitted the available audio inputs returned are for all slots reserved.

2.15.2.5 `GetRfFormats` (**tunerNum**, **slotNo** = 0)

Description

This function returns the available RF formats for a given tuner on a particular av switch

Parameters:

← **tunerNum Integer:** Specifies the tuner to query in the avswitch. This value can be 1 or 2.

← **slotNo Integer:** (optional) Allows the user to get the available RF formats on just one slot rather than all slots reserved.

Returns:

Array: An array of tuples consisting of slot number and available RFformats. 0 is returned if there is no AV switch configured in the rack.

Example:

```
# Get available rf formats from tuner 1 of all reserved slots
>>ret=GetRfFormats(1)
>>print ret
>>[[1, ['pal_secam_i', 'ntsc', 'pal_secam_bg', 'pal_secam_dk', 'pal_secam_l', 'pal_secam_lp']]]

# Get available rf formats from tuner 1 of one specific slot
>>ret=GetRfFormats(1, 1)
>>print ret
>> ['pal_secam_i', 'ntsc', 'pal_secam_bg', 'pal_secam_dk', 'pal_secam_l', 'pal_secam_lp']
```

Note:

If the *slotNo* parameter is omitted the available RF formats returned are for all slots reserved.

2.15.2.6 `GetRfTunerConfiguration` (**tunerNum**, **slotNo** = 0)

Description

This function returns the RF tuner configuration for a given tuner on a particular av switch

Parameters:

← **tunerNum Integer:** Specifies the tuner to query in the avswitch. This value can be 1 or 2.

← **slotNo Integer:** (optional) Allows the user to get the RF tuner configuration on just one slot rather than all slots reserved.

Returns:

Array: An array of tuples consisting of slot number and a tuple of frequency in kHz and format. 0 is returned if there is no AV switch configured in the rack.

Example:

```
# Get available RF configuration from tuner 1 of all reserved slots
>>ret=GetRfTunerConfiguration(1)
>>print ret
>>[[1, ['602000', 'ntsc']]]

# Get available RF configuration from tuner 1 from one specific slot
>>ret=GetRfTunerConfiguration(1, 1)
>>print ret
>> ['602000', 'ntsc']
```

Note:

If the *slotNo* parameter is omitted the available RF formats returned are for all slots reserved.

2.15.2.7 `ConfigureRfTuner` (**tunerNum, format, frequency, slotNo** = 0)

Description

This function configures the specified RF tuner on a particular av switch

Parameters:

← **tunerNum Integer:** Specifies the tuner to configure in the avswitch. This value can be 1 or 2.

← **format String:** A valid rf tuner format

← **frequency Integer:** A frequency in the supported frequency range in kHz

← **slotNo Integer:** (optional) Allows the user to configure the RF tuner on just one slot rather than all slots reserved.

Returns:

Array: First entry indicates if the call suceeded. The second entry is a message. Third entry is a list of slots the command was enacted on.

Example:

```
# Configure RF tuner 1 for all reserved slots
>>ret=ConfigureRfTuner(1,"pal_secam_i",602000)
>>print ret
>>[1, 'OK', [1,2]]

# Configure RF tuner 1 for of one specific slot
>>ret=ConfigureRfTuner(1,"pal_secam_i",602000,1)
>>print ret
>>[1, 'OK', [1]]
```

Note:

If the *slotNo* parameter is omitted the available RF formats returned are for all slots reserved.

2.15.2.8 `ConfigureHdDownConverter` (**format**, **slotNo** = 0)

Description

This function configures the HD down converter on a particular av switch

Parameters:

← **format** **String:** A valid HD format, which can be "pal" or "ntsc"

← **slotNo** **Integer:** (optional) Allows the user to configure the HD down converter on just one slot rather than all slots reserved.

Returns:

Array: First entry indicates if the call suceeded. The second entry is a message. Third entry is a list of slots the command was enacted on.

Example:

```
# Get available rf formats from tuner 1 of all reserved slots
>>ret=ConfigureHdDownConverter("pal")
>>print ret
>>[1, 'OK', [1,2]]

# Get available audio inputs of one specific slot
>>ret=ConfigureHdDownConverter("pal",1)
>>print ret
>>[1, 'OK', [1]]
```

Note:

If the *slotNo* parameter is omitted the available RF formats returned are for all slots reserved.

2.16 Telephone Line Simulator API

These commands allow a test script to control a Telephone Line Simulator (TLS) device.

Enums

- enum OffHookMode
- enum TLSStatus

Functions

- SetTLSOffHookMode
- GetTLSOffHookMode
- RetrieveTLSStatus
- RetrieveTLSLastDialed
- ClearTLSLastDialed

2.16.1 Detailed Description

These commands allow a test script to control a Telephone Line Simulator (TLS) device.

Note that the telephone line simulator is an optional extra on StormTest racks and is not available by default.

2.16.2 Function Documentation

2.16.2.1 `SetTLSOffHookMode` (**offHookMode**, **slotNo** = 0)

Description

This function sets the off-hook mode for a port on the telephone line simulator device.

Parameters:

← **offHookMode OffHookMode:** the mode of the off hook simulation

← **slotNo Integer:** (optional) The slot number to set OffHookMode on

Returns:

List: First entry indicates if the call suceeded. The second entry is a message. Third entry is a list of slots the command was enacted on.

Example:

```
>>SetTLSOffHookMode(OffHookMode.DialTone, 1)  # Set to present dial tone for slot 1
or
>>SetTLSOffHookMode(OffHookMode.DialTone)   # Set to present dial tone for all reserved slots
```

Note:

If the *slotNo* parameter is omitted the default is to operate on all slots reserved

2.16.2.2 GetTLSOffHookMode (**slotNo** = 0)

Description

This function returns the off-hook mode that the TLS ports are configured with.

Parameters:

← **slotNo Integer:** (optional) Allows the user to get the off-hook mode on just one slot rather than all slots reserved.

Returns:

Array: An array of arrays consisting of slot number and the off-hook mode.

Example:

```
# Get the off-hook mode of all reserved slots
>>ret=GetTLSOffHookMode()
>>print ret
>>[[1, 'DialTone'], [3, 'DialTone']]

# Get the off-hook mode of one specific slot
>>ret=GetTLSOffHookMode(3)
>>print ret
>>DialTone
```

Note:

If the *slotNo* parameter is omitted the default is to operate on all slots reserved.

2.16.2.3 `RetrieveTLSStatus` **(slotNo** = 0**)**

Description

This function returns the TLS status

Parameters:

← **slotNo Integer:** (optional) Allows the user to get the TLS status on just one slot rather than all slots reserved.

Returns:

Array: An array of arrays consisting of slot number and the TLS status.

Example:

```
# Get the TLS status of all reserved slots
>>ret=RetrieveTLSStatus()
>>print ret
>>[[1, 'OffHook'], [3, 'OnHook']]

# Get the TLS status of one specific slot
>>ret=RetrieveTLSStatus(3)
>>print ret
>>OnHook
```

Note:

If the *slotNo* parameter is omitted the default is to operate on all slots reserved.

2.16.2.4 `RetrieveTLSLastDialed` (**slotNo** = 0)

Description

This function returns the DTMF numbers received since the last off-hook transition.

Parameters:

← **slotNo Integer:** (optional) Allows the user to get the last DTMF dialed on just one slot rather than all slots reserved.

Returns:

Array: An array of arrays consisting of slot number and the last DTMF dialed.

Example:

```
# Get the last DTMF dialed of all reserved slots
>>ret=RetrieveTLSLastDialed()
>>print ret
>>[[1, "12345#"], [3, "323232"]]

# Get the last DTMF dialed of one specific slot
>>ret=RetrieveTLSLastDialed(3)
>>print ret
>>"323232"
```

Note:

If the *slotNo* parameter is omitted the default is to operate on all slots reserved.

2.16.2.5 `ClearTLSLastDialed` (**slotNo** = 0)

Description

This function clears the DTMF numbers received since the last off-hook transition.

Parameters:

← **slotNo Integer:** (optional) Allows the user to clear the last DTMF dialed on just one slot rather than all slots reserved.

Returns:

None

Example:

```
# Clear the last DTMF dialed of all reserved slots
>>ClearTLSLastDialed()

# Clear the last DTMF dialed of one specific slot
>>ClearTLSLastDialed(3)
```

Note:

If the *slotNo* parameter is omitted the default is to operate on all slots reserved.

2.17 Licensing API

This section contains the functions that allow a user to check the StormTest licensing status.

Functions

- StormTestLicenseDetails
- OCRLicenseDetails

2.17.1 Detailed Description

This section contains the functions that allow a user to check the StormTest licensing status.

StormTest requires a number of licenses to be fully functional. Every StormTest server must have a valid StormTest server license. Similarly, every StormTest Client must have a valid client license. Note that a client can be a running test script or the StormTest remote control application. Basically any process that attempts to reserve a slot on a StormTest server will require a client license.

Client licenses are served out from a pool and one license is consumed every time a new user connects to a StormTest server by calling ConnectToServer(). A new user is defined as a username@hostname. This means that one user can run as many tests as they want on one PC and they will use only one client license. However, they will need another client license to run tests on a different PC with the same username. In addition, in order to use the OCR functionality, at least one StormTest server in the test facility must have an OCR license. The OCR engine is licensed for a certain number of characters per month. This level can be adjusted according to an individual customers demand.

2.17.2 Function Documentation

2.17.2.1 `StormTestLicenseDetails` **(server** = `None`**)**

Description

This function returns a multi-line string containing details on the following

- Server host id
- Total number of server licenses
- Number of server licenses available
- List of server license users
- Total number of client licenses
- Number of client licenses available
- List of client license users By default, the license details are retrieved via the currently connected StormTest server. However, if no server is connected, or if a server name is passed as a parameter, the details will be retrieved via this server.

Parameters:

← **server String:** (optional) Specify the StormTest server to get the license details from. If the server does not use the standard port, then the server name must include the port number (i.e. 'server:port')

Returns:

licDetails **String:** The StormTest license details.

Example:

```
>>ret = StormTestLicenseDetails()
>>print ret
>>License Status for server server1 :
>>------------------------------------------------------------
>>Server host id: "00xxyyzzaabb"
>>
>>------------------------------------------------------------
>>Number of StormTest Server licenses:     5
>>Available StormTest Server licenses:     4
>>
>>StormTest Server Users:
>>    Administrator@server1
>>
>>------------------------------------------------------------
>>Number of StormTest Client licenses:     5
>>Available StormTest Client licenses:     5
>>
>>StormTest Client Users:
>>    None
>>------------------------------------------------------------
```

2.17.2.2 `OCRLicenseDetails` (**server** = `None`)

Description

This function returns a colon separated string of details relating to the installations OCR license. Each StormTest installation is licensed to allow up to a set maximum number of characters or pages to be OCRed each period (normally a period is a month). Details are given here of the number of units remaining in the current period that can be OCRed. Once the number drops to zero, OCR functionality will not be available again until a new period begins or an increased allowance is purchased.
By default, the OCR license details for the currently connected StormTest server are returned. However, if no server is connected, or if the details for a different server are required, the server name can be passed as a parameter.
This Function does not require a client license to be called.

Parameters:

← **server String:** (optional) Specify the StormTest server to get OCR license details from. If the server does not use the standard port, then the server name must include the port number (i.e. 'server:port')

Returns:

licDetails **String:** The license details for the OCR engine.

Example:

```
>>ret = OCRLicenseDetails()
>>print ret
>>Available characters per month 10000000: Remaining characters 9925335
```

2.18 StormTest Database access API

These functions allow the user to access the StormTest database, retrieving and setting the fields relating to the devices under test.

Functions

- SetMasterConfigurationServer
- CreateDutInstanceInSlot
- UpdateDutInstance
- UpdateDutInstanceInSlot
- GetDutInstanceInSlot
- GetDutDetails
- GetSoftwareComponentsSupported
- AddSwComponentToDut
- GetSwComponentFromDut
- GetAllSmartcards
- SetSmartcardNumber
- GetSmartcardNumber
- SetSmartcardAttribute
- GetSmartcardAttribute
- GetHwComponents
- FindModelsByHardware
- CreateDutModel
- UpdateDutModel
- DeleteDutModel
- ListDutModels

2.18.1 Detailed Description

These functions allow the user to access the StormTest database, retrieving and setting the fields relating to the devices under test.

NOTE: All these functions require the SetMasterConfigurationServer() call to have been made prior to their use. Failure to do this will result in a Stormtest exception being raised.

2.18.2 Function Documentation

2.18.2.1 `SetMasterConfigurationServer` **(serverAddress, cfgPort** = "8001"**)**

Description

This function sets the machine address of the master StormTest configuration server. Optionally the port used for the configuration server can be specified also. The configuration server must be set if the user wishes to submit jobs for remote execution on any StormTest server in the facility or to access the StormTest database.

Note:

If used, then SetMasterConfigurationServer must be called before ConnectToServer().

Parameters:

← **serverAddress String:** The address of the server

← **cfgPort String:** (optional) The port to use for the configuration server - unless the server set up is unusual, don't specify this and leave the default as 8001.

Returns:

None

Example:

```
>>SetMasterConfigurationServer("master_server")
```

2.18.2.2 `CreateDutInstanceInSlot` (slotId, name, description, serialNumber, modelName, modelType, hwVersionMajor, hwVersionMinor, fCreateHwVersionIfNeeded = `True`)

Description

This function is used to create an entry in the database for a device under test (DUT) i.e. a DUT.

Parameters:

← **slotId Integer** : unique facility wide slotId

← **name String** : name to give to the DUT (can be "" but not *None*)

← **description String** : description to give to the DUT (can be "" but not *None*)

← **serialNumber String** : unique serial number of DUT - must be unique within a modelName / modelType

← **modelName String** : name of the DUT model within the facility (must already exist in database)

← **modelType String** : name of the type of the DUT model within the facility (must already exist in database)

← **hwVersionMajor String** : Major hw version of the DUT as a string (need not already exist - see fCreateHwVersionIfNeeded)

← **hwVersionMinor String** : Minor hw version of the DUT as a string (need not already exist - see fCreateHwVersionIfNeeded)

← **fCreateHwVersionIfNeeded Bool** : (optional) If false, creation of dut instance will fail unless model,type and hw versions already exist. If true then if either major or minor hw versions do not exist a new model/type with the specified hw versions will be created (all other attributes of the model/type will be cloned). It is a requirement that at least one model with valid hw versions is created manually in the database before doing this.

Returns:

idOfDut **String:** The id of the newly created dut.

Example:

```
>>CreateDutInstanceInSlot(99945, "my dut","A new dut", "899a", "DutType","1","34")
```

2.18.2.3 UpdateDutInstance (**dutId**, **name** = None, **description** = None, **serial_number** = None, **ipaddress** = None)

Description

This function is used to update a DUT entry in the database

Parameters:

← **dutId Integer** : unique facility wide DUT id to be updated

← **name String** : name to be given. None for no change

← **description String** : description for the DUT. None for no change

← **serial_number String** : serial number for the DUT. None for no change

← **ipaddress String** : ip address of the DUT. None for no change

Returns:

None

Example:

```
>>UpdateDutInstance(45, "my dut")
```

2.18.2.4 `UpdateDutInstanceInSlot` (**server**, **slotNo**, **name** = `None`, **description** = `None`, **serial_number** = `None`, **ipaddress** = `None`)

Description

This function is used to update a DUT entry in the database

Parameters:

← **server String** : server name where DUT is.

← **slotNo Integer** : slot no within the server.

← **name String** : name to be given. None for no change

← **description String** : description for the DUT. None for no change

← **serial_number String** : serial number for the DUT. None for no change

← **ipaddress String** : ip address of the DUT. None for no change

Returns:

None

Example:

```
>>UpdateDutInstanceInSlot("server", 1, "my dut",ipaddress="192.168.1.16")
```

2.18.2.5 `GetDutInstanceInSlot` (**slotNo** = `0`, **fields** = `None`)

Description

This function gets information about the DUT in the specified slot.

Parameters:

← **slotNo Integer** : (optional) The slot number in the rack that the DUT is located in.

← **fields Integer** : (optional) List of fields to return. If omitted then all fields bar ipaddress and modelId are returned (as per 1.4 version). valid fields names are: name, description, modelName, modelType, hwVersionMajor, hwVersionMinor, serialNumber, ipaddress, modelId

Returns:

Array : If *slotNo* is 0 then return is an array of tuples else just one tuple is returned.

contents are (slotNumber, slotId, name, description, serialNumber, modelName, modelType, hwVersionMajor, hwVersionMinor) as per CreateDutInstanceInSlot() function parameter definition. If fields is given then the subset specified is returned. in the order listed in fields

2.18.2.6 `GetDutDetails` (dutId)

Description

This function gets information about the DUT of the specified DUT id.

Parameters:

← **dutId Integer** : The DUT id.

Returns:

Dictionary : Dictionary of fields->values. None on an error. Contents vary by version but so far the list is: slotid (NULL on DUT not in a slot), dutid, name, description, serialnumber, ipaddress, modelname, modeldescription, manufacturer, manfmodelname, hwversion, minorhwversion, modeltype, modeltypedescription, network

2.18.2.7 `GetSoftwareComponentsSupported` (**componentName, slotNo** = 0)

Description

This function gets information for the device in the slot. The collections and components and the supported status must be added manually outside of this API.

Parameters:

← **componentName String** : Name of a component to search for (case sensitive). Use empty string for all components.

← **slotNo Integer** : (optional) By default, data for all reserved slots is returned, unless this parameter is used.

Returns:

Array : The return value is an array of tuples (if *slotNo=0*) or a single tuple (if *slotNo!=0*). Each tuple is (slotNo, array of components). Each component is an array: [componentName, Description, version, collectionName, collectionDescription, status]

2.18.2.8 `AddSwComponentToDut` (**componentName, collectionName, version, description, SlotNo** = `0`, **fCreateVersionIfNeeded** = `True`)

Description

This function adds a specific software component to the device in the slot.

Parameters:

← **componentName String** : Name of component to add (must already exist in the database).

← **collectionName String** : Name of collection where the component is (must already exist in database).

← **version String** : Version of component (need not exist if fCreateVersionIfNeeded is True).

← **description String** : Description of component (may be empty, in which case the value of a newly created component is taken from database of an existing component).

← **SlotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

← **fCreateVersionIfNeeded Bool** : (optional) Flag to create the version if it is not in database. When this is false the call will fail if version is not in database.

Returns:

Tuple : A list containing (status, slot list)

2.18.2.9 `GetSwComponentFromDut` (**componentName, collectionName, slotNo** = 0)

Description

This function returns a list of components of the actual DUT in the slot (not those that are supported, but those applied to it).

Parameters:

← **componentName String** : Name of component to return. Use "" for all components.

← **collectionName String** : Collection to use to filter components. Use "" for all collections.

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

Array : An array of tuples (if *slotNo=0*) or a single tuple (if *slotNo!=0*). Each tuple is (slotNo, array of components). Each component is [componentName, version, description, collectionName]

2.18.2.10 `GetAllSmartcards` ()

Description

This function returns a list of all smartcards in the faciltiy. Just the numbers are returned.

Parameters:

None

Returns:

Array : An array smartcard IDs.

2.18.2.11 `SetSmartcardNumber` (**smartcardNumber**, **slotNo** = `0`, **fCreateIfNeeded** = `True`)

Description

This function sets the smartcard number for the DUT in the slot.

Parameters:

← **smartcardNumber String** : Unique card number (as a string). Must be unique facility wide.

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

← **fCreateIfNeeded Bool** : (optional) If true and smartcard does not exist, it will be created.

Returns:

Tuple : A tuple containing (status, slot list)

2.18.2.12 `GetSmartcardNumber` **(slotNo** = 0**)**

Description

This function gets the smartcard number for the DUT in the slot.

Parameters:

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

Array : An array of tuples (if *slotNo=0*) or a single tuple (if *slotNo!=0*). Each tuple contains (slotNo, smartcardNumber).

2.18.2.13 `SetSmartcardAttribute` (**attributeName, attributeValue, slotNo** = 0)

Description

This function sets the smartcard attribute for the smartcard for the DUT in the slot.

Parameters:

← **attributeName String** : Name of attribute to set.

← **attributeValue String** : Value of attribute to set.

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

Tuple : A tuple containing (status, slot list)

2.18.2.14 `GetSmartcardAttribute` **(attributeName, slotNo** = 0**)**

Description

This function gets the smartcard attribute for the DUT in the slot.

Parameters:

← **attributeName String** : Name of attribute to get.

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

Array : An array of tuples (if *slotNo=0*) or a single tuple (if *slotNo!=0*). Each tuple contains (slotNo, attributeValue).

2.18.2.15 `GetHwComponents` (**componentName** = `""`, **componentType** = `""`, **slotNo** = `0`)

Description

This function gets the list of components of a specified name and type.

Parameters:

← **componentName String** : (optional) Name of component to search for, "" for all components. You can use the '∗' wildcard.

← **componentType String** : (optional) Type name of component to search for, "" for all component types. You can use the '∗' wildcard.

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

Array : An array of tuples (if *slotNo=0*) or a single tuple (if *slotNo!=0*). Each tuple is (slotNo, array of components). Each component is [componentName, componentType, description, manufacturer, chipset, parameters]. For many components, the manufacturer, chipset and parameters will be empty strings.

2.18.2.16 `FindModelsByHardware` (**componentName**, **componentType** = "", **componentCount** = 1)

Description

This function finds a list of models which have the specified hardware components.

Parameters:

← **componentName String** : Name of component to search for. You can use the '∗' wildcard.

← **componentType String** : (optional) Type name of component to search for, "" for all component types. You can use the '∗' wildcard.

← **componentCount Integer** : (optional) The minimum number of components that the model must support.

Returns:

Array : An array of tuples, one per model that matches the criteria. Each tuple is [modelName, Manufacturer, HwVersion] and is thus suitable for passing to ReserveDut() function call (ignoring the serial number / smartcard values)

Example:

```
>># This will find all models with 4 or more hardware components of type AVOut.
>>ret = FindModelsByHardware("","AVOut",4)
```

2.18.2.17 `CreateDutModel` (**modelName, modelType, irDatabase, detailedInformation, permitDuplicateNames** = `False`)

Description

This function creates a new model in the database which can subsequently be used for creating dut instances. The Admin Console provides the same functionality in a GUI manner.

Parameters:

← **modelName String** : The name of the model to create. This is case sensitive

← **modelType String** : The type of the model to create. If such a type does not exist, it will be created. This value is case sensitive so it is possible to end up with types called 'zapper' and 'Zapper'.

← **irDatabase String** : The IR database to use for this model. This must have already been created using the IR Trainer applet in the Admin Tools. This value is case sensitive so must match exactly the value used in the IR Trainer.

← **detailedInformation** @ Dictionary of strings : A dictionary of extra information about the model in a key value manner. This allows future expansion of the range of information stored without breaking existing scripts. The keys are NOT case sensitive but the values are. Possible keys are: Description (a description of the model), Manufacturer (the name of the model's manufacturer), ManufacturerModel (the model name given by the manufacturer to the model being created), MajorHwVersion (the major version string of the model), MinorHwVersion (the minor version string of the model), Broadcaster (the broadcaster that the model is tied to), VideoTransform (the transformation to apply to the video between design to world coordinate space. Must be a CoordConverter instance). All values are optional and if omitted will default to empty strings - it is permissible to pass {} (the empty dictionary) as the detailedInformation parameter.

← **permitDuplicateNames Bool** : (optional) Set to true if the function should create a model with a duplicate name if modelName already refers to an existing model. Default for this, if omitted, is False so that model names are unique.

Returns:

List : List of status number (integer), status message (string), model id created (integer) The status number is 0 on success and the model id is 0 on failure.

Example:

```
>># This will create a new model with a description set but all other fields empty and fail if there is a model called
>>info = {}
>>info["Description"] = "some description of the model"
>>ret = CreateDutModel("model name", "model type", "Existing Ir Database", info)
>>print ret
>>>[0,'OK',789]
```

2.18.2.18 `UpdateDutModel` (**modelId, informationToUpdate, permitDuplicateNames** = `False`)

Description

This function updates and existing model. The model may have been created either by CreateDutModel or manually via the admin console.

Parameters:

← **modelId Integer** : Id of the model to update (see ListDutModels to get the ids of existing models)

← **informationToUpdate Dictionary** of strings : A dictionary of information about the model to be updated in a key value manner. This allows future expansion of the range of information stored without breaking existing scripts. The keys are NOT case sensitive but the values are. Possible keys are: Name (the name of the model as per CreateDutModel), ModelType (the type of the model as per CreateDutModel), IrDatabase (the IR database to use as per CreateDutModel), Description (a description of the model), Manufacturer (the name of the model's manufacturer), ManufacturerModel (the model name given by the manufacturer to the model being created), MajorHwVersion (the major version string of the model), MinorHwVersion (the minor version string of the model), Broadcaster (the broadcaster that the model is tied to), VideoTransform (the transformation to apply to the video between design to world coordinate space. Must be a CoordConverter instance). If a key is omitted then that value is left unchanged (unlike CreateDutModel, it will NOT be set to the empty string - to do that, an empty string must be supplied explicitly)

← **permitDuplicateNames Bool** : (optional) Set to true if the function should change the model name if it would result in a model with a duplicate name of an already existing model. Default for this, if omitted, is False so that model names are unique. This flag can be ignored if no attempt is being made to change the model name itself (ie if Name is not used as a key)

Returns:

List : List of status number (integer), status message(string). Status number is 0 on success.

Example:

```
>>#  This will update a model by changing the hw versions
>>info = {}
>>info["MajorHwVersion"] = "1"
>>info["MinorHwVersion"] = "0.8a"
>>ret = UpdateDutModel(789, info)
>>print ret
>>>[0,'OK']
```

2.18.2.19 `DeleteDutModel` (modelId)

Description

This function deletes a model AND ALL DUTs which are of that model type. Use with caution.

Parameters:

← **modelId Integer** : Id of the model to delete (see ListDutModels to get the ids of existing models)

Returns:

List : List of status number (integer), status message(string). Status number is 0 on success.

Example:

```
>># This will delete a model
>>ret = DeleteDutModel(789)
>>print ret
>>>[0,'OK']
```

2.18.2.20 `ListDutModels` ()

Description

This function will return a list of all existing DUT models in the system

Parameters:

None

Returns:

List : List of 3 objects. The first element is the status and the second is the status message. The final element is a list of dictionaries. Each element of the list represents a model in the system. The contents of the model information dictionary are key=value pairs, with the keys being that of UpdateDutModel plus the key ModelId being set to the model id and VideoTransform being the video transform. This is a CoordConverter or None if there is no transform specified in the database.

Example:

```
>># This will list all models
>>ret = ListDutModels()
>>print ret
>>>[0, 'OK',[{'ModelId': '789', 'Name': 'model name' ...
```

2.19 Audio Video Analysis API

These commands control the audio and video analysis feature of StormTest.

Enums

- enum PerceptualParameters
- enum VideoFormat
- enum VQAVideoOptions
- enum VQASilenceMode
- enum VQAAudioOptions
- enum TriggerCondition
- enum VQAEventType
- enum VQAEvent
- enum VQAVideoResult
- enum VQAAudioResult
- enum VQAEngineStatus
- enum VQAEngine
- enum MeasurementTime

  MeasurementTime Enum.

Functions

- VQACreateEngine
- VQAStartAnalysis
- VQAStopAnalysis
- VQACloseAnalyzer
- VQAWaitEvent
- VQACreateTrigger
- VQACreateTriggerFromImage
- VQACreateTriggerFromIcon
- VQACreateTriggerFromColor
- VQACreateVideoOptions
- VQACreateAudioOptions
- VQAGetLicenseInfo
- VQAIsSlotLicensed
- StartVideoAnalysis
- StopVideoAnalysis
- SetVideoAnalysisParameters
- ReadVideoCalibrationData
- WriteVideoCalibrationData
- DeleteVideoCalibrationData
- GetVideoAnalysisResult
- GetVideoAnalysisData
- ClearVideoAnalysisResults
- ResetAudioAnalysisParameters
- SetAudioAnalysisParameters

- ReadAudioCalibrationData
- WriteAudioCalibrationData
- DeleteAudioCalibrationData
- SetAudioMeasurementTime
- GetAudioMeasurementTime
- SetAudioAnalysisChannel
- GetAudioAnalysisChannel
- SetAudioToneParameters
- GetAudioToneParameters
- SetAudioImpulseParameters
- GetAudioImpulseParameters
- SetSubBandPowerBandList
- GetSubBandPowerBandList
- SetOutOfBandPowerBandList
- GetOutOfBandPowerBandList
- SetSpuriousToneBandList
- GetSpuriousToneBandList
- StartAudioAnalysis
- StopAudioAnalysis
- GetOverallAudioAnalysisStatus
- GetResultPowerSpectrum
- GetResultToneDetection
- GetResultAverageToneAmplitude
- GetResultSpuriousTone
- GetResultSubbandPower
- GetResultOutOfBandPower
- GetResultImpulseNoise

2.19.1 Detailed Description

These commands control the audio and video analysis feature of StormTest.

This is licensed separately from the main server on a per server basis. These calls check for a valid license and will not work without a valid license. There are two types of audio/video analysis supported by StormTest. Video Quality Analysis (VQA) introduced in release 3.2.3 analyzes the video in a manner similar to a group of human viewers, generating a Mean Opinion Score (MOS). Using this requires some knowledge of the way your video is generated and the API provides many options. Functions and classes specific to this type of analysis are prefixed with VQA to distinguish them from the older video analysis. Release 3.2.5 added audio analyis, producing a similar MOS score. The audio analyis works on chunks of audio of 1 second in length.

VQA has an option to use memory to either perform a VQA analysis over a minute or 2 at full frame rate or to smooth out variations in processing over a longer duration. This is the backlog option. The memory used is shared with the High Speed Video Timing API For standard issue slave servers, the total memory available varies slightly over time due to dynamic memory allocation/deallocation in the server. However, it has been verified that if the total memory requested between VQA and all slots of High Speed Timer on the slave is less than 4.6656 billion then the requests will be satisfied. (On HV16HD, there are four slaves, slots 1-4 being the first slave). Each image of video requires width $*$ hight $*$ 1.5 bytes. So from the above calculation, you could allocate all memory to VQA and at 1920 x 1080 that would mean a maximum backlog of 4665600000 / (1920 ∗ 1080 ∗ 1.5) => 1500. Alternatively, you could allocate 375 to each slot of high speed timer and not use the VQA.

A second method uses a reference stream supplied with predefined video and audio patterns. This was introduced in release 2.4 of StormTest.

2.19.2 Function Documentation

2.19.2.1 `VQACreateEngine` (**videoOptions** = `None`, **audioOptions** = `None`)

Description

This function creates a VQAEngine. This is identical to simply creating a VQAEngine using code: VQAEngine(videoOptions, audioOptions)

Parameters:

← **videoOptions** VQAVideoOptions : (optional) video options

← **audioOptions** VQAAudioOptions : (optional) audio options

Returns:

engine *VQAEngine* : the VQAEngine object

Example:

```
>>>vqa = VQACreateEngine()  # this is identical to vqa = VQAEngine()
```

2.19.2.2 `VQAStartAnalysis` (**engine**, **trigger** = `None`, **callbackFunction** = `None`, **slotNo** = `0`)

Description

This function starts video analysis using a previously created VQAEngine. Please see the documentation for VQAEngine.Start() method for full details. This function is identical to calling Start(trigger, callbackFunction, slotNo) on the VQAEngine object.

Parameters:

← **engine VQAEngine** : the engine that has previously been created

← **trigger TriggerCondition** : (optional) The trigger condition to start VQA analysis.

callbackFunction[in] VQACallback : (optional) The callback function to be called whenever an event of interest occurs.

← **slotNo Integer** : (optional) The slot number to use. If omitted or set to 0 then all reserved slots will be used.

Returns:

None

Exceptions:

StormTestExceptions is thrown on an error (for example bad parameters, not enough memory, invalid slot, no communication).

Example:

```
>>>vqa = VQACreateEngine()
>>>VQAStartAnalysis(vqa)                # start on all reserved slots immediately and without a callback
# the above code is identical in effect to:
>>>vqa = VQAEngine()
>>>vqa.Start()
```

2.19.2.3 `VQAStopAnalysis` **(engine, trigger** = `None`**)**

Description

This function stops video analysis using a previously created VQAEngine. Please see documentation for VQAEngine.Stop() method for full details. This function is identical to calling Stop(trigger) on the VQAEngine object.

Parameters:

← **engine VQAEngine** : the engine that has previously been created

← **trigger TriggerCondition** : (optional) the trigger condition to stop the analysis.

Returns:

None

Exceptions:

StormTestExceptions is thrown on an error (engine not running, invalid paramters, no communication)

Example:

```
>>>vqa = VQACreateEngine()
>>>VQAStartAnalysis(vqa)
>>>stopCondition = VQACreateTrigger()
>>>stopCondition.Delay = 30
>>>VQAStopAnalysis(vqa, stopCondition)    # stop on a trigger. That is set to 30 seconds so the stop will stop 30 second
# The above code is identical to:
>>>vqa = VQAEngine()
>>>vqa.Start()
>>>stopCondition = TriggerCondition()
>>>stopCondition.Delay = 30
>>>vqa.Stop(stopCondition)    # stop on a trigger. That is set to 30 seconds so the stop will stop 30 seconds in the fut
```

2.19.2.4 `VQACloseAnalyzer` (engine)

Description

This function close a video analyzer using a previously created VQAEngine. Please see documentation for VQAEngine.Close() method for full details. This function is identical to calling Close() on the VQAEngine object.

Parameters:

← **engine VQAEngine** : the engine that has previously been created

Returns:

None

Exceptions:

StormTestExceptions is thrown on an error (engine not running, invalid paramters, no communication)

Example:

```
# example to just run for a while, get no results and stop everything. Not the most useful program ever written.
>>>vqa = VQACreateEngine()
>>>VQAStartAnalysis(vqa)
>>>WaitSec(5)
>>>VQACloseAnalyzer(vqa)    # stop everything and clean up
# The above code is identical to:
>>>vqa = VQAEngine()
>>>vqa.Start()
>>>WaitSec(5)
vqa.Close()                   # stop everything and clean up
```

2.19.2.5 `VQAWaitEvent` (engine, eventList)

Description

This function waits for one or more events from a previously created VQAEngine. Please see documentation for vQAEngine.WaitEvent() method for full details. This function is identical to calling WaitEvent(eventList) on the VQAEngine object.

Parameters:

← **engine VQAEngine** : the engine that has previously been created

← **eventList** List : list of events. The call will return when any one of those events occur.

Returns:

events List : List of VQAEvent objects. Each event object identifies its type and some optional parameters.

Note:

You should be aware of threading issues with this call. In particular, you should not create multiple threads, each waiting on the same event. If, for example, you create 2 threads, each waiting on the Start event, there is no guarantee that both threads will get the event - they might but it is highly likely that only 1 thread will get the event and the other would wait forever (or until engine is closed or slot released whichever comes first).

Example:

```
>>>vqa = VQAEngine()
>>>vqa.Start()
>>>WaitSec(2)
>>>events = VQAWaitEvent([])
>>>print "We have {0} events".format(len(events))
>>>We have 25 events
# The above code is identical to:
>>>vqa = VQAEngine()
>>>vqa.Start()
>>>WaitSec(2)
>>>events = vqa.WaitEvent([])
```

2.19.2.6 `VQACreateTrigger` ()

Description

This function creates a default TriggerCondition object. It is identical to creating the trigger object using trigger = TriggerCondition().

Parameters:

None

Returns:

TriggerCondition : the default trigger object. This will simply trigger immediately.

Example:

```
>>>create a trigger to trigger after 1000 frames
>>>trigger = VQACreateTrigger()
>>>trigger.FrameCount = 1000
```

2.19.2.7 `VQACreateTriggerFromImage` **(image, rect** = `None`, **threshold** = 98**)**

Description

This function creates a trigger condition object from an image. It is identical to calling TriggerCondition.FromImage(image, rect). Please see documentation for the FromImage() method of TriggerCondition for full details

Parameters:

← **image** StormTestImageObject : the image to use as the reference.

← **rect** List or Tuple : (optional) the region of interest within the image.

← **threshold** double : (optional) the threshold for the image match as per CompareImageEx.

Returns:

TriggerCondition : the completed trigger condition ready to use.

Example:

```
>>>create a trigger to trigger on image but using rect at (0,0,400,300)
>>>trigger = VQACreateTriggerFromImage(image, [0,0,400,300])
```

2.19.2.8 `VQACreateTriggerFromIcon` (**image, location, threshold** = 98)

Description

This function creates a trigger condition object from an icon. It is identical to calling Trigger-Condition.FromIcon(image, location). Please see documentation for the FromIcon() method of TriggerCondition for full details

Parameters:

← **image** StormTestImageObject : the image to use as the reference icon.

← **location** List or Tuple : the location of the icon within the image.

← **threshold** double : (optional) the threshold for the icon match as per CompareIconEx.

Returns:

TriggerCondition : the completed trigger condition ready to use.

Example:

```
>>>create a trigger to trigger on icon but using location at (89,77)
>>>trigger = VQACreateTriggerFromIcon(image, [89,77])
```

2.19.2.9 `VQACreateTriggerFromColor` **(rect, color, tolerance** = (4,4, **flatness** = 98, **peakError** = 0**)**

Description

This function creates a trigger condition object from a color. It is identical to calling TriggerCondition.FromColor(rect, color, tolerance, flatness, peakError). Please see documentation for the FromColor() method of TriggerCondition for full details

Parameters:

← **rect** List or Tuple : the region of interest within the screen capture.

← **color** List or Tuple : the color as a List or Tuple of Red, Green, Blue values.

← **tolerance** List or Tuple : (optional) the tolerance to apply to color comparison as per CompareColorEx function.

← **flatness** double : (optional) the flatness of the color as per CompareColorEx function.

← **peakError** double : (optional) the peak error of the color as per CompareColorEx function.

Returns:

TriggerCondition : the completed trigger condition ready to use.

Example:

```
>>>create a trigger to trigger on color black at (0,0, 400,100)
>>>trigger = VQACreateTriggerFromColor((0,0,400,100), (0,0,0))
```

2.19.2.10 `VQACreateVideoOptions` ()

Description

This function creates the default video options object. It is identical to simply creating the object using the default constructor: options = VQAVideoOptions().

Parameters:

None

Returns:

VQAVideoOptions : The completely constructed VQAVideoOptions object.

Example:

```
>>>vqaOptions = VQACreateVideoOptions()
>>>print vqaOptions.Jerkiness, vqaOptions.Backlog, vqaOptions.Format
>>>True 500 H264
>>>vqaOptions.Backlog = 1500
>>>print vqaOptions.Jerkiness, vqaOptions.Backlog, vqaOptions.Format
>>>True 1500 H264
```

2.19.2.11 `VQACreateAudioOptions` ()

Description

This function creates the default audio options object. It is identical to simply creating the object using the default constructor: options = VQAAudioOptions().

Parameters:

None

Returns:

VQAAudioOptions : The completely constructed VQAAudioOptions object.

Example:

```
>>>vqaOptions = VQACreateAudioOptions()
>>>print vqaOptions.SilenceInfluence, vqaOptions.Enabled
>>>0 True
>>>vqaOptions.SilenceInfluence = 25
>>>print vqaOptions.SilenceInfluence, vqaOptions.Enabled
>>>25 True
```

2.19.2.12 `VQAGetLicenseInfo` (server)

Description

Gets the VQA License information for one or more servers. This function can be called before connecting to a server or reserving a slot. It can only find out about servers in the current facility - this can be controlled via SetMasterConfigurationServer. If SetMasterConfigurationServer is not called, the default facility set during installation of StormTest client software is used.

Parameters:

← **server** String or List : The server name to query for VQA license info. It can be a list of server names or the empty list to mean 'all servers'.

Returns:

licenseInfo *List* : List of licensing information. Each item in the list is a Tuple of (serverName, slotNumber, expiryTime). The expiryTime is the expiry time in seconds since the Unix epoch (1/1/1970 00:00:00 UTC) when the license will expire. Perpetual licenses are returned with an expiryTime equal to the maximum intger. Only server/slot combinations with a valid license are returned. If a server has all 16 slots licensed then there are 16 elements in the list.

Example:

```
>>>print VQAGetLicenseInfo("myServer")
>>>[(myServer,5,1499817600),(myServer,6,1499817600),(myServer,7,1499817600),(myServer,8,1499817600)]
>>>print VQAGetLicenseInfo("nonLicensedServer")
>>>[]
```

2.19.2.13 `VQAIsSlotLicensed` (**slotNo** = 0)

Description

Tests whether the slot is licensed for VQA or not. It can only test reserved slots and is intended for scripts to easily determine whether they should start VQA video analysis or not.

Parameters:

← **slotNo** Integer : (optional) slot number to test

Returns:

status Boolean : If slotNo is non zero then the return will be a simple True / False status as to whether the slot is licensed for VQA. If slotNo is 0 then a list of tuples will be returned. Each tuple is (slotNo, status).

Note:

It is theoretically possible to call this function a few seconds before a license expires and thus you could have a situation where VQAIsSlotLicensed returns True but VQAStartAnalysis fails due to an expired license. To prevent this, the VQAIsSlotLicensed call ensures that the license has at least 10 minutes to run. You may, of course, ignore the return from VQAIsSlotLicensed and continue to attempt VQAStartAnalysis().

Exceptions:

StormTestExceptions is raised if slotNo is not reserved to the script.

Example:

```
>>>print VQAIsSlotLicensed(1)
>>>True
>>>print VQAIsSlotLicensed()
>>>[(1,True)]
```

2.19.2.14 `StartVideoAnalysis` (**fps**, **testStream** = `"TestStreamA"`, **useValuesFromDatabase** = `True`, **continueOnError** = `True`, **slotNo** = `0`)

Description

This API initiates the video analysis. It cannot be used while a High Speed Timer is reserved on the same slot.

Parameters:

← **fps Integer:** The number of frames captured per seconds during analysis

← **testStream String:** (optional) The test stream on which the video analysis is based, currently only TestStreamA is supported

← **useValuesFromDatabase Boolean:** (optional) Instruct the call to read the database. This is new in 2.7 so tests using this parameter cannot run on 2.6 and earlier.

← **continueOnError Boolean:** (optional) Instruct the call to continue the analysis if database has no values (using default values.) This is new in 2.7 so tests using this parameter cannot run on 2.6 and earlier

← **slotNo Integer:** (optional) The slot number on which to start the video analysis.

Returns:

Boolean: status (True for success/False for failure). If slotNo is 0 then a list of items is returned. Each item is a list of [slotNo, bool] where the bool is true or false as described earlier.

Note:

2.7 added parameters. Older scripts will run without modification as the API detects the type of the 3rd parameter an an integer, (if specified) OR it will use the defaults that a 2.6 test used. ie StartVideoAnalysis(fps) will behave the same on 2.7 and 2.6. BUT StartVideoAnalysis(fps,"TestStreamA",True,False,0) will not work on 2.6

Example:

```
ret = StartVideoAnalysis(5,"TestStreamA",True, False,1)
print ret
>>>True
WaitSec(10)   # run the analysis for 10 seconds
ret = StopVideoAnalysis()
print ret
>>>[[5,True]]
```

2.19.2.15 `StopVideoAnalysis` **(slotNo** = 0**)**

Description

Stops the video analysis. This function can only be called after StartVideoAnalysis has been called.

Parameters:

← **slotNo Integer:** (optional) The slot number on which to start the video analysis.

Returns:

Boolean: status (True for success/False for failure). If slotNo is 0 then a list of items is returned. Each item is a list of [slotNo, bool] where the bool is true or false as described earlier.

Example:

```
ret = StopVideoAnalysis()
print ret
>>>[[1,True]]
```

2.19.2.16 `SetVideoAnalysisParameters` (**calData, slotNo** = 0)

Description

Sets the video analysis parameters to the calData (a string). The calData can be a NamedString. Using a NamedString would be recommended as the format of data is complex. After using this function, StartVideoAnalysis() should be called by setting the useValuesFromDatabase parameter to False (otherwise this functions operation will be overwritten)

Parameters:

← **calData string** : the calibration data. A non documented format. Can be obtained from ReadVideoCalibrationData

← **slotNo integer** : (optional) The slot number where the calibration data is being set.

Returns:

boolean: True for success. If slotNo is 0 then a list of items is returned, each item is a list of [slotNo, bool] where the bool is true or false as described earlier.

2.19.2.17 `ReadVideoCalibrationData` **(modelId, hwOutput** = `None`**)**

Description

Looks up the video data for the specified model id and hardware output. The return is a string (not documented for public use). An exception is thrown on an error. The returned string can be passed to WriteVideoCalibrationData() or SetVideoAnalysisParameters(). This function can be called without reserving a slot.

Parameters:

← **modelId integer:** The model id. These can be found from ListDutModels.

← **hwOutput string:** (Optional) The output of interest (valid only with a specific AVSwitch). Can be None if no AVSwitch is in system to mean 'default raw output'.

Returns:

string: The calibration data (None if there is no data). Format is undocumented.

2.19.2.18 `WriteVideoCalibrationData` (**modelId, data, hwOutput** = `None`)

Description

Writes the video data for the specified model id and hardware output. An exception is thrown on an error. This function can be called without reserving a slot. This function is intended for scripts doing maintenance functions where the data from a model may need to be cloned to many other known similar models. It would have limited use in a normal test script.

Parameters:

← **modelId integer:** The model id. These can be found from ListDutModels.

← **data string:** The calibration data to write. This should be returned from ReadVideoCalibrationData.

← **hwOutput string:** (optional) The output of interest (valid only with a specific AVSwitch). Can be None if no AVSwitch is in system to mean 'default raw output'.

Returns:

None

2.19.2.19 `DeleteVideoCalibrationData` (**modelId, hwOutput** = `None`)

Description

Deletes the video data for the specified model id and hardware output. An exception is thrown only if there is a problem with communicating to the database or parameters are the wrong type. No exception is thrown if there is no data to delete (either the model id is invalid or the hwOutput does not exist). This function can be called without reserving a slot.

Parameters:

← **modelId integer:** The model id. These can be found from ListDutModels.

← **hwOutput string:** (Optional) The output of interest (valid only with a specific AVSwitch). Can be None if no AVSwitch is in system to mean 'default raw output'.

Returns:

None

2.19.2.20 `GetVideoAnalysisResult` **(slotNo** = 0**)**

Description

Retrieves the result of the video analysis result run from the server. This function can only be run after StopVideoAnalysis has been called.

Parameters:

← **slotNo Integer:** (optional) The slot number on which to start the video analysis.

Returns:

Array: The return is a Array of arrays containing slot number, status (true/false) indicating whether the success or not of the retrieval.
A list containing two bool values indicating pass/fail for the video quality and video timing measures respectively, and a third element which is an array of tuples consisting of the (Frames,Captured,Unmatched), (LumaVariance,MaxDetected,Threshold) (ChromaVariance,MaxDetected,Threshold), (MaxLuma,MaxLumaDiff,ToleranceLuma), (MaxChroma,MaxChromaDiff,ToleranceChroma), (MaxJitter,MaxDetected,Threshold), (ColorBlockErrorCount,Count) The returned results are described as follows:
Frames Captured, is the number of video frames captured during the analysis.
Frames Unmatched, is the number of frames where no pattern match could be found.
LumaVariance MaxDetected, is the maximum luma variance value detected on all video frames analysed.
LumaVariance Threshold, is the threshold that the LumaVariance is set to, which determines a success or failure.
ChromaVariance MaxDetected, is the maximum chroma variance value detected on all video frames analysed.
ChromaVariance Threshold, is the threshold that the ChromaVariance is set to, which determines a success or failure.
.MaxLuma MaxLumaDiff, is the maximum luma difference found from the reference colors during the analysis.
MaxLuma ToleranceLuma, is the threshold that the luma difference is set to, which determines a success or failure.
MaxChroma MaxChromaDiff, is the maximum chroma difference found from the reference colors during the analysis.
MaxChroma ToleranceChroma, is the threshold that the chroma difference is set to, which determines a success or failure.
MaxJitter MaxDetected, is the maximum percentage that the jitter was calculated to be during the analysis.
MaxJitter Threshold, is the threshold that the percentage jitter value is set to, which determines a success or failure.
ColorBlockErrorCount Count, is the count of the number a blocks analysed which had errors in them.

Example:

```
ret = GetVideoAnalysisResult(1)
print ret
>>( True, [(True, True)], [('Frames', 50, 0), ('LumaVariance', 2.1066948676325699, 4.2144000000000004),
>>('ChromaVariance', 2.0231123477330399, 3.9857), ('MaxLumaDiff', 2, 15), ('MaxChromaDiff', 1, 15), ('MaxJitter', 37, 50
>>('ColorBlockErrorCount', 0)]) #A successful result
```

2.19.2.21 `GetVideoAnalysisData` (**returnRawResult** = `False`, **slotNo** = `0`)

Description

Retrieves the video analysis results from the last run video analysis sequence on the slot of interest. This function can only be run after StopVideoAnalysis has been called.

Parameters:

← **returnRawResult Bool:** (optional) A flag to determine whether or not to return the raw result along with the processed data If the raw result is returned it is the last item in the result array

← **slotNo Integer:** (optional) The slot number on which to get the video analysis data

Returns:

Array: An array of arrays containing slot number,status (true/false) indicating whether the success or not of the retrieval.
The result array is described in the following manner (Frames,Captured,Unmatched), (LumaVariance,MaxDetected,Threshold), (ChromaVariance,Max Detected,Threshold), (MaxLuma,MaxLumaDiff,ToleranceLuma), (MaxChroma,MaxChromaDiff,ToleranceChroma), (MaxJitter,MaxDetected,Threshold), (MinColorValues,MinDetected), (MaxColorValues,MaxDetected), (AvgColorValues,AvgDetected), (ColorBlockErrorCount,Count)[rawResult])
If returnRawResult is set to true the last item in the above array will be the actual raw data numbers. The returned results are described as per GetVideoAnalysisResult and:
MinColorValues, is a string describing the minimum color values detected during the analysis.
MaxColorValues, is a string describing the maximum color values detected during the analysis.
AvgColorValues, is a string describing the average color values detected during the analysis.

Example:

```
ret = GetVideoAnalysisData(False,1)
print ret
(True, [('Frames', 50, 0), ('LumaVariance', 2.1290282747187601, 4.5122999999999998), ('ChromaVariance', 2.01698818979377
('MaxLumaDiff', 1, 15), ('MaxChromaDiff', 5, 15), ('MinColorValues', u'grey\t119,128,128 white\t223,127,126 cyan\t161,16
yellow\t200,12,146 green\t138,50,30 magenta\t100,203,223 blue\t38,240,109 red\t77,88,240'),
('MaxColorValues', 'grey\t120,128,128 white\t224,127,126 cyan\t162,166,13 yellow\t202,13,146 green\t139,51,31 magenta\t1
blue\t39,241,109 red\t78,89,241'), ('AvgColorValues', u'grey\t119,128,128 white\t224,127,126 cyan\t162,166,12 yellow\t20
green\t138,51,31 magenta\t101,203,224 blue\t39,241,109 red\t77,89,241'),('ColorBlockErrorCount', 0)])
```

2.19.2.22 `ClearVideoAnalysisResults` (**slotNo** = 0)

Description

Clears the detailed results from a previous video analysis. The results are stored after analysis and consume resources. When you have finished retrieving the results, it is a good idea to clear them with this function. After calling this function the overall summary result is still available but the detailed values will no longer be available. The results will be cleared automatically when a slot is reserved, when a new Video analysis is started or when a High Speed Timer is reserved (the underlying implementation of video analysis uses the High Speed Timer infrastructure).

Parameters:

← **slotNo Integer:** (optional) The slot number on which to clear the results

Returns:

None

Example:

```
>> ClearVideoAnalysisResults()
```

2.19.2.23 `ResetAudioAnalysisParameters` ()

Description

This function resets the audio analysis parameters to the default values. These include the measurement time, tone parameters, impulse parameters band for out of band power, band list for sub band power, band list for spurious tone detection.

Returns:

None

Example:

```
>>ResetAudioAnalysisParameters()
```

2.19.2.24 `SetAudioAnalysisParameters` **(calData, slotNo** = 0**)**

Description

Sets the audio analysis parameters to the calData (a string). The calData can be a NamedString. Using a NamedString would be recommended as the format of data is complex. After using this function, StartAudioAnalysis() should be called by setting the useValuesFromDatabase parameter to False (otherwise this functions operation will be overwritten)

Parameters:

← **calData string** : the calibration data. A non documented format. Can be obtained from ReadAudioCalibrationData

← **slotNo integer** : (optional) The slot number where the calibration data is being set.

Returns:

boolean: True for success. If slotNo is 0 then a list of items is returned, each item is a list of [slotNo, bool] where the bool is true or false as described earlier.

2.19.2.25 `ReadAudioCalibrationData` (**modelId**, **hwOutput** = `None`)

Description

Looks up the audio data for the specified model id and hardware output. The return is a string (not documented for public use). An exception is thrown on an error. The returned string can be passed to WriteAudioCalibrationData() or SetAudiooAnalysisParameters(). This function can be called without reserving a slot.

Parameters:

← **modelId integer:** The model id. These can be found from ListDutModels.

← **hwOutput string:** (Optional) The output of interest (valid only with a specific AVSwitch). Can be None if no AVSwitch is in system to mean 'default raw output'.

Returns:

string: The calibration data (None if there is no data). Format is undocumented.

2.19.2.26 `WriteAudioCalibrationData` **(modelId, data, hwOutput** = `None`**)**

Description

Writes the audio data for the specified model id and hardware output. An exception is thrown on an error. This function can be called without reserving a slot. This function is intended for scripts doing maintenance functions where the data from a model may need to be cloned to many other known similar models. It would have limited use in a normal test script.

Parameters:

← **modelId integer:** The model id. These can be found from ListDutModels.

← **data string:** The calibration data to write. This should be returned from ReadAudioCalibrationData.

← **hwOutput string:** (optional) The output of interest (valid only with a specific AVSwitch). Can be None if no AVSwitch is in system to mean 'default raw output'.

Returns:

None

2.19.2.27 `DeleteAudioCalibrationData` (**modelId, hwOutput** = `None`)

Description

Deletes the audio data for the specified model id and hardware output. An exception is thrown only if there is a problem with communicating to the database or parameters are the wrong type. No exception is thrown if there is no data to delete (either the model id is invalid or the hwOutput does not exist). This function can be called without reserving a slot.

Parameters:

← **modelId integer:** The model id. These can be found from ListDutModels.

← **hwOutput string:** (Optional) The output of interest (valid only with a specific AVSwitch). Can be None if no AVSwitch is in system to mean 'default raw output'.

Returns:

None

2.19.2.28 `SetAudioMeasurementTime` (timePeriod)

Description

This function sets the time period within which the audio is measured and the power spectrum is generated.

Parameters:

← **timePeriod MeasurementTime:** time to compute a power spectrum result.

Returns:

None

Example:

```
>>SetAudioMeasurementTime(MeasurementTime.OneEighthSec)
```

Note:

The API makes an assumption of 16kHz audio. The StormTest HD system uses a sampling value of 48kHz. To preserve readability of code, the enumerated types are converted to give an approximate time similar to that specified. The actual values for HD are thus: OneSixteenthSec = 85.3mS, OneEighthSec = 171mS, OneFourthSec = 341mS, HalfSec = 683mS, OneSec = 1.365sec, TwoSec, FourSec and EightSec = 2.731sec (the low level cannot use more than 128K sample length). If you wish to set the FFT sample count explicitly, remember that for an HD system, you need to set a value of 1/4 of what you want. So for HD system to get 4096 samples, call SetAudioMeaurementTime(4096/4). Only powers of 2 are permitted and the range for HD is 1024 -> 32768 inclusive. (This will translate to 4096 -> 131072 samples, 85.3 mS -> 2.731 seconds).
Users should understand the trade-off between the time (how quickly a result can be obtained) and the frequency resolution to use this API. The default is MeasurementTime.OneEighthSec (i.e. 2048 samples used for standard definition audio at 16 kHz to compute an FFT result)

2.19.2.29 `GetAudioMeasurementTime` ()

Description

This function gets the time period within which the audio is measured and the power spectrum is generated.

Returns:

Number of samples for computing an FFT result (see MeasurementTime) See note on SetAudioMeasurementTime for HD systems.

2.19.2.30 `SetAudioAnalysisChannel` (**audioChannel**, **slotNo** = 0)

Description

This function sets the channel to use on HD systems for audio analysis and audio detection API calls. (GetAudioLevel, WaitAudioAbsence, WaitAudioPresence). It has no effect on standard definition systems. By default, Left channel is used after reserving a slot until explicitly set to Right. It only effects the next audio operation and does not alter an audio analysis in progress.

Parameters:

← **audioChannel,:** (**AudioChannel**) The audio channel to use.

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

Boolean or List of Lists: When slot is zero this is a list of lists, each being [SlotNumber, Status] but just status if slotNo is non zero. Return is True if successfully set, else false.

Note:

This setting is independent of any AV switch in the system. For standard definition systems, this will not follow any setting of the AVSwitch.

Example:

```
>>SetAudioAnalysisChannel(AudioChannel.Left)
>>[[5,True]]
```

2.19.2.31 `GetAudioAnalysisChannel` (**slotNo** = 0)

Description

This function gets the audio channel set by SetAudioAnalysisChannel. On standard definition systems it always returns AudioChannel.Left. It returns the current channel in use if any analysis is in progress but otherwise the channel that will be used for the next analysis.

Parameters:

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

AudioChannel or List of Lists: When slot is zero this is a list of lists, each being [SlotNumber, AudioChannel] but just the AudioChannel if slotNo is non zero.

Note:

This setting is independent of any AV switch in the system. For standard definition systems, this will not follow any setting of the AVSwitch. It will always return Left.

Example:

```
>>GetAudioAnalysisChannel()
>>[[5,Left], [6, Right]]
```

2.19.2.32 `SetAudioToneParameters` (**toneFrequency** = `125.0`, **toneAmplitude** = `2.0`, **toneTolerance** = `10.0`, **numOfTonesToAverage** = `10`, **guardFrequency** = `90.0`, **guardAmplitude** = `-26.36`)

Description

Sets parameters for the tone detection algorithm The tone detection algorithm detects if a certain frequency, at an expected algorithm, is present within a defined tolerance. It also detects if a (configurable) number of successive measurements are the same, within that tolerance. The analysis allows for a "guard" tone, used to detect test stream loop points. If this is >0, then results containing that frequency are ignored.

Parameters:

← **toneFrequency Double** - Expected frequency in Hz of the tone

← **toneAmplitude Double** - Expected amplitude in dBu of the tone

← **toneTolerance Double** - Allowed tolerance in dB from the expected amplitude or average amplitude

← **numOfTonesToAverage Integer** - Number of sets of results to average

← **guardFrequency Double** - Expected frequency in Hz of the guard tone

← **guardAmplitude Double** - Expected amplitude in dBu of the guard tone

Returns:

None

Example:

```
SetAudioToneParameters(125.0, -6.36, 9.0, 10, 90.0, -26.36)
# Sets the analysis tone parameters
```

2.19.2.33 `GetAudioToneParameters` ()

Description

Gets the current audio tone parameters

Returns:

List - List of the tone analysis parameters. Same order as the SetAudioToneParameters routine

Example:

```
 >>toneParameters = GetAudioToneParameters()
 >>print "Tone parameters: " + str(toneParameters)

# Retrieves the tone analysis parameters and prints them
```

2.19.2.34 `SetAudioImpulseParameters` (**impulseThreshold** = `0.1`, **maxAllowedPerSecond** = `2`)

Description

Sets parameters for the impulse detection algorithm. The impulse detection algorithm detects detects impulse above a certain threshold, and will ignore a certain number per second. This is used for craxkle/pop detection.

Parameters:

← **impulseThreshold Double** - Maximum allowed impulse size in dBu

← **maxAllowedPerSecond Integer** - Maximum number allowed per second

Returns:

None

Example:

```
>>SetAudioImpulseParameters(-40.34, 2)
# Sets the analysis tone parameters
```

2.19.2.35 `GetAudioImpulseParameters` ()

Description

Gets the current audio impulse parameters

Returns:

List - List of the tone impulse parameters. Same order as the SetAudioImpulseParameters routine

Example:

```
 >>impulseParameters = GetAudioImpulseParameters()
 >>print "Impulse parameters: " + str(impulseParameters)

# Retrieves the audio impulse parameters and prints them
```

2.19.2.36 `SetSubBandPowerBandList` (**bandList** = [])

Description

Set the list of bands from which the sub-band power algorithm will get the power results from.

Parameters:

← **bandList** **List** of tuples of doubles. Each tuple consists three doulbe values for start frequency in Hz, end frequency in Hzand power threshold in dB. If bandList is empty list, the StormTest defaults will be used.

Returns:

None

Example:

```
>>SetSubBandPowerBandList([(100,1000,-12),(1000,7000,-24)])
# Sets the analysis tone parameters
```

2.19.2.37 `GetSubBandPowerBandList` ()

Description

Get the list of bands from which the sub-band power algorithm will get the power results from.

Returns:

List - List of the tuples. Each tuple consists three doulbe values for start frequency in Hz, end frequency in Hzand power threshold in dB.

2.19.2.38 `SetOutOfBandPowerBandList` (**bandList** = [])

Description

Set the band from which the out of band power algorithm will get the power results from.

Parameters:

← **bandList List** - consists list of tuples. Each tuple consits of three double values for start frequency in Hz, end frequency in Hz and power threshold in dBu. If bandList is empty list, the StormTest defaults will be used.

Returns:

None

Example:

```
>>SetOutOfBandPowerBandList([(100,1000,-12),(1000,7000,-24)])
# Sets the analysis tone parameters
```

2.19.2.39 `GetOutOfBandPowerBandList` ()

Description

Get the band from which the out of band power algorithm will get the power results from.

Returns:

List - List of three doulbe values for start frequency in Hz, end frequency in Hz and power threshold in dB.

2.19.2.40 `SetSpuriousToneBandList` **(bandList** = []**)**

Description

Set the list of bands from which the spurious tone detection algorithm will check for spurious tones.

Parameters:

← **bandList List** of tuples of doubles. Each tuple consists three doulbe values for start frequency in Hz, end frequency in Hz and power threshold in dB. If bandList is empty list, the StormTest defaults will be used.

Returns:

None

Example:

```
>>SetSpuriousToneBandList([(100,1000,-12),(1000,7000,-24)])
# Sets the analysis tone parameters
```

2.19.2.41 `GetSpuriousToneBandList` ()

Description

Get the list of bands from which the spurious tone detection algorithm will check for spurious tones.

Returns:

List - List of the tuples. Each tuple consists three doulbe values for start frequency in Hz, end frequency in Hzand power threshold in dB.

2.19.2.42 `StartAudioAnalysis` (**useValuesFromDatabase** = `True`, **continueOnError** = `True`, **slotNo** = `0`)

Description

This function starts running a set of audio analysis algorithms on the given slots and the results can be obtained afterwards for each of the algorithm.

Parameters:

← **useValuesFromDatabase boolean** : (optional) By default, the function reads the database for the audio analysis parameters. If >1 slot is used in one script this will cause a problem if the values are different in the database as all audio analyses in a script must use the same set of parameters. If using the database, the current setting of the AV switch (if used on the server as this is optional hardware) will be read so that different DUT audio sources can use different parameters.

← **continueOnError Boolean:** (optional) Instruct the call to continue the analysis if database has no values (using default values.) This is new in 2.7 so tests using this parameter cannot run on 2.6 and earlier

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

boolean: whether the audio analysis has been started successfully. Raises a StormTest exception if there is no license for audio analysis. If slotNo is 0 then a list of items is returned. Each item is a list of [slotNo, bool] where the bool is true or false as described earlier.

Note:

2.7 added a parameter. Older scripts will run without modification as the API detects the type of the 2nd parameter an an integer, (if specified) OR it will use the defaults that a 2.6 test used. ie StartAudioAnalysis() will behave the same on 2.7 and 2.6. BUT StartAudioAnalysis(True,False,0) will not work on 2.6

Example:

```
ret = StartAudioAnalysis(slotNo=16)
print ret
>>>True
WaitSec(10)  # run audio analysis for 10 seconds.
ret = StopAudioAnalysis()
print ret
>>>[[16,True]]
```

2.19.2.43 `StopAudioAnalysis` (**slotNo** = 0)

Description

This function stops audio analysis that has been started previously.

Parameters:

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

boolean: whether the audio analysis has been stopped successfully. If slotNo is 0 then a list of items is returned. Each item is a list of [slotNo, bool] where the bool is true or false as described earlier.

Example:

```
>>StopAudioAnalysis(16)
>>True
```

2.19.2.44 `GetOverallAudioAnalysisStatus` **(slotNo** = 0**)**

Description

This function returns the overall results of audio analysis as a simple pass / fail list. The overall is a pass if all the results are pass (excluding the GetResultPowerSpectrum which is just the raw information). The other returns are true/false on each of the six algoritms tested so a script can provide a simple summary easily. Only useful if running on a pre-conditioned test stream - it is also much faster than getting all the individual results and performing the logical AND of all the values.

Parameters:

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

List: 7 booleans representing pass/fail on Overall, Tone, Average Tone, Sub Band Power, Out of Band Power, Spurious Tone and Impulse Noise (in that order) - list of lists if used with slot 0

Example:

```
>>GetOverallAudioAnalysisStatus()
>>[[1,True, True, True, True, True, True, True],[2,False, True, True, True, True, False, True]
>>GetOverallAudioAnalysisStatus(2)
>>[False, True, True, True, True, False, True]
# Slot 2 failed with a spurious tone somewhere.
```

2.19.2.45 `GetResultPowerSpectrum` (**slotNo** = 0)

Description

This function gets the power spectrum from the last audio analysis run.

Parameters:

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

List: A list of power spectrum results. Each result is formated as [slot_num, status(True/False), power_spectrum_list] where status indicates whether the power spectrums are obtained successfully and power_spectrum_list is a list of power spectrum result that is a list of double values in dB.

Example:

```
>>GetResultPowerSpectrum()
>>[[1,True, [ [-91.152053184410732, -80.712950600594866, -90.533206627045416, ...],[power_list_2], ..., [power_list_n]
      [2,True, [ [-91.152053184410732, -80.712950600594866, -90.533206627045416, ...],[power_list_2], ..., [power_list
>>GetResultToneDetection(1)
>>[1,True, [ [-91.152053184410732, -80.712950600594866, -90.533206627045416, ...],[power_list_2], ..., [power_list_n]
```

2.19.2.46 `GetResultToneDetection` (**slotNo** = 0)

Description

This function gets the expected tone detection results from the last audio analysis run.

Parameters:

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

List: A list of tone detection results, each of which consists [slot_num, overall_status(True/False), tone_detect_list] where overall_status is True if all of the results in tone_detect_list are True, otherwise False, and tone_detect_list is [result(True/False), tone_amplitude, expected_tone_amplitude, tolerance]

Example:

```
>>GetResultToneDetection()
>>[[1, True, [[True, -7.0548659344779008, -9.0, 3.0], ...],...,  [True, -7.5334839270869569, -9.0, 3.0]],
      [2,False, [[True, -7.0548659344779008, -9.0, 3.0], ...],...,  [True, -7.5334839270869569, -9.0, 3.0]] ]
>>GetResultToneDetection(1)
>>[1, True, [[True, -7.0548659344779008, -9.0, 3.0], ...],...,  [True, -7.5334839270869569, -9.0, 3.0]]
```

2.19.2.47 `GetResultAverageToneAmplitude` (**slotNo** = 0)

Description

This function gets the average tone amplitude results from the last audio analysis run.

Parameters:

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

List: A list of average tone amplitude results, each of which consists [slot_num, overall_-status(True/False), average_amplitude_list] where overall_status is True if all of the results in average_amplitude_list are True, otherwise False, and average_amplitude_list is [result(True if max_deviation < tolerance, otherwise False), average_tone_amplitude, tolerance, max_-deviation]. If the slotNo is specified then the slot number is not returned and the result is as shown in the example below.

Example:

```
>>GetResultAverageToneAmplitude()
>>[[1, True, [[True, -5.9446547899255577, 0.90000000000000002, 7.2353169855323793e-005], [True, -5.9446373227612934, 0
>>GetResultAverageToneAmplitude(1)
>>[True, [[True, -5.9446547899255577, 0.90000000000000002, 7.2353169855323793e-005], [True, -5.9446373227612934, 0.900
```

2.19.2.48 `GetResultSpuriousTone` **(slotNo** = 0**)**

Description

This function gets the spurious tone detection results from the last audio analysis run.

Parameters:

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

List: A list of spurious tone detection results, each of which consists [slot_num, overall_-status(True/False), spurious_tone_list] where overall_status is True if all of the results in spurious_tone_list are True, otherwise False, and spurious_tone_list is [result(True if none of the values in moving_sum_power_list is above the threshold, otherwise False), start_-frequency, end_frequency, threshold, detected_spur_tone_list]. detected_spur_tone_list is a list of [start_fr, end_fr, threshold, detected_spur_tone_dBu]. within the band.

Example:

```
>>GetResultSpuriousTone()
>>[[8, False, [[True, []], [True, []], [True, []], [True, []], [False, [[163.0, 7000.0, -40.0, -38.800262866466035]]],
>>GetResultSpuriousTone(8)
>>[False, [[True, []], [True, []], [True, []], [True, []], [False, [[163.0, 7000.0, -40.0, -38.800262866466035]]], [Tr
```

2.19.2.49 `GetResultSubbandPower` (**slotNo** = 0)

Description

This function gets the subband power results from the last audio analysis run.

Parameters:

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

List: A list of subband power results, each of which consists [slot_num, overall_status(True/False), subband_power_list] where overall_status is True if all of the results in subband_power_list are True, otherwise False, and subband_power_list is a list of list [result(True if band_power < threshold, otherwise False), start_frequency, end_frequency, threshold, band_power]

Example:

```
>>SetSubBandListSubBand([(0, 250, -20), (250,500,-50)]) # look at band of 0-250 Hz with toleranece -20 dB and 250-500
>>GetResultSubbandPower()
>>[[1,False, [ [False, [[0, 250.0, -20.0, -25.180790459224692], [250.0, 500.0, -50.0, -33.346035032659643]]],
                     [ [False, [[0, 250.0, -20.0, -25.180790459224692], [250.0, 500.0, -50.0, -33.346035032659643]]],
      [2,False, [ [False, [[0, 250.0, -20.0, -25.180790459224692], [250.0, 500.0, -50.0, -33.346035032659643]]],
                     [ [False, [[0, 250.0, -20.0, -25.180790459224692], [250.0, 500.0, -50.0, -33.346035032659643]]],
>>GetResultSubbandPower(1)
>>[False, [ [False, [[0, 250.0, -20.0, -25.180790459224692], [250.0, 500.0, -50.0, -33.346035032659643]]],
                     [ [False, [[0, 250.0, -20.0, -25.180790459224692], [250.0, 500.0, -50.0, -33.346035032659643]]],
```

2.19.2.50 `GetResultOutOfBandPower` (**slotNo** = 0)

Description

This function gets the out of band power results from the last audio analysis run.

Parameters:

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

List: A list of out of band power results, each of which consists [slot_num, overall_status(True/False), oob_power_list] where overall_status is True if all of the results in oob_power_list are True, otherwise False, and oob_power_list is a list of list [result(True if band_power < threshold, otherwise False), [[start_frequency, end_frequency, threshold, band_power]]]

Example:

```
>>SetBandListOutOfBand([100, 250, -20])
>>GetResultOutOfBandPower()
>>[[1,False, [[False, [[100, 250.0, -20.0, -25.180790459224692]]], [False, [[100, 250.0, -20.0, -25.180790459224692]]]
      [2, False, [[False, [[100, 250.0, -20.0, -25.180790459224692]]],  [False, [[100, 250.0, -20.0, -25.1807904592246
>>GetResultOutOfBandPower(1)
>>[False, [[False, [[100, 250.0, -20.0, -25.180790459224692]]],  [False, [[100, 250.0, -20.0, -25.180790459224692]]],
```

2.19.2.51 `GetResultImpulseNoise` (**slotNo** = 0)

Description

This function gets the impulse noise results from the last audio analysis run.

Parameters:

← **slotNo Integer** : (optional) By default, function acts on all reserved slots, unless this parameter is used.

Returns:

List: A list of impulse noise results, each of which consists [slot_num, overall_-status(True/False), impulse_noise_list] where overall_status is True if all of the results in impulse_noise_list are True, otherwise False, and impulse_noise_list is [result(True if no impulse noise detected, otherwise False), num_impulses, [top_5_impulse, top_5_impulse, top_-5_impulse, top_5_impulse, top_5_impulse]]

Example:

```
>>GetResultImpulseNoise()
>>[[1,False, [[False, 22.0, [-57.834483053106744, -57.774108360047805, -57.644864977277592, -55.51915255490313, -55.19
      [2,False, [[False, 22.0, [-57.834483053106744, -57.774108360047805, -57.644864977277592, -55.51915255490313, -55
>>GetResultImpulseNoise(1)
>>[False, [[False, 22.0, [-57.834483053106744, -57.774108360047805, -57.644864977277592, -55.51915255490313, -55.19632
```

2.20 Job Configuration API

These commands control the job creation and configuration.

Functions

- GetJobConfigurator

2.20.1 Detailed Description

These commands control the job creation and configuration.

This allows a user to submit jobs for remote execution on a StormTest server. This is a powerful feature which leverages the fact that each StormTest facility has a central database which holds the configuration of all servers/boxes in the facility and also has a central subversion repository that stores test scripts and reference images.

A user can use this API to easily upload test scripts to the central repository, and then submit jobs to slots/boxes in the facility to run those tests. The tests will be scheduled at the earliest possible time and the user can then check the status of the test run at a later time. All logs generated by the test are then available for viewing via an ftp url.

Access to this functionality is now provided via the StormTest Developer Suite application. This API is retained for legacy reasons.

NOTE: All these functions require the SetMasterConfigurationServer() call to have been made prior to their use. Failure to do this will result in a Stormtest exception being raised.

2.20.2 Function Documentation

2.20.2.1 `GetJobConfigurator` ()

Description

This function gets the object to configure the jobs for remote execution on the StormTest server. See the "Remote Job Dispatcher Guide" for more details on how to use this API.

Parameters:

None

Returns:

jobConfig **Object:** Job configurator object.

Example:

```
>>SetMasterConfigurationServer("master_server")
>>jobCfg = GetJobConfigurator()
```

2.21 Global Variables

There are a few global variables defined.

Variables

- STORMTEST_PUBLIC = stormtest_client.public_area
- STORMTEST_PRIVATE = stormtest_client.private_area
- _use_legacy_server_colors = False
- AllowIgnoredScreens = False

2.21.1 Detailed Description

There are a few global variables defined.

Most are read only and set by the API at start up. Some are writable. Writing to global variables is usually discouraged but occasionally it is necessary

2.21.2 Variable Documentation

2.21.2.1 STORMTEST_PUBLIC = stormtest_client.public_area

Description

When test scripts are written to run under the StormTest client daemon, these variables can be used to indicate if a reference image is to be found in the public part of the repository or in the private part. This allows the user to write a test script that will be uploaded to his private area but which relies on reference images that are in the public area of the subversion repository. The default location for image files can be set using the SetDirectories() command.

Example 1:

```
>>if STORMTEST_PUBLIC:    # Running under client daemon
>>    refImgDir = os.path.join(STORMTEST_PUBLIC,"path","to","refImages")
>>else:                   # Running interactively
>>    refImgDir = os.path.join("relativePath","onLocalPC","refImages")
>>
>>SetDirectories("logs", refImgDir) # logDir is ignored when running under client daemon
>>
>>result = CompareImage("Playback.bmp",None,None,10) # Searches for image in Public area
```

A user can also override this setting for individual calls to CompareImage() or CompareIcon() by passing in the full path to the image, including the PUBLIC/PRIVATE directory.

Example 2:

```
>>privateImage = os.path.join(STORMTEST_PRIVATE,"path","to","refImages","Playback.bmp")
>>
>>result = CompareImage("Playback.bmp",None,None,10) # Searches for image in Public area due to the
>>                                                   # SetDirectories() call above
>>
>>result = CompareImage(privateImage,None,None,10) # Searches for image in private area
```

2.21.2.2 STORMTEST_PRIVATE = stormtest_client.private_area

Description

See the STORMTEST_PUBLIC description

2.21.2.3 _use_legacy_server_colors = False

Description

Prior to version 3.0, there was a bug where a script called CaptureImageEx after calling ReserveSlot(..., videoFlag=False) in that the returned image would have a different color balance compared to calling CaptureImageEx after calling ReserveSlot(..., videoFlag=True). This has been corrected in version 3.0 and both now return an image with the same color balance. The effect was not noticable with the naked eye but CompareColorEx() could show it. For any user who has a large number of scripts using ReserveSlot(..., videoFlag=False) and CompareColorEx() then it may be useful to restore the previous behavior. Do this by setting _use_legacy_server_colors = True **before** calling ConnectToServer(). The setting will be used for the whole script.

2.21.2.4 AllowIgnoredScreens = False

Description

Some 3.0 beta users relied upon the ValidateNavigatorScreen() validating an ignored screen. This was a bug that has been fixed prior to 3.0 release (ignored screens are now fully ignored). However, to restore the original behavior, this flag can be set to True. It must be changed prior to opening a Navigator. The flag affects **all** navigators opened after changing the flag. Changing it while a Navigator is open is undefined behavior - it probably won't do anything but the behavior **cannot** be relied upon in any way (including crashing the system). It should be changed only when all navigators are closed. The recommendation is that it is part of the startup code of the script.

2.21.3 Function Parameters

This section gives more detail on some of the parameters passed to functions in the StormTest API.

Parameter Formats

- Serial Port Setup Parameters
- Returned Serial Port Data
- Returned CheckSlots data
- Debug Log Information
- Supported Video Formats
- Multi-dimensional array of results
- StormTestImageObject
- NamedRegion
- NamedColor
- NamedString
- NamedImage
- NamedScreenDefinition
- Languages supported by the OCR engine

2.21.4 Detailed Description

This section gives more detail on some of the parameters passed to functions in the StormTest API. It also explains and gives examples of the returned data from many of the StormTest API functions.

2.21.5 Serial Port Setup Parameters

Description

These are the allowable serial port parameters and are constructed as part of an array.

These are the returned values:

[BAUDRATE, DATABITS, PARITIES, STOPBITS, FLOWCTRL, PREFIX]

Example:

```
[19200,8,"none",1,"none"]
```

Parameters:

BAUDRATE Integer

Note:

The following values are valid:

```
50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 19200 38400 57600 115200 230400
```

Parameters:

DATABITS Integer

Note:

The following values are valid:

```
5 6 7 8
```

Parameters:

PARITIES String

Note:

The following values are valid:

```
"none" "even" "odd"
```

Parameters:

STOPBITS Integer

Note:

The following values are valid:

```
1 2
```

Parameters:

FLOWCTRL String

Note:

The following values are valid:

```
"none" "soft" "hard" "both"
```

Parameters:

PREFIX String: (optional) This value will prefix the log filename if the prefix is used.

2.21.6 Returned Serial Port Data

Description

This is the data returned for each slot by the SendCommandViaSerial() function.

These are the returned values:

[SLOTNUMBER, STATUS, EXPECTSTRING, LOG]

Parameters:

SLOTNUMBER Integer: The slot number the data is returned on.

STATUS Bool: True means the expected string was found on the serial output. False means there was a timeout before a match was found.

EXPECTSTRING String: The expected string.

LOG String: An extract from the serial log. The extract starts from the point that the command was sent to the set-top box, and continues until the line where the expected string was found.

Example:

```
>>ret=SendCommandViaSerial("4","TEST PASSED",60)
>>print ret
>>[(2, False, 'TEST PASSED', 'CMD> 4\n'),
>> (2, True, 'TEST PASSED', 'CMD> 4\nSome test output\nMore testoutput\nLine of text\
            containing TEST PASSED and some text until a newline character\n')]
```

2.21.7 Returned CheckSlots data

Description

These are the parameters returned for each slot by the CheckSlots() function.

These are the returned values:

[STATUS, USERNAME, DESCRIPTION]

Parameters:

STATUS Integer: The status for the index.

USERNAME String: The name of the user who reserved the slot as defined by the system.

DESCRIPTION String: A user defined string passed in by ConnectToServer.

Example:

```
# Return value structure (16 entries in the array)
[
    [1, 'Baggio', 'Stress Test']
        ...
        ...
    [0, ' ', ' ']
]

>>ret=CheckSlots()
>>for i in range(16):
>>  if ret[i][0] == 1:  print "Slot",i,"is reserved"
>>  else:               print "Slot",i,"is available"
```

Note:

These are the valid returned slot indicator values:

- 0: The slot is free.
- 1: The slot is reserved.
- 2: The slot is available but powered off.

2.21.8 Debug Log Information

Description

These are the parameters returned for each slot by the GetObserveLog() function.

These are the returned values:

[SLOTNUMBER, LOGFILESIZE, FILENAME]

Parameters:

SLOTNUMBER Integer: The slot that created the log.

LOGFILESIZE Integer: The log file size. If this value is 0, it indicates that the log file is empty.

FILENAME String: The log file name and path.

Example:

```
>>print GetObserveLog()
>>[(11, 0L, 'C:\\test\\Prefix_11_Timestamp_mon.log'),     # 0 bytes in the monitor log
                                                           # for slot 11

>> (12, 25L, 'C:\\test\\Prefix_12_Timestamp_mon.log')]     # 25 bytes in the monitor log
                                                           # for slot 12
```

2.21.9 Supported Video Formats

- "CIF" CIF, 352 x 288 video resolution
- "QCIF" Quarter CIF, 176 x 144 video resolution
- "2CIF" 2xCIF, 704 x 288 video resolution
- "DCIF" Double CIF, 528 x 384 video resolution
- "4CIF" 4xCIF, 704 x 576 video resolution
- For HD servers, the return is of the form "w x h" where w is the width in pixels and h is the height in pixels. For 4K servers, the return is as for HD but when setting the resolution the following values are supported:
- "4CIF" (704 x 576)
- "CIF" (352 x 288)
- "QCIF" (176 x 144)
- "704x576" (you can use "4CIF" or "704x576" to achieve this resolution)
- "352x288" (you can use "CIF" or "352x288" to achieve this resolution) = "176x144" (you can use "QCIF" or "176x144" to achieve this resolution)
- "640x480"
- "720x480"
- "720x576"
- "800x600"
- "1280x720"
- "1920x1080"
- There should be no spaces in the string.

2.21.10 Multi-dimensional array of results

Description

This return type is used by a number of different functions in the StormTest API. It is a flexible return type that allow information on multiple slots to be returned to the test. The return type is structured as an array of tuples. Each tuple consists of the slot number, the result of the function (True/False) and a filename.
The filename can represent a number of things including a screen capture filename (for CompareImage()) or a video log filename (for StartVideoLog()). If there is no file name for that instance of the return value, then *None* is returned in place of a filename.
If the user passes a *slotNo* to the called function, then the return value will be one tuple, rather than an array of tuples.
This type of return value is used by:

- CompareImage()
- CompareIcon()
- CaptureScreen()
- StartVideoLog()
- StopVideoLog()
- DetectMotion()

These are the returned values:

[SLOTNUMBER, STATUS, FILENAME]

Parameters:

SLOTNUMBER Integer: The slot that this status relates to.

STATUS Bool: Indicator as to success or failure of the called function. The actual meaning depends on the specific function called.

FILENAME String: The name of a file, including the path to it.

Example when *slotNo* is **not** passed to function:

```
# Return value structure
[
  (1, False, 'filename.bmp'),   # Index 0, result from slot 1 fail
  (9, True, None)               # Index 1, result from slot 9 success
]

# Example of accessing the return values:
>>ret=DetectMotion(None, 10)
>>for slot, result, file in ret
>>  if result == False:
>>      print "Motion not detected on slot",slot
```

Example when *slotNo* **is** passed to function:

```
# Return value structure
[1, False, 'filename.bmp']      # result from slot 1 fail

# Example of accessing the return value:
>>ret=DetectMotion(None, 10, slotNo=1)
>>if ret[1] == False:
>>  print "Motion not detected on slot",ret[0]
```

2.21.11 StormTestImageObject

Description

This is the returned object type from a number of image comparison and capture functions. The returned type is a object of class StormTestImageObject. In Release 3.1 and later the object can be constructed from a PIL (Python Imaging Library) image. The StormTestImageObject has a finalizer, however, you should call Close() as soon as you are finished using the image - HD images in particular can be large and keeping them in memory longer than necessary will risk an out of memory error from Python. This type of return value is used by:

- CaptureImageEx()
- WaitImageMatch()
- WaitImageNoMatch()
- WaitIconMatch()
- WaitIconNoMatch()

The following functions are available within this class:

StormTestImageObject(img)

The constructor allows you to construct a StormTestImageObject from another object. This other object must be a Python Imaging Library (PIL) image object.

Parameters:

← **img PIL** Image : A Python Imaging Library (PIL) image object. The PIL image must be in RGBA format or capable of being converted to RGBA format. An exception will be thrown if the image is not a valid PIL Image. The PIL image is not altered so after the cosntructor there will be 2 images in memory.

Returns:

StormTestImageObject : The completely constructed StormTestImageObject.

Note:

You can also create a StormTestImageObject from a file using the static method FromFile.

Save(filename, jpegQuality)

This will save a file to the log file **and** also to the log folder output (or a path relative to that folder). If the output is just a file name then the file will be in the log folder. If a relative path is given then **2 copies** of the file are produced: one for the log file and one where the script has directed. Under no circumstances will an existing file be overwritten. A new name will be made for the image. This is to preserve the integrity of the log file.

Parameters:

filename String: The filename to save the image as. It will be saved relative to the output directory so this must not be an absolute path

jpegQuality Integer: The jpeg quality for a saved jpeg image, range is 1 - 100.

Returns:

Bool true/false

FromFile(filePath)

This is a static method to return a StormTestImageObject from a file path. It will raise an exception if the file path cannot be read, does not exist, or the contents are not a valid image file. The image file must be PNG, JPEG, BMP or TIFF format.

Parameters:

filePath String: The path to the file to load. It may be relative or absolute. If an absolute path is not given then the path is assumed to be relative to the imgFolder set by SetDirectories()

Returns:

StormTestImageObject : The created object from the file.

Note:

FromFile is available in release 3.3.3.

GetLastSavedName()

This function returns the last saved file name. As the Save() method does not return the filename, you may need this to know just where the file was saved.

Parameters:

None

Returns:

String : the full path of the file that was saved by the most recent Save() call

SaveToFile(filename, jpegQuality)

This will save a file to the hard disk. It will not save the file into the log file nor the log folder. It will **overwrite** any existing file without warning. The path may be relative to the output folder or an absolute path.

Parameters:

filename String: The filename to save the image to.

jpegQuality Integer: The jpeg quality for a saved jpeg image, the range is 1 - 100

Returns:

Bool true/false

Crop(rectangle)

Parameters:

rectangle Array: Area for cropping

Returns:

see StormTestImageObject

ToPILImage(retainRawImage = False)

Converts the image to a Python Imaging Library (PIL) Image

Parameters:

retainRawImage bool: If true, the raw image will remain in StormTestImageObject. The default is to close it and free memory on the assumption that the script will want to do further processing using PIL.

IsValid()

This function checks whether the object constains a valid image. This is one method that can be safely called after Close() - it will return False

Parameters:

None

Returns:

Boolean : True if the StormTestImageObject is a valid image, otherwise False.

Clone()

This clones a StormTestImageObject returning an exact copy.

Parameters:

None

Returns:

StormTestImageObject: the cloned image. The original image is unmodified.

Offset(offset)

This offsets a StormTestImageObject by the specified amount. Even if the offset is (0,0) a copy of the image is returned.

Parameters:

← **offset List:** Offset as 2 integers for the x, y direction of the offset.

Returns:

StormTestImageObject: the offset image. The original image is unmodified.

Transform(converter)

Transforms an image using a CoordConverter object. This function is provided when you have chosen to capture an image on a slot where a transform is needed. So that you can examine the raw image, CaptureImageEx() will not apply the slot transform. However, if you now need the transformed image, for example to process the image with some custom filtering prior to an OCR, you can use the Transform() method to transform the image. This is only availble in version 3.2 of StormTest

Parameters:

← **converter CoordConverter** : the transformation to apply. The final size of the image will be the WorldSize of this converter.

Returns:

StormTestImageObject : The transformed image. The original image is not modified by the transform.

Close()

Parameters:

None,: Frees image memory earlier than Garbage collector would. Highly advised if using a lot of images. This must be the last call used on the object (other than IsValid)

Example using contructor:

```
import Image
pil = Image.open("myfile.png")
stormtestImage = Imaging.StormTestImageObject(pil)
```

Example using FromFile()

```
stormtestImage = Imaging.StormTestImageObject.FromFile("myfile.png")
```

Example Using Save() and GetLastSavedName

```
res = CaptureImageEx (None, None)
slot,res,image,filename = res[0]
image.Save("imgFromMemory.png")
print image.GetLastSavedName()
>>>C:\tests\sample_20140721_151525_66\imgFromMemory.png.png
image.Save("imgFromMemory.png")
print image.GetLastSavedName()
>>>C:\tests\sample_20140721_151525_66\imgFromMemory_2.png.png
```

Example Using SaveToFile()

```
res = CaptureImageEx (None, None)
slot,res,image,filename = res[0]
image.SaveToFile("c:\\myfolder\\imgFromMemory.png")
```

Example Using Crop()

```
rect = [100,100,100,100]
res = CaptureImageEx (None, None)
slot,res,image,filename = res[0]
newImage = image.Crop(rect)
```

Example using Close and IsValid

```
result = CaptureImageEx(None, None)
slot, status, image, filename = result[0]
print image.IsValid()
>>>True
image.Close()
print image.IsValid()
>>>False
```

Example using Transform:

```
# example where we capture at HD but we know the image has been upscaled so we
# wish to downscale it back to SD and supply asymetric scaling and offset.
converter = CoordConverter()
converter.SetFromRectangles([0,0,1920,1080], [5,5,680,568])
converter.WorldSize = (704, 576)
res = CaptureImageEx (None, None)
slot,res,image,filename = res[0]
im2 = image.Transform(converter)
```

Example using Clone and Offset:

```
res = CaptureImageEx (None, None)
slot,res,image,filename = res[0]
newImage = image.Offset((4,-2))
print image.IsValid(), newImage.IsValid()
>>>True, True
image.Close()
print image.IsValid(), newImage.IsValid()
>>>False, True
```

2.21.12 NamedRegion

Description

This is the returned object from the FindNamedRegion function. It is a shallow wrapper object that can be used in many calls instead of a rectangle. The constructor is not documented as this class is not intended that user scripts should create instances of NamedRegion.

The following functions/properties are available within this class:

Name

This is the name of the object. It is always just the name even for absolute objects.

IsAbsolute

This is bool, true if the object is an absolute object, created from an absolute path

LookupBySlot(slotNo, forceDutSpecific=False)

This will find the region as a tuple of the named region for the specified slot. This method follows the following steps: This will find the rectangle value for the specified slot from a NamedRegion. This method follows the following steps - the first look up that succeeds terminates the process.

1. Find DUT model for specified slot.
2. If NamedRegion is absolute find DUT specific model region in the explicit path.
3. If NamedRegion is absolute find the generic region in the explicit path.
4. If NamedRegion is relative, search the paths for the slotNo looking for DUT specific region.
5. If NamedRegion is relative, search the paths for the slotNo looking for generic region.
6. If NamedRegion is relative, search the 'all slots' paths looking for the DUT specific region.
7. If NamedRegion is relative, search the 'all slots' paths looking for the generic region. If the region is not found, a StormTestExceptions exception is raised. For some API calls, slotNo will be set to None. In that case steps 1, 2, 4, 5, 6 above are skipped. The LookupBySlot is cached - later calls with the same slot number will always return the same value and will not incur network overhead in contacting the database.

Parameters:

← **slotNo Integer** : The slot number for which the lookup occurs. 0 is not permitted but None is. A slotNo of None will only look for an absolute region or one in the 'all slots' paths and DUT specific lookup is ignored.

← **forceDutSpecific Boolean** : Optional. When true, forces a DUT specific lookup so that the generic version is never returned. Steps 3, 5 and 7 are omitted. This parameter was introduced in version 3.2 of StormTest

Returns:

Tuple : The 4 values of the rectangle as a tuple

Example:

```
r = FindNamedRegion("/my projects/title")
print r.LookupBySlot(4)
>>>(0,0,100,67)
print r.LookupBySlot(6)
>>>(100,10,80,20)
```

LookupByDut(dutModelId)

This will find the rectangle value for the specified model from a NamedRegion. This method follows the following steps - the first look up that succeeds terminates the process.

1. If NamedRegion is absolute find DUT specific model region in the explicit path.
2. If NamedRegion is relative, search the 'all slots' paths looking for the DUT specific region. If the region is not found, a StormTestExceptions exception is raised.

Parameters:

← **dutModelId Integer** or String : The DUT model, either as an integer id from determined by ListDutModels() or as a string.

Returns:

Tuple : The 4 values of the rectangle as a tuple

Example:

```
r = FindNamedRegion("/my projects/title")
print r.LookupByDut("My DUT")
>>>(0,0,100,67)
```

Refresh()

This will clear the cache for the object (for any and all slots that have been used)

Parameters:

None

Returns:

None

2.21.13 NamedColor

Description

This is the returned object from the FindNamedColor function. It is a shallow wrapper object that can be used in many calls instead of a color. When this happens, the tolerance, flatness and peak error parameters of the associated call are retrieved from the NamedColor and any values passed in to the function concerned are ignored. Most functions do not require the tolerance, flatness or peak error in the sense that they are marked as optional parameters. The constructor is not documented as this class is not intended that user scripts should create instances of NamedColor.

The following functions/properties are available within this class:

Name

This is the name of the object. It is always just the name even for absolute objects.

IsAbsolute

This is bool, true if the object is an absolute object, created from an absolute path

LookupBySlot(slotNo, forceDutSpecific=False)

This will find the color value for the specified slot from a NamedColor. This method follows the following steps - the first look up that succeeds terminates the process.

1. Find DUT model for specified slot.
2. If NamedColor is absolute find DUT specific model color in the explicit path.
3. If NamedColor is absolute find the generic color in the explicit path.
4. If NamedColor is relative, search the paths for the slotNo looking for DUT specific color.
5. If NamedColor is relative, search the paths for the slotNo looking for generic color.
6. If NamedColor is relative, search the 'all slots' paths looking for the DUT specific color.
7. If NamedColor is relative, search the 'all slots' paths looking for the generic color. If the color is not found, a StormTestExceptions exception is raised. For some API calls, slotNo will be set to None. In that case steps 1, 2, 4, 5, 6 above are skipped. The LookupBySlot is cached - later calls with the same slot number will always return the same value and will not incur network overhead in contacting the database.

Parameters:

← **slotNo Integer** : The slot number for which the lookup occurs. 0 is not permitted but None is. A slotNo of None will only look for an absolute color or one in the 'all slots' paths and DUT specific lookup is ignored.

← **forceDutSpecific Boolean** : Optional. When true, forces a DUT specific lookup so that the generic version is never returned. Steps 3, 5 and 7 are omitted.

Returns:

Tuple : The value of the color as a tuple of (color tuple, tolerance tuple, flatness, peak error)

Example:

```
r = FindNamedColor("/my projects/highlite color")
print r.LookupBySlot(4)
>>>((255,0,0), (8,8,8), 97.5, 1.0)
print r.LookupBySlot(6)
>>>((0,255,0), (20,20,20), 90, 2.0)
```

LookupByDut(dutModelId)

This will find the color value for the specified model from a NamedColor. This method follows the following steps - the first look up that succeeds terminates the process.

1. If NamedColor is absolute find DUT specific model color in the explicit path.
2. If NamedColor is relative, search the 'all slots' paths looking for the DUT specific color. If the color is not found, a StormTestExceptions exception is raised.

Parameters:

← **dutModelId Integer** or String : The DUT model, either as an integer id from determined by ListDutModels() or as a string.

Returns:

Tuple : The value of the color as a tuple of (color tuple, tolerance tuple, flatness, peak error)

Example:

```
r = FindNamedColor("/my projects/highlite color")
print r.LookupByDut("My DUT")
>>>((255,0,0), (8,8,8), 97.5, 1.0)
```

Refresh()

This will clear the cache for the object (for any and all slots that have been used)

Parameters:

None

Returns:

None

2.21.14 NamedString

Description

This is the returned object from the FindNamedString function. It is a shallow wrapper object that can be used in many calls instead of a string. The constructor is not documented as this class is not intended that user scripts should create instances of NamedString.

The following functions/properties are available within this class:

Name

This is the name of the object. It is always just the name even for absolute objects.

IsAbsolute

This is bool, true if the object is an absolute object, created from an absolute path

LookupBySlot(slotNo, forceDutSpecific=False)

This will find the string value for the specified slot from a NamedString. This method follows the following steps - the first look up that succeeds terminates the process.

1. Find DUT model for specified slot.
2. If NamedString is absolute find DUT specific model string in the explicit path.
3. If NamedString is absolute find the generic string in the explicit path.
4. If NamedString is relative, search the paths for the slotNo looking for DUT specific string.
5. If NamedString is relative, search the paths for the slotNo looking for generic string.
6. If NamedString is relative, search the 'all slots' paths looking for the DUT specific string.
7. If NamedString is relative, search the 'all slots' paths looking for the generic string. If the string is not found, a StormTestExceptions exception is raised. For some API calls, slotNo will be set to None. In that case steps 1, 2, 4, 5, 6 above are skipped. The LookupBySlot is cached - later calls with the same slot number will always return the same value and will not incur network overhead in contacting the database.

Parameters:

← **slotNo Integer** : The slot number for which the lookup occurs. 0 is not permitted but None is. A slotNo of None will only look for an absolute string or one in the 'all slots' paths and DUT specific lookup is ignored.

← **forceDutSpecific Boolean** : Optional. When true, forces a DUT specific lookup so that the generic version is never returned. Steps 3, 5 and 7 are omitted.

Returns:

String : The value of the string

Example:

```
r = FindNamedString("/my projects/title")
print r.LookupBySlot(4)
>>>'My super DUT'
print r.LookupBySlot(6)
>>>'Also a super DUT'
```

LookupByDut(dutModelId)

This will find the string value for the specified model from a NamedString. This method follows the following steps - the first look up that succeeds terminates the process.

1. If NamedString is absolute find DUT specific model string in the explicit path.
2. If NamedString is relative, search the 'all slots' paths looking for the DUT specific string. If the string is not found, a StormTestExceptions exception is raised.

Parameters:

← **dutModelId Integer** or String : The DUT model, either as an integer id from determined by ListDutModels() or as a string.

Returns:

String : The value of the string

Example:

```
r = FindNamedString("/my projects/title")
print r.LookupByDut("My DUT")
>>>'My good DUT'
```

Refresh()

This will clear the cache for the object (for any and all slots that have been used)

Parameters:

None

Returns:

None

2.21.15 NamedImage

Description

This is the returned object from the FindNamedImage function. It is a shallow wrapper object that can be used in many calls instead of a StormTestImageObject. The constructor is not documented as this class is not intended that user scripts should create instances of NamedImage.

The following functions/properties are available within this class:

Name

This is the name of the object. It is always just the name even for absolute objects.

IsAbsolute

This is bool, true if the object is an absolute object, created from an absolute path

LookupBySlot(slotNo, forceDutSpecific=False)

This will find the image value for the specified slot from a NamedImage. This method follows the following steps - the first look up that succeeds terminates the process.

1. Find DUT model for specified slot.
2. If NamedImage is absolute find DUT specific model image in the explicit path.
3. If NamedImage is absolute find the generic image in the explicit path.
4. If NamedImage is relative, search the paths for the slotNo looking for DUT specific image.
5. If NamedImage is relative, search the paths for the slotNo looking for generic image.
6. If NamedImage is relative, search the 'all slots' paths looking for the DUT specific image.
7. If NamedImage is relative, search the 'all slots' paths looking for the generic image. If the image is not found, a StormTestExceptions exception is raised. For some API calls, slotNo will be set to None. In that case steps 1, 2, 4, 5, 6 above are skipped. The LookupBySlot is cached - later calls with the same slot number will always return the same value and will not incur network overhead in contacting the database.

Parameters:

← **slotNo Integer** : The slot number for which the lookup occurs. 0 is not permitted but None is. A slotNo of None will only look for an absolute image or one in the 'all slots' paths and DUT specific lookup is ignored.

← **forceDutSpecific Boolean** : Optional. When true, forces a DUT specific lookup so that the generic version is never returned. Steps 3, 5 and 7 are omitted.

Returns:

Tuple : The StormTestImageObject, width and height of the image.

Example:

```
r = FindNamedImage("/my projects/logo")
print r.LookupBySlot(4)
>>>(<ClientAPI.Imaging.StormTestImageObject object>, 48, 24)
print r.LookupBySlot(6)
>>>(<ClientAPI.Imaging.StormTestImageObject object>, 64, 32)
```

LookupByDut(dutModelId)

This will find the image value for the specified model from a NamedImage. This method follows the following steps - the first look up that succeeds terminates the process.

1. If NamedImage is absolute find DUT specific model image in the explicit path.
2. If NamedImage is relative, search the 'all slots' paths looking for the DUT specific image. If the image is not found, a StormTestExceptions exception is raised.

Parameters:

← **dutModelId Integer** or String : The DUT model, either as an integer id from determined by ListDutModels() or as a string.

Returns:

Tuple : The StormTestImageObject, width and height of the image.

Example:

```
r = FindNamedImage("/my projects/logo")
print r.LookupByDut("My DUT")
>>>(<ClientAPI.Imaging.StormTestImageObject object>, 48, 24)
```

Refresh()

This will clear the cache for the object (for any and all slots that have been used)

Parameters:

None

Returns:

None

2.21.16 NamedScreenDefinition

Description

This is the returned object from the FindNamedScreenDefinition function. It is a shallow wrapper object. The constructor is not documented as this class is not intended that user scripts should create instances of NamedScreenDefinition.

The following functions/properties are available within this class:

Name

This is the name of the object. It is always just the name even for absolute objects.

IsAbsolute

This is bool, true if the object is an absolute object, created from an absolute path

LookupBySlot(slotNo, forceDutSpecific=False)

This will find the screen definition value for the specified slot from a NamedScreenDefinition. This method follows the following steps - the first look up that succeeds terminates the process.

1. Find DUT model for specified slot.
2. If NamedScreenDefinition is absolute find DUT specific model ScreenDefinition in the explicit path.
3. If NamedScreenDefinition is absolute find the generic ScreenDefinition in the explicit path.
4. If NamedScreenDefinition is relative, search the paths for the slotNo looking for DUT specific ScreenDefinition.
5. If NamedScreenDefinition is relative, search the paths for the slotNo looking for generic ScreenDefinition.
6. If NamedScreenDefinition is relative, search the 'all slots' paths looking for the DUT specific ScreenDefinition.
7. If NamedScreenDefinition is relative, search the 'all slots' paths looking for the generic ScreenDefinition. If the ScreenDefinition is not found, a StormTestExceptions exception is raised. For some API calls, slotNo will be set to None. In that case steps 1, 2, 4, 5, 6 above are skipped. The LookupBySlot is cached - later calls with the same slot number will always return the same value and will not incur network overhead in contacting the database.

Parameters:

← **slotNo Integer** : The slot number for which the lookup occurs. 0 is not permitted but None is. A slotNo of None will only look for an absolute color or one in the 'all slots' paths and DUT specific lookup is ignored.

← **forceDutSpecific Boolean** : Optional. When true, forces a DUT specific lookup so that the generic version is never returned. Steps 3, 5 and 7 are omitted.

Returns:

ScreenDefinition : The value of the ScreenDefinition

Example:

```
r = FindNamedScreenDefinition("/my projects/sdo1")
print r.LookupBySlot(4)
>>><stormtest.ClientAPI.ScreenDefinition object>
```

LookupByDut(dutModelId)

This will find the screen definition value for the specified model from a NamedScreenDefinition. This method follows the following steps - the first look up that succeeds terminates the process.

1. If NamedScreenDefinition is absolute find DUT specific model ScreenDefinition in the explicit path.
2. If NamedScreenDefinition is relative, search the 'all slots' paths looking for the DUT specific ScreenDefinition. If the ScreenDefinition is not found, a StormTestExceptions exception is raised.

Parameters:

← **dutModelId Integer** or String : The DUT model, either as an integer id from determined by ListDutModels() or as a string.

Returns:

ScreenDefinition : The value of the ScreenDefinition

Example:

```
r = FindNamedScreenDefinition("/my projects/sdo1")
print r.LookupByDut("My DUT")
>>><stormtest.ClientAPI.ScreenDefinition object>
```

Refresh()

This will clear the cache for the object (for any and all slots that have been used)

Parameters:

None

Returns:

None

2.21.17 Languages supported by the OCR engine

Description

The StormTest OCR engine is shipped with OCR dictionaries for the following languages. The user may improve the accuracy of the OCR engine by setting the OCR language to be one or more of the following languages.

Language	ISO 639-3 Code
Afrikaans	afr
Chinese (Mandarin)(See Note 1)	cmn
Czech	ces
Danish	dan
Dutch	nld
English	eng
Finnish	fin
Flemmish	vls
French	fra
German	deu
Greek	ell
Hebrew(See Note 1)	heb
Irish	gle
Italian	ita
Japanese(See Note 1)	jpn
Korean(See Note 1)	kor
Norwegian (Bokmål)	nob
Polish	pol
Portuguese	por
Russian	rus
Spanish	spa
Swedish	swe
Thai(See Note 1)	tha
Turkish	tur
Vietnamese(See Note 1)	vie

The user may specify more than one language dictionary to be used by the OCR engine.

Note:

1. An extra license will be required for these languages - contact StormTest support for details.

Example: Setting default dictionaries for the StormTest OCR engine

```
# Tell the OCR engine to use the German dictionary.
>>result = OCRSetLanguage('deu')
>>print result
>>True

# Tell the OCR engine to use the Spanish, Portugese and French dictionaries.
>>result = OCRSetLanguage(('spa','por','fra'))
>>print result
>>True

# Tell the OCR engine to use an invalid dictionary - this will raise an exception
>>result = OCRSetLanguage('abc')
 Exception in user code:
 StormTestExceptions: Invalid language: [abc]
```

2.21.18 Coordinate Transformations with Screen definition Objects

Description

When using screen definition objects, there is potential compexity in determining what the final coordinates used will be. It is affected by the DesignSize property of the object, the DesignSize property of the screen definition, any CoordConverters attached to the screen definition object and the CoordConverter set as the slot transformation in SetDesignToWorldTransform.

This note explains how the system calculates the coordinates. The aim is to "do the right thing" and act sensibly. Here, SDO means Screen Definition Object and converter refers to an instance of CoordConverter. design size is taken to mean the DesignSize property.

When an SDO is executed, it has multiple regions which are really the things that are executed not the SDO which is a container. As an example, we'll discuse how a ColorRegion is executed. There is a rectangle in the ColorRegion object - this is a set of numbers and we already have an image captured. The first step is to find the design size of those coordinates. It does this by looking in the Design Size property of the color test. If this is null, it looks in the design size of the parent SDO, if that is null, the search continues up the chain. The search stops on the first non null Design Size value. If we reach the end of the chain without finding a design size, the next step is to look at the slot transform that has been set. If that has a design size then we use that as the design size of the color region test. If no transform has been set (or a transform without a design size), we use the size of the image as the design size.

At this stage, we have a non zero design size for the coordinates. This is guaranteed. We call this the "Test Design Size" internally. If the slot transform has been set with a design size that is not equal to the Test Design Size then we convert the color region coordinates to the slot transform design size. To do this we search the coord converter list of the color region's parent SDO for a converter where that converters design size and world size match the Test Design Size and the slot transform's design size respectively. If no such converter exists, we walk up the parent list until we find one that matches. Finally, if none match, we make one that is a simple linear conversion. We use this to convert the coordinate of the color region into coordinates in the slot transform size. This produces a rectangle in the "Final Design Size" If the slot transform has not been set or has a zero design size, we skip this step and final design size is the same as test design size and the rectangle is unchanged.

We now have the rectangle in the final design size that we need. If the slot transform has a non zero design size, we simply use the slot transform. If the rotation is zero, we convert the coordinates and then examine the appropriate area of the captured image. If the rotation is non zero, we modify the image not the coordinates and then use the design size coordinates in the converted image.

If the slot transform has an empty design size then we search the color region's parent coordinate converter list for a converter where the design size and world size match the final design size and captured image size respectively. As before we search up the chain. There is one little quirk here: if no matching coordinate converter is found AND the final design size == captured image size then we do nothing, else we make a linear converter. This allows a user to specify a custom transform for 1920x1080 -> 1920x1080 where the design size and captured image size are both 1920x1080.

If we have a converter, we proceed in the same manner as if a slot transform is set: we check for rotation and either convert the coordinates or the image. One point to note is that we ignore rotation for detect motion - this is because the whole transform is intended for use with TV tester and rotation angles are <1 degree and at that stage the error on a detect motion rectangle is minimal.

3 Enum Documentation

3.1 AlgorithmDescriptor Enum Reference

Public Member Functions

- __init__

Public Attributes

- Name
- Vendor
- Id
- Version
- Aliases

Description

The object defines an Sdo Extension algorithm description. It is NOT the algorithm itself but describes the algorithm in enough detail for a script writer to know if they wish to use the algorithm (if returned from the system) as well as being able to define an algorithm to use (if it is desired to create a region using a specific algorithm). This object is available in Release 3.4

3.1.1 Member Function Documentation

3.1.1.1 __init__ (self, id = None, name = None, vendor = None, version = None)

Description

The constructor makes an AlgorithmDescriptor. Use this when you know in advance which algorithm to use. The constructor will check the algorithm is on the system and raise an exception if it cannot find the algorithm requested. If more than 1 algorithm matches the request an exception is raised. Omitting a parameter from the constructor sets that parameter to 'any' when searching for algorithms.

Parameters:

← **id String** : The id of the algorithm. If you want precise control over the algorithm selected, you should set the id to the intended value.

← **name String** : The name of the algorithm. This can change between versions and is an unreliable search criterion.

← **vendor String** : The vendor of the algorithm. This is not always specified and is an unreliable search criterion.

← **version String** : The version of the algorihm. This can be used with the id to precisely select an algorithm. There is no check for "later version" because the format of the version is unknown to StormTest.

Returns:

AlgorithmDescriptor : The AlgorithmDescriptor object

Example:

```
>>>algorithm = AlgorithmDescriptor("12453B27-4736-4EAA-BC92-655DE7716871")
>>>algorithm = AlgorithmDescriptor(name = "my algorithm", version = "V 1.0")
```

3.2 AudioChannel Enum Reference

Static Public Attributes

- string Left = 'Left'
- string Right = 'Right'

Description

AudioChannel Enum. Defines the valid values for audio channel.

Parameters:

Left : Left audio channel

Right : Right audio channel

3.3 AudioRegion Enum Reference

Public Member Functions

- __getattr__
- __setattr__
- __init__
- __del__
- __dir__
- Close
- Clone

Public Attributes

- Name
- Verify
- AudioThreshold
- Timeout
- Channel

Private Member Functions

- __checkObject
- __resetObject

Private Attributes

- __cache

Static Private Attributes

- __handle = None
- dictionary __cache = {}
- __closed = False

Description

This represents an audio test. It permits a screen to be verified to include or exclude audio. There is no 'region' of audio but for consistency, the AudioRegion keeps the naming and properties conventions of the other regions.

Parameters:

Name String : The name of the region - this is a human readable name.

Comment String: A general comment. It is not used by StormTest and is provided for the user.

DesignSize Tuple: This has no significance for audio.

Parent ScreenDefinition: The parent ScreenDefinition. It will be set on all objects returned from the StormTest API and should be set on objects passed into the StormTest API (but the API is tolerant of users who forget to set the Parent or choose not to set it).

Verify Boolean: A bool to indicate whether the region is for verification purposes (True) or just reading (False).

PassOnNoMatch Boolean: A bool, defaulting to False, which allows the reversal of the condition - that is, the 'pass' status is when there is no audio in the region. This property is ignored if Verify is False.

ReturnScreenImage ReturnScreen: An enumerated type to indicate whether the return should include the full screen at the end of audio detection. It is an enumeration of type ReturnScreen. The default is Inherit which means the image return will be controlled by the enclosing ScreenDefinition

AudioThreshold Integer : The audio level in dBu, above which the audio is considered 'present'. There is no default so user code must set a value.

Timeout Float : The maximum time in seconds to wait for audio to exceed the threshold. Where the PassOnNoMatch is False then this is the time to wait for audio to drop below the threshold. If Verify is False, then this is ignored and a simple measurement of the level occurs. The default value is None and user code must set a timeout.

Channel AudioChannel : For HD systems, this specifies the left or right audio channel. Ignored for Standard Definition systems

VerifyStatus Boolean: The output of the verification process. It will be True if this object's verification criteria passed, else False. If Verify is False then it is True as nothing can fail.

Image StormTestImageObject: This is a StormTestImageObject holding the image of the region used for motion detection. It will be the image that was visible at the end of the audio detection. It can be None. The table below summarises when this property is not None

ReturnScreenImage value	Condition for Image to be not None
Always	Always
Never	Never
OnSuccess	VerifyStatus is True
OnFail	VerifyStatus is False
Inherit	Depends on enclosing ScreenDefinition. If no such parent, then as per OnFail

Parameters:

ActualAudio Float : The actual audio level in dBu. It will be None if no audio measurement took place. If Verify is True and VerifyStatus is True then this is the audio level that caused the verification to pass. This is independent of the value of PassOnNoMatch - the value is the valute that triggered the verification to be True. If the verification failed then this is the closest value to the passing threshold.

3.3.1 Member Function Documentation

3.3.1.1 __getattr__ (self, attr)

3.3.1.2 __setattr__(self, attr, value)

3.3.1.3 __checkObject (self) [private]

3.3.1.4 __resetObject (self, handle) [private]

3.3.1.5 __init__ (self, name = '', verify = True, audioThreshold = None, timeout = None, channel = 'Left')

Description

The constructor makes an AudioRegion object. The empty constructor AudioRegion() makes an object to detect audio on the left channel.

Parameters:

← **name String** : Sets the Name property of the constructed region

← **verify Boolean** : Sets the Verify property of the constructed region

← **audioThreshold Integer** : Sets the AudioThreshold property of the constructed region

← **timeout Float** : Sets the Timeout property of the constructed region

← **channel Integer** : Sets the Channel property of the constructed region

Returns:

AudioRegion : The AudioRegion object

Example:

```
>>region = AudioRegion("epg screen", audioThreshold=-20)
```

3.3.1.6 __del__ (self)

Description

The AudioRegion has a finalizer as a last resort clean up when the object goes out of scope. Normally the AudioRegion is part of a ScreenDefinition - in that case, you don't need to take any precautions over AudioRegion clean up - the owning ScreenDefinition manages it. If you create 'orphaned' AudioRegions, see the Close() method.

Parameters:

None

Returns:

None

3.3.1.7 __dir__ (self)

3.3.1.8 def Close (self)

Description

Cleans up the AudioRegion releasing any internal images. Do NOT call this if the AudioRegion is owned by a ScreenDefinition - the ScreenDefinition will manage the lifetime and cleanup of all owned objects. However, if you create an orphaned AudioRegion (either directly or by removing it from the ScreenDefinition) then you should call Close() when finished with the object.

Parameters:

None

Returns:

None

Example:

```
# This is somewhat contrived example which loads a ScreenDefinition from a file
# executes it to get a result and then 'orphans' the AudioRegion (assuming it
# is the first region in the screen definition
>>>sdo = ScreenDefinition()
>>>sdo.Load("file.stscreen")
>>>result = WaitScreenDefMatch(sdo, slotNo=4)[0]
>>>r = result.Regions[0]
>>>result.Regions = []    # now the result sdo has no regions but r is an AudioRegion and 'orphaned'
# do stuff with r
# this frees the potentially large image in r. Without Close() you would wait for Python to garbage collect it.
>>>r.Close()
```

3.3.1.9 def Clone (self)

Description

This clones an AudioRegion returning an exact copy.

Parameters:

None

Returns:

AudioRegion: the cloned regions. The original region is unmodified.

Example:

```
>>>r1 = AudioRegion(audioThreshold = -57)
>>>r2 = r1.Clone()
>>>print r2.AudioThreshold
>>>-57.00001
```

3.4 Color Enum Reference

Public Member Functions

- __init__

Public Attributes

- RGB
- Tolerances
- Flatness
- PeakError

Description

This represents a color object, combining the RGB elements, tolerances, flatness and peak error into one object This is available in release 3.4 and is only supported in a CustomRegion

Parameters:

RGB List : A List of 3 integers representing Red, Green, Blue color components

Tolerances List : A List of 3 integers representing the tolerance on red, green and blue components

Flatness Float : The flatness of the color in the range 0 to 100

PeakError Float : The peak error of the color in the range 0 to 100

3.4.1 Member Function Documentation

3.4.1.1 __init__ (self, color = [0, tolerances = [4, flatness = 98, peakError = 0)

3.5 ColorRegion Enum Reference

Public Member Functions

- __getattr__
- __setattr__
- __init__
- __del__
- __dir__
- Close
- Clone

Public Attributes

- Name
- Rect
- Verify
- Tolerance
- Flatness
- PeakError
- ReferenceColor

Private Member Functions

- __checkObject
- __resetObject

Private Attributes

- __cache

Static Private Attributes

- __handle = None
- dictionary __cache = {}
- __closed = False

Description

This represents a color comparison test.

Parameters:

Name String: The name of the region - this is a human readable name.

Comment String : A general comment. It is not used by StormTest and is provided for the user.

DesignSize Tuple: This is a List of 2 float values representing the width and height of the coordinate system used when designing the object. Given this information it is possible to combine objects designed at different times by different people at different resolutions and reuse the components. It effectively makes the embedded coordinates proportional coordinates.

Parent ScreenDefinition : The parent ScreenDefinition. It will be set on all objects returned from the StormTest API and should be set on objects passed into the StormTest API (but the API is tolerant of users who forget to set the Parent or choose not to set it).

Rect Tuple: The rectangle to use - a tuple / List of left, top, width, height as used elsewhere in the API. It can also be a _namedRegion object.

Verify Boolean: A bool to indicate whether the region is for verification purposes (True) or just reading (False).

PassOnNoMatch Boolean: A bool, defaulting to False, which allows the reversal of the condition - that is, the 'pass' status is when the color of the region does NOT match the color of the captured screen. This property is ignored if Verify is False.

ReturnScreenImage ReturnScreen: An enumerated type to indicate whether the return should include the cropped area of the screen used for the color comparison. It is an enumeration of type ReturnScreen. The default is Inherit which means the image return will be controlled by the enclosing ScreenDefinition

ReferenceColor List, Tuple or NamedColor : The reference color as a Red, Green, Blue list (all integers). This is ignored if Verify is False. Default is (0,0,0) (Black). This is the same as the color parameter of CompareColorEx() function.

Tolerance List : The tolerance on the color when comparing. It is a list of 3 integers. This is ignored if the ReferenceColor is a NamedColor object or if Verify is False. This is the same as the tolerances parameter of CompareColorEx() function. Default value (4,4,4)

Flatness Float : The required flatness of the color. This is ignored if the ReferenceColor is a NamedColor or if Verify is False. This is the same as the flatness parameter of the CompareColorEx function. Default value 98.

PeakError Float : The required peak error of the color. This is ignored if the ReferenceColor is a NamedColor or if Verify is False. This is the same as the peakError parameter of the CompareColorEx function. Default value 0.

VerifyStatus Boolean: The output of the verification process. It will be True if this object's verification criteria passed, else False. If Verify is False then it is True as nothing can fail.

Image StormTestImageObject: This is a StormTestImageObject holding the image of the region used for color comparison. It can be None. The table below summarises when this property is not None

ReturnScreenImage value	Condition for Image to be not None
Always	Always
Never	Never
OnSuccess	VerifyStatus is True
OnFail	VerifyStatus is False
Inherit	Depends on enclosing ScreenDefinition. If no such parent, then as per OnFail

Parameters:

ActualColor Tuple : The detected color as a typle of 3 integers. It will be None if no color comparison has occurred.

ActualFlatness Float : The detected flatness. It will be None if no color comparison has occurred.

ActualPeakError Float : The detected peak error. It will be None if no color comparison has occurred.

3.5.1 Member Function Documentation

3.5.1.1 __getattr__ (self, attr)

3.5.1.2 __setattr__ (self, attr, value)

3.5.1.3 __checkObject (self) [private]

3.5.1.4 __resetObject (self, handle) [private]

3.5.1.5 __init__ (self, name = '', rect = None, verify = True, referenceColor = (0,0, tolerance = (4,4, flatness = 98, peakError = 0)

Description

The constructor makes a ColorRegion object. The empty constructor ColorRegion() makes a region to compare the whole screen to black. All parameters to the constructor are optional

Parameters:

← **name String** : Sets the Name property of the constructed region

← **rect Tuple**, List or NamedRegion : Sets the Rect property of the constructed region

← **verify Boolean** : Sets the Verify property of the constructed region

← **referenceColor List**, Tuple or NamedColor : Sets the ReferenceColor property of the constructed region

← **tolerance List** : Sets the Tolerance property of the constructed region

← **flatness Float** : Sets the Flatness property of the constructed region

← **peakError Float** : Sets the PeakError property of the constructed region

Returns:

ColorRegion : The constructed ColorRegion object

Example:

```
>>region = ColorRegion("highlight area", (100,20,80,25), (255,255,0))
```

3.5.1.6 __del__ (self)

Description

The ColorRegion has a finalizer as a last resort clean up when the object goes out of scope. Normally the ColorRegion is part of a ScreenDefinition - in that case, you don't need to take any precautions over ColorRegion clean up - the owning ScreenDefinition manages it. If you create 'orphaned' ColorRegions, see the Close() method.

Parameters:

None

Returns:

None

3.5.1.7 __dir__ (self)

3.5.1.8 def Close (self)

Description

Cleans up the ColorRegion releasing any internal images. Do NOT call this if the ColorRegion is owned by a ScreenDefinition - the ScreenDefinition will manage the lifetime and cleanup of all owned objects. However, if you create an orphaned ColorRegion (either directly or by removing it from the ScreenDefinition) then you should call Close() when finished with the object.

Parameters:

None

Returns:

None

Example:

```
# This is somewhat contrived example which loads a ScreenDefinition from a file
# executes it to get a result and then 'orphans' the ColorRegion (assuming it
# is the first region in the screen definition
>>>sdo = ScreenDefinition()
>>>sdo.Load("file.stscreen")
>>>result = WaitScreenDefMatch(sdo, slotNo=4)[0]
>>>r = result.Regions[0]
>>>result.Regions = []    # now the result sdo has no regions but r is an ColorRegion and 'orphaned'
# do stuff with r
# this frees the potentially large image in r. Without Close() you would wait for Python to garbage collect it.
>>>r.Close()
```

3.5.1.9 def Clone (self)

Description

This clones an ColorRegion returning an exact copy.

Parameters:

None

Returns:

ColorRegion: the cloned regions. The original region is unmodified.

Example:

```
>>>r1 = ColorRegion(rect=(0,0,100,10))
>>>r2 = r1.Clone()
>>>print r2.Rect
>>>(0, 0, 100, 10)
```

3.6 CompareAlgorithm Enum Reference

Description

CompareAlgorithm Enum. Defines the compare algorithm to use in the image comparision functions CompareImageEx(), CompareIconEx(), WaitImageMatch(), WaitImageNoMatch(), WaitIconMatch(), WaitIconNoMatch(). Currently only one algorithm is available

Note:

The comparison algorithm is designed for images and not areas of constant color. If you use the image functions on an area of constant color then the result is likely to be a failed comparison even though they look the same to the human eye - the reason is that using analog capture a region of flat color will have minor variations due to noise and these will be compared as the only variations between the image and they will fail. Use the compare color functions to compare regions of one color.

Parameters:

Compare1 : Algorithm 1 is used for the comparison

Legacy : Not supported. Intention was to allow Legacy comparison with new function but algorithm is not considered suitable.

Example:

```
ret = CompareImageEx("refImage.bmp", "testImage.bmp",algorithm=CompareAlgorithm.Compare1)
```

3.7 CoordConverter Enum Reference

Public Member Functions

- __init__
- __getattr__
- __setattr__
- Clear
- SetMatrix
- SetFromRectangles
- Rotate
- Scale
- Translate
- Invert
- Convert

Private Member Functions

- _Serialize
- _DeSerialize

Static Private Attributes

- __handle = None

Description

This class encapsulates a coordinate converter. It is used to convert points or rectangles between coordinate systems. Although we define 2 coordinate systems: design space and world space, a coordinate converter represents just one transformation and thus 2 instances are used to fully support the design to world and world to design transformations. The script can set the transformation on any slot to be a coordinate converter and this converter is either the design to world or world to design. Design space is where the design of the test occurs (eg, using 704 x 576 as full screen for standard definition or 1920 x 1080 as full screen for HD). World space is the 'real world' and represents the final captured image. For example running on HD, the real world might be 1280 x 720 as the DUT might not be 1920 x 1080. The coordinate converter can then be used to scale design rectangles to world rectangles. Of course, there is nothing to stop a script writer from using this class to convert any set of points/rectangles from any cartesian coordinate system to another.

Parameters:

Matrix Tuple: Read Only. A tuple of 6 numbers (float) representing the 3x3 matrix for the transform. The order is important and represents 3 rows of 2 columns. The last column is fixed as 0, 0, 1 to make a usable 3x3 matrix for later calculations and is not returned here.

Rotation Float: Read Only. The amount of rotation around the origin, in degrees, represented by the transformation. Since this is generic mathematics, the rotation is counter clockwise for a normal right-handed coordinate system. However, the normal coordinate system in computer graphics (and thus StormTest) is left handed and so the rotation is clockwise as would look at images and coordinates in Stormtest. The CoordConverter assumes: rotation, scaling, translation is applied in that order.

Note:

Rotation should be avoided due to performance overhead. It is needed for the HT01 TV Tester due to mechanical constraints but even then users should try to avoid any rotation of the system.

Parameters:

Scaling Tuple: Read Only. The scaling of the converter in the x and y directions. there are just 2 elements to the tuple. The CoordConverter assumes: rotation, scaling, translation is applied in that order.

Translation Tuple: Read Only. This tuple is the translation by the transformation in the x and y directions. The CoordConverter assumes: rotation, scaling, translation is applied in that order.

DesignSize Tuple or List: The intended design size of this converter. 2 values in the tuple, both must be integers. Any non integer will be converted to integers if possible. This value guides the ScreenDefinition in selecting a converter at run time when the design size of a test item does not match the world size. You can set a converter in a screen definition with the same design and world size and if this matches the test item then it will be used.

WorldSize Tuple or List: The intended world size of this converter. 2 values in the tuple, both must be integers. Any non integer will be converted to integers if possible. This value guides the ScreenDefinition in selecting a converter at run time when the design size of a test item does not match the world size. You can set a converter in a screen definition with the same design and world size and if this matches the test item then it will be used.

3.7.1 Member Function Documentation

3.7.1.1 __init__ (self, sourceConverter = None)

Description

The constructor makes a CoordConverter.

Parameters:

← **sourceConverter None**, CoordConverter or List: the source to base this CoordConverter upon. If None then the identity convverter is constructed. The source object can be another CoordConverter instance, a List/Tuple of 2 items, a List/Tuple of 4 items or a List/Tuple of 6 items. If 2 items are given then they are interpreted as the translation component (offset). If 4 items are given then they are interpreted as the linear (scale and rotate) component. If 6 items are given they are interpreted as the full 3 x 2 matrix (3 rows of 2 columns). In all cases, the constructor checks that the final matrix is invertible (all translations are, by definition, invertible).

Returns:

CoordConverter : The completely constructed CoordConverter object. An exception is thrown if the parameters are invalid

Example:

```
>>>cc1 = CoordConverter()
>>>print cc1.Rotation, cc1.Scaling, cc1.Translation
>>>0.0 (1.0, 1.0) (0, 0)
>>>cc2 = CoordConverter([4,4])
>>>print cc2.Rotation, cc2.Scaling, cc2.Translation
>>>0.0 (1.0, 1.0) (4, 4)
>>>cc3 = CoordConverter([2,0,0,2])
>>>print cc3.Rotation, cc3.Scaling, cc3.Translation
>>>0.0 (2.0, 2.0) (0, 0)
>>>cc4 = CoordConverter([3,0,0,3,-1,3])
>>>print cc4.Rotation, cc4.Scaling, cc4.Translation
>>>0.0 (3.0, 3.0) (-1, 3)
>>>cc5 = CoordConverter(cc4)
>>>print cc5.Rotation, cc5.Scaling, cc5.Translation
>>>0.0 (3.0, 3.0) (-1, 3)
```

3.7.1.2 __getattr__ (self, attr)

3.7.1.3 __setattr__ (self, attr, value)

3.7.1.4 def Clear (self)

Description

Clears the CoordConverter back to the Indentity transform (output = input for all inputs)

Parameters:

None

Returns:

None

Example:

```
>>>converter = CoordConverter()
>>>converter.Clear()
>>>print converter.Matrix
>>> (1.0, 0.0, 0.0, 1.0, 0.0, 0.0)
```

3.7.1.5 def SetMatrix (self, matrix)

Description

Set the CoordConverter to use the specified matrix.

Parameters:

← **matrix Tuple:** The matrix is a List/Tuple of 2, 4 or 6 items as described under the constructor.

Returns:

None

Example:

```
>>>cc = CoordConverter()
>>>cc.SetMatrix([1,2,3,4,5,6])
>>>print cc.Matrix
>>>(1.0, 2.0, 3.0, 4.0, 5.0, 6.0)
>>>print cc.Rotation, cc.Scaling, cc.Translation
>>>26.56505 (1.118034, 4.472136) (5, 6)
```

3.7.1.6 def SetFromRectangles (self, sourceRectangle, destRectangle, designSize = None, worldSize = None)

Description

Set the CoordConverter transform from 2 rectangles. The converter will convert from the source to destination.

Parameters:

← **sourceRectangle Tuple:** The rectangle as a Tuple/List of 4 points (left, top, width, height) of the rectangle.

← **destRectangle Tuple:** The rectangle as a Tuple/List of 4 points (left, top, width, height) of the rectangle OR as a list of 4 Tuple/List items. Each Tuple/List being a point. This allows an arbitrary scale/rotate transform.

← **designSize Tuple:** (optional) The DesignSize to be set for the converter. This is optional and if not specified then the size of the sourceRectangle is assumed to be the DesignSize.

← **worldSize Tuple:** (optional) The WorldSize to be set for the converter. This is optional and if not specified then the size of the destRectangle is assumed to be the WorldSize.

Returns:

None

Note:

Strictly speaking only 3 points are needed not 4 to describe the arbitrary rectangle as the rectangle must be a parallelogram. The implementation checks the 4th point to make sure a parallelogram has been specified.
If the destRectangle is not 4 items of left, top, width, height then the CoordConverter WorldSize cannot be set - the intended world size MUST be set manually.

Exceptions:

StormTestExceptions is raised if the CoordConverter cannot find a transform to translate between the specified rectangles.

Example:

```
# Create a coord transform for PAL -> HD where the PAL picture ends up overlapping the HD a bit (and thus would be trunc
>>>cc = CoordConverter()
>>>cc.SetFromRectangles([0,0,704,576], [-10,-10, 1930, 1090])
>>>print cc.Rotation, cc.Scaling, cc.Translation
>>>0.0 (2.741477, 1.892361) (-10, -10)
>>>print cc.Matrix
>>>(2.741477, 0.0, 0.0, 1.892361, -10.0, -10.0)
>>>print cc.WorldSize, cc.DesignSize
>>>(1930, 1090) (704, 576)
```

3.7.1.7 def Rotate (self, degrees)

Description

Update the existing transformation by adding a rotation about the origin. The direction is mathematically counter clockwise but since the normal coordinate system of StormTest is left handed, the rotation is clockwise as it appears in StormTest.

Parameters:

← **degrees Float:** The rotation in degrees. Both positive and negative values are allowed.

Returns:

None

Example:

```
>>>cc = CoordConverter()
>>>print cc.Matrix
>>>(1.0, 0.0, 0.0, 1.0, 0.0, 0.0)
>>>cc.Rotate(1.3)
>>>print cc.Matrix
>>>(0.9997426, 0.02268733, -0.02268733, 0.9997426, 0.0, 0.0)
>>>print cc.Rotation, cc.Scaling, cc.Translation
>>>1.3 (0.9999999, 0.9999999) (0, 0)
```

3.7.1.8 def Scale (self, x, y)

Description

Update the existing transformation by adding a scaling factor. The scale factors must be positive and non negative.

Parameters:

← **x Float:** The scale factor in the horizontal direction to be applied.

← **y Float:** The scale factor in the vertical direction to be applied.

Returns:

None

Example:

```
>>>cc = CoordConverter()
>>>print cc.Matrix
>>>(1.0, 0.0, 0.0, 1.0, 0.0, 0.0)
>>>cc.Scale(1.6, 0.8)
>>>print cc.Matrix
>>>(1.6, 0.0, 0.0, 0.8, 0.0, 0.0)
>>>print cc.Rotation, cc.Scaling, cc.Translation
>>>0.0 (1.6, 0.8) (0, 0)
```

3.7.1.9 def Translate (self, x, y)

Description

Update the existing transformation by adding an ofset. The offset may be negative or positive or zero.

Parameters:

← **x Float:** The offset in the horizontal direction to be applied.

← **y Float:** The offset in the vertical direction to be applied.

Returns:

None

Example:

```
>>>cc = CoordConverter()
>>>print cc.Matrix
>>>(1.0, 0.0, 0.0, 1.0, 0.0, 0.0)
>>>cc.Translate(6.9, -8.4)
>>>print cc.Matrix
>>>(1.0, 0.0, 0.0, 1.0, 6.9, -8.4)
>>>print cc.Rotation, cc.Scaling, cc.Translation
>>>0.0 (1.0, 1.0) (7, -8)
```

3.7.1.10 def Invert (self)

Description

Creates a new CoordConverter that is the inverse transform of this CoordConverter

Parameters:

None

Returns:

transform **CoordConverter:** The inverted transform

Example:

```
>>>cc = CoordConverter()
>>>cc.Rotate(1.3)
>>>print cc.Matrix
>>>(0.9997426, 0.02268733, -0.02268733, 0.9997426, 0.0, 0.0)
>>>inverse = cc.Invert()
>>>print inverse.Matrix
>>>(0.9997426, -0.02268733, 0.02268733, 0.9997426, 0.0, 0.0)
>>>print inverse.Rotation, inverse.Scaling, inverse.Translation
>>>-1.3 (1.0, 1.0) (0, 0)
```

3.7.1.11 def Convert (self, item)

Description

Converts an item using the transform.

Parameters:

← **item List:** The item to be converted. It may be a Tuple/List of 2 items and is interpreted as a single point. It may be a List of Tuples/Lists each of which is 2 items in which case it is interpreted as a list of points. It may be a Tuple/List of 4 items in which case it is interpreted as a rectangle (Left, Top, Width, Height). It may be a List of Tuples/Lists each of which is 4 items in which case it is interpreted as a list of rectangles, each rectangle being Left, Top, Width and Height.

Returns:

transformedItem **List:** The transformed items. It is a copy of the input (the input is not changed) of a similar format - all input Tuples are converted to Lists on output, however.

Example:

```
# Create a coord transform for PAL -> HD where the PAL picture ends up overlapping the HD a bit (and thus would be trunc
>>>cc = CoordConverter()
>>>cc.SetFromRectangles([0,0,704,576], [-10,-10, 1930, 1090])
>>>print cc.Convert([8,100])
>>>[12, 179]
>>>print cc.Convert([[8,100],[100,100],[8,200],[100,200]])
>>>[[12, 179], [264, 179], [12, 368], [264, 368]]
>>>print cc.Convert([8,100,100,100])
>>>[12, 179, 274, 189]
>>>print cc.Convert([[8,100,100,100],[600,500,250,120]])
>>>[[12, 179, 274, 189], [1635, 936, 685, 227]]
```

3.7.1.12 _Serialize (self) [private]

3.7.1.13 _DeSerialize (self, serialisedString) [private]

3.8 CustomRegion Enum Reference

Public Member Functions

- __getattr__
- __setattr__
- __init__
- __del__
- __dir__
- Close
- Clone

Public Attributes

- Descriptor
- Name
- Verify

Private Member Functions

- __checkObject
- __resetObject

Private Attributes

- __cache

Static Private Attributes

- __handle = None
- dictionary __cache = {}
- __closed = False

Description

This represents a Custom Algorithm test. It permits a screen to be verified with a custom algorithm. You can create a custom algorithm but there is no guarantee that when executed the algorithm is installed on the system so the user must take that into account when checking ScreenDefinition results. This is only available in release 3.4

Parameters:

Descriptor AlgorithmDescriptor : The algorithm that will be executed by this CustomRegion.

Name String : The name of the region - this is a human readable name and nothing to do with the algorithm name

Comment String: A general comment. It is not used by StormTest and is provided for the user.

DesignSize Tuple : This is a List of 2 float values representing the width and height of the coordinate system used when designing the object. Given this information it is possible to combine objects designed at different times by different people at different resolutions and reuse the components. It effectively makes the embedded coordinates proportional coordinates.

Parent ScreenDefinition: The parent ScreenDefinition. It will be set on all objects returned from the StormTest API and should be set on objects passed into the StormTest API (but the API is tolerant of users who forget to set the Parent or choose not to set it).

Verify Boolean: A bool to indicate whether the region is for verification purposes (True) or just reading (False).

PassOnNoMatch Boolean: A bool, defaulting to False, which allows the reversal of the condition - that is, the 'pass' status is when the custom algorithm cannot verify the region. This property is ignored if Verify is False.

ReturnScreenImage ReturnScreen: An enumerated type to indicate whether the return should include the full screen at the end of the custom algorithm. It is an enumeration of type ReturnScreen. The default is Inherit which means the image return will be controlled by the enclosing ScreenDefinition

CustomInput object : The algorithm may define input parameters. These can be discovered at run time (via dir()) but should also be documented by the algorithm writer. See the example of the constructor to use the custom input properties.

VerifyStatus Boolean: The output of the verification process. It will be True if this object's verification criteria passed, else False. If Verify is False then it is True as nothing can fail.

Image StormTestImageObject: This is a StormTestImageObject holding the image of the region used for the custom algorithm. The exact meaning of this is determined by the custom algorithm It can be None. The table below summarises when this property is not None

ReturnScreenImage value	Condition for Image to be not None
Always	Always
Never	Never
OnSuccess	VerifyStatus is True
OnFail	VerifyStatus is False
Inherit	Depends on enclosing ScreenDefinition. If no such parent, then as per OnFail

Parameters:

CustomOutput object : The algorithm may define output parameters. These can be discovered at run time (via dir()) but should also be documented by the algorithm writer. See the example of the constructor to use the custom input properties.

3.8.1 Member Function Documentation

3.8.1.1 __getattr__ (self, attr)

3.8.1.2 __setattr__ (self, attr, value)

3.8.1.3 __checkObject (self) `[private]`

3.8.1.4 __resetObject (self, handle) [private]

3.8.1.5 __init__ (self, descriptor, name = '', verify = True)

Description

The constructor makes a CustomRegion object.

Parameters:

← **descriptor AlgorithmDescripor** : The type of algorithm to use in the custom region

← **name String** : Sets the Name property of the constructed region

← **verify Boolean** : Sets the Verify property of the constructed region

Returns:

CustomRegion : The CustomRegion object

Example:

```
>>algorithms = ScreenDefinition.GetExtensionAlgorithms()
>>region = CustomRegion(algorithms[0], "epg screen")      # This is not totally useful as we don't know what algorithm w
>>region.EpgText = "EPG OK"                               # The EpgText is a custom input property (hopefully)
>>region.EpgSize = [500,60]                               # Also EpgSize ia a custom input property
...
>>print region.ActualSize                                 # ActualSize ia a custom output property
>>[]
```

3.8.1.6 __del__ (self)

Description

The CustomRegion has a finalizer as a last resort clean up when the object goes out of scope. Normally the CustomRegion is part of a ScreenDefinition - in that case, you don't need to take any precautions over CustomRegion clean up - the owning ScreenDefinition manages it. If you create 'orphaned' CustomRegions, see the Close() method.

Parameters:

None

Returns:

None

3.8.1.7 __dir__ (self)

3.8.1.8 def Close (self)

Description

Cleans up the CustomRegion releasing any internal images. Do NOT call this if the CustomRegion is owned by a ScreenDefinition - the ScreenDefinition will manage the lifetime and cleanup of all owned objects. However, if you create an orphaned CustomRegion (either directly or by removing it from the ScreenDefinition) then you should call Close() when finished with the object.

Parameters:

None

Returns:

None

Example:

```
# This is somewhat contrived example which loads a ScreenDefinition from a file
# executes it to get a result and then 'orphans' the CustomRegion (assuming it
# is the first region in the screen definition
>>>sdo = ScreenDefinition()
>>>sdo.Load("file.stscreen")
>>>result = WaitScreenDefMatch(sdo, slotNo=4)[0]
>>>r = result.Regions[0]
>>>result.Regions = []    # now the result sdo has no regions but r is an CustomRegion and 'orphaned'
# do stuff with r
# this frees the potentially large image in r. Without Close() you would wait for Python to garbage collect it.
>>>r.Close()
```

3.8.1.9 def Clone (self)

Description

This clones a CustomRegion returning an exact copy.

Parameters:

None

Returns:

CustomRegion: the cloned region. The original region is unmodified.

Example:

```
>>>r1 = CustomRegion(descriptor, "a named item")
>>>r2 = r1.Clone()
>>>print r2.Name
>>>a named item
```

3.9 DebugLevel Enum Reference

Static Public Attributes

- int NONE = 1
- int ERROR = 2
- int WARNING = 3
- int INFO = 4
- int VERBOSE = 5

DebugLevel Enum. Introduced in release 2.8. Defines the valid values for the SetDebugLevel and WriteDebugLine functions. When used in SetDebugLevel, it defines which level of information is put into the log file. All levels are cumulative in that the less severe include the more severe so setting the level to INFO, includes ERROR and WARNING messages as well. When used in WriteDebugLine(), the DebugLevel indicates the severity of the message.

Parameters:

NONE : When used in SetDebugLevel() indicates no tracing be shown. Should not be used in WriteDebugLine() but if used will force the line to appear whenever a log file is created.

ERROR : Error messages - typically issues which have stopped StormTest API completing a task

WARNING : Warning messages - typically issues which should be fixed but allowed StormTest API to complete its task

INFO : Information messages - routine information about actions in StormTest API

VERBOSE : Verbose messages which are intended only to help debug problems in the StormTest API.

Example:

```
>>SetDebugLevel(1, DebugLevel.INFO)     # set debug output to be INFO
>>WriteDebugLine("my debug line", DebugLevel.ERROR) # write the comment 'my debug line' as an error message
```

3.10 HDMIColorSpace Enum Reference

Static Public Attributes

- int Unknown = 1
- int RGB = 0
- int LimitedRGB = 1
- int YUV601 = 2
- int YUV709 = 3
- int XVYCC_601 = 4
- int XVYCC_709 = 5
- int YUV601_FULL = 6
- int YUV709_FULL = 7

Description

VideoColorSpace Enum. Defines the names for color spaces for video systems. Not all StormTest systems support all values of color space.

Parameters:

Unknown : The color space is unknown for the input

RGB : Full Range RGB (0-255 for each of Red, Green and Blue)

LimitedRGB : Limited range RGB (16-235)

YUV601 : YUV to BT.601 (Standard Definition color conversion)

YUV709 : YUV to BT.709 (Normal HD color conversion)

XVYCC_601 : Extended gamut YUV to BT.601

XVYCC_709 : Extended gamut YUV to BT.709

YUV601_FULL : Full range (0-255) YUV to BT.601

YUV709_FULL : Full range (0-255) YUV to BT.709

3.11 ImageRegion Enum Reference

Public Member Functions

- __getattr__
- __setattr__
- __init__
- __del__
- __dir__
- Close
- Clone
- SetImage

Public Attributes

- Name
- Rect
- Verify
- Threshold
- ReferenceImage

Private Member Functions

- __checkObject
- __resetObject

Private Attributes

- __cache

Static Private Attributes

- __handle = None
- dictionary __cache = {}
- __closed = False

Description

This represents an image comparison test. Although it can be used in full image mode, it is recommended to use icon mode because that is more efficient. The reference image or icon has to be stored within the ImageRegion unless a named resource is used. All properties have default values unless explicitly set by the script after construction.

Parameters:

Name String : The name of the region - this is a human readable name.

Comment String : A general comment. It is not used by StormTest and is provided for the user.

DesignSize Tuple: This is a List of 2 float values representing the width and height of the coordinate system used when designing the object. Given this information it is possible to combine objects designed at different times by different people at different resolutions and reuse the components. It effectively makes the embedded coordinates proportional coordinates.

Parent ScreenDefinition: The parent ScreenDefinition. It will be set on all objects returned from the StormTest API and should be set on objects passed into the StormTest API (but the API is tolerant of users who forget to set the Parent or choose not to set it).

Rect Tuple : The rectangle to use - a tuple / List of left, top, width, height as used elsewhere in the API. The width and height are ignored in icon mode but must be set correctly in non icon mode. It is recommended that they are set to match the icon size in icon mode to aid user code documentation and debugging. It can also be a _namedRegion object.

Verify Boolean: A bool to indicate whether the region is for verification purposes (True) or just reading (False).

PassOnNoMatch Boolean: A bool, defaulting to False, which allows the reversal of the condition - that is, the 'pass' status is when the image/icon of the region does NOT match the captured screen. This property is ignored if Verify is False.

ReturnScreenImage ReturnScreen: An enumerated type to indicate whether the return should include the cropped area of the screen used for the image comparison. It is an enumeration of type ReturnScreen. The default is Inherit which means the image return will be controlled by the enclosing ScreenDefinition

IconMode Boolean: A bool, defaulting to True which is true to indicate that the ReferenceImage is an icon. In this case th eposition of the icon is specified by the Rect and the size of the icon is the size of the ReferenceImage.

ReferenceImage StormTestImageObject or NamedImage : This is the reference image used in the image comparison test. If IconMode is True, this image is assumed to be a part of the screen locatied at the top corner of the Rect property. If Iconmode is False then the image is assumed eo be the same size as the captured screen and Rect specifies an area such that the image in the captured screen is matched against the equivalent area in the ReferenceImage.

Threshold Integer: The threshold that the similarity between the captured image and the reference image must equal or exceed in order to be considered a 'matching image'. This is ignored if Verify is False. Default value is 90.

VerifyStatus Boolean: The output of the verification process. It will be True if this object' verification criteria passed, else False. If Verify is False then it is True as nothing can fail.

Image StormTestImageObject: This is a StormTestImageObject holding the image of the region used for image comparison. It can be None. The table below summarises when this property is not None

ReturnScreenImage value	Condition for Image to be not None
Always	Always
Never	Never
OnSuccess	VerifyStatus is True
OnFail	VerifyStatus is False
Inherit	Depends on enclosing ScreenDefinition. If no such parent, then as per OnFail

Parameters:

ActualMatch Float:. After processing this is the actual amount of match between the captured image and the reference image as a percentage between 0 and 100. It will be None if the image comparison has not been performed.

3.11.1 Member Function Documentation

3.11.1.1 __getattr__ (self, attr)

3.11.1.2 __setattr__ (self, attr, value)

3.11.1.3 __checkObject (self) [private]

3.11.1.4 __resetObject (self, handle) `[private]`

3.11.1.5 __init__ (self, name = '', threshold = 90, rect = None, verify = True)

Description

The constructor makes an ImageRegion object. The empty constructor ImageRegion() makes a region to compare the whole screen but has no reference image so you would need to set that separately. All parameters to the constructor are optional

Parameters:

← **name String** : Sets the Name property of the constructed region

← **threshold Integer** : Sets the threshold of the image comparison

← **rect Tuple**, List or NamedRegion : Sets the Rect property of the constructed region

← **verify Boolean** : Sets the Verify property of the constructed region

Returns:

ImageRegion : The constructed ImageRegion object

Example:

```
>>region = ImageRegion("Logo region", 97, (100,20,80,25))
```

3.11.1.6 __del__ (self)

Description

The ImageRegion has a finalizer as a last resort clean up when the object goes out of scope. Normally the ImageRegion is part of a ScreenDefinition - in that case, you don't need to take any precautions over ImageRegion clean up - the owning ScreenDefinition manages it. If you create 'orphaned' ImageRegions, see the Close() method.

Parameters:

None

Returns:

None

3.11.1.7 __dir__ (self)

3.11.1.8 def Close (self)

Description

Cleans up the ImageRegion releasing any internal images. Do NOT call this if the ImageRegion is owned by a ScreenDefinition - the ScreenDefinition will manage the lifetime and cleanup of all owned objects. However, if you create an orphaned ImageRegion (either directly or by removing it from the ScreenDefinition) then you should call Close() when finished with the object.

Parameters:

None

Returns:

None

Example:

```
# This is somewhat contrived example which loads a ScreenDefinition from a file
# executes it to get a result and then 'orphans' the ImageRegion (assuming it
# is the first region in the screen definition
>>>sdo = ScreenDefinition()
>>>sdo.Load("file.stscreen")
>>>result = WaitScreenDefMatch(sdo, slotNo=4)[0]
>>>r = result.Regions[0]
>>>result.Regions = []    # now the result sdo has no regions but r is an ImageRegion and 'orphaned'
# do stuff with r
# this frees the potentially large image in r. Without Close() you would wait for Python to garbage collect it.
>>>r.Close()
```

3.11.1.9 def Clone (self)

Description

This clones an ImageRegion returning an exact copy.

Parameters:

None

Returns:

ImageRegion: the cloned regions. The original region is unmodified.

Example:

```
>>>r1 = ImageRegion(rect=(0,0,100,10))
>>>r2 = r1.Clone()
>>>print r2.Rect
>>>(0, 0, 100, 10)
```

3.11.1.10 def SetImage (self, imgToSet)

Description

Sets the reference image of the region. This is more flexible in terms of input parameters than just setting the ReferenceImage directly and is the preferred method to set the image because it copies the image.

Parameters:

← **imgToSet String**, StormTestImageObject, NamedImage or PIL Image : The image to use as the reference. With the exception of a NamedImage, the imgToSet is converted to a StormTestImageObject and stored as the ReferenceImage. A copy is made if the imgToSet is already a StormTestImageObject. This means the script can use, reuse or dispose of the imgToSet object after calling this function.

Returns:

None

Example:

```
>>>i = ImageObject()
>>>i.SetImage("c:\\myfiles\\sd.png")
```

3.12 LogRegionStyle Enum Reference

Static Public Attributes

- Pass = stormtest_client.LogRegionStyle.Pass
- Warn = stormtest_client.LogRegionStyle.Warn
- Fail = stormtest_client.LogRegionStyle.Fail
- Serial = stormtest_client.LogRegionStyle.Serial
- Observe = stormtest_client.LogRegionStyle.Observe
- Console = stormtest_client.LogRegionStyle.Console
- Verbose = stormtest_client.LogRegionStyle.Verbose
- Entry = stormtest_client.LogRegionStyle.Entry

Description

LogRegionStyle Enum. Defines the names for the log region styles. The supported styles are:

Parameters:

Pass : this is color coded Green.

Warn : this is color coded Orange.

Fail : this is color coded Red.

Serial : this is color coded gray (the same as serial logs in the log files)

Observe : this is color coded purple (the same as Observe lines in the log files)

Console : this is color coded gray (the same as console output in the log files)

Verbose : this is color coded light magenta (the same as verbose lines in the log file)

Entry : this is color coded blue (the same as function entry/exit lines in the log file)

3.13 MeasurementTime Enum Reference

MeasurementTime Enum.

Static Public Attributes

- int OneSixteenthSec = 1024
- int OneEighthSec = 2048
- int OneFourthSec = 4096
- int HalfSec = 8192
- int OneSec = 16384
- int TwoSec = 32768
- int FourSec = 65536
- int EightSec = 131072

MeasurementTime Enum.

Defines the valid time period in which the audio power spectrum is measured and generated. The valid time values are determined by the number of samples used by the audio analysis algorithms to compute a power spectrum result. This enum helps users set the power based algorithms more intuitively by using the time period value than the actual number of samples.

Parameters:

OneSixteenthSec	1/16 second (roughly the time for 1024 samples at 16 kHz).
OneEighthSec	1/8 second (roughly the time for 2048 samples at 16 kHz).
OneFourthSec	1/4 second (roughly the time for 4096 samples at 16 kHz).
HalfSec	1/2 second (roughly the time for 8192 samples at 16 kHz).
OneSec	one seconds (roughly the time for 16384 samples at 16 kHz).
TwoSec	two seconds (roughly the time for 32768 samples at 16 kHz).
FourSec	four seconds (roughly the time for 65536 samples at 16 kHz).
EightSec	eight seconds (roughly the time for 131072 samples at 16 kHz).

3.14 MetaDataType Enum Reference

Static Public Attributes

- string NamedRegion = "NamedRegion"
- string NamedString = "NamedString"
- string NamedColor = "NamedColor"
- string NamedImage = "NamedImage"
- string NamedScreenDef = "NamedScreenDef"
- string Namespace = "Namespace"
- string Schedule = "Schedule"
- string Dut = "Dut"
- string DutModel = "DutModel"
- string Server = "Server"
- string Slot = "Slot"

Description

MetaDataType Enum. Defines the types of objects for which meta data can be assigned. The MetaDataType is passed in to the function calls to identify the object type for which meta data is being read or written.

Parameters:

NamedRegion	: A named region resource
NamedString	: A named string resource
NamedColor	: A named color resource
NamedImage	: A named image resource
NamedScreenDef	: A named screen definition resource
Namespace	: A named folder (namespace)
Schedule	: A schedule
Dut	: A DUT instance
DutModel	: A DUT model
Server	: A server
Slot	: A slot within the server

3.15 MotionRegion Enum Reference

Public Member Functions

- __getattr__
- __setattr__
- __init__
- __del__
- __dir__
- Close
- Clone

Public Attributes

- Name
- Rect
- Verify
- MotionThreshold
- Timeout

Private Member Functions

- __checkObject
- __resetObject

Private Attributes

- __cache

Static Private Attributes

- __handle = None
- dictionary __cache = {}
- __closed = False

Description

This represents a detect motion test. This needs the ability to grab multiple images so can only be used as part of a Screen Definition object that has a slot specified for capturing. It is highly recommended that it is the last used test whether that be for verification or simply reading the motion in a region. Otherwise it will delay other non motion detection regions.

Parameters:

Name String : The name of the region - this is a human readable name.

Comment String : A general comment. It is not used by StormTest and is provided for the user.

DesignSize Tuple : This is a List of 2 float values representing the width and height of the coordinate system used when designing the object. Given this information it is possible to combine objects designed at different times by different people at different resolutions and reuse the components. It effectively makes the embedded coordinates proportional coordinates.

Parent ScreenDefinition: The parent ScreenDefinition. It will be set on all objects returned from the StormTest API and should be set on objects passed into the StormTest API (but the API is tolerant of users who forget to set the Parent or choose not to set it).

Rect Tuple : The rectangle to use - a tuple / List of left, top, width, height as used elsewhere in the API. It can also be a _namedRegion object.

Verify Boolean : A bool to indicate whether the region is for verification purposes (True) or just reading (False).

PassOnNoMatch Boolean : A bool, defaulting to False, which allows the reversal of the condition - that is, the 'pass' status is when there is no motion in the region. This property is ignored if Verify is False.

ReturnScreenImage ReturnScreen: An enumerated type to indicate whether the return should include the cropped area of the screen used for the motion detection. It is an enumeration of type ReturnScreen. The default is Inherit which means the image return will be controlled by the enclosing ScreenDefinition

MotionThreshold Integer : The degree of motion that constitutes motion. It ranges from 0 to 100. Default is 5.

Timeout Float : The maximum time in seconds to wait for motion to occur. Where the PassOnNoMatch is False then this is the time to wait to confirm that there is a static picture. If Verify is False, then this is the time to monitor the motion and the maximum motion in this time is returned. The default value is None and user code must set a timeout.

VerifyStatus Boolean : The output of the verification process. It will be True if this object's verification criteria passed, else False. If Verify is False then it is True as nothing can fail.

Image StormTestImageObject: This is a StormTestImageObject holding the image of the region used for motion detection. It will be the image or portion of the image that was last used for motion detection (not the first in the sequence). It can be None. The table below summarises when this property is not None

ReturnScreenImage value	Condition for Image to be not None
Always	Always
Never	Never
OnSuccess	VerifyStatus is True
OnFail	VerifyStatus is False
Inherit	Depends on enclosing ScreenDefinition. If no such parent, then as per OnFail

Parameters:

ActualMotion Float : The degree of motion detected (range is 0 to 100). It will be None if no motion detection took place. If Verify is True and VerifyStatus is True then this is the motion that caused the verification to pass.

The motion detection works by first grabbing an image and calculating the average and standard deviation of the red, green and blue colors. It then captures additional images, calculates the same metrics and if any differ by more than the threshold then this is the 'motion' returned. The cycle of image grab and calculation continues until either a value greater than or equal to the threshold is detected or the timeout is reached. The most recent tested value is returned. In the case of motion not exceeding the threshold this is the difference between the first image and the image at the timeout - even if intermediate images had more motion.

When Verify is false or PassOnNoMatch is true the exact same algorithm is applied as above: the detection stops when the same conditions are satisfied. Then the logic involved in Verify and PassOnNoMatch is applied to determine the pass or fail status of the region.

A consequence of the algorithm used is that it is optimised for what a human perceives as moving video. There are two known cases where this may cause confusion. If the video is simply noise then the average and standard deviations don't change over time. So StormTest will not report motion. To a human there is no live video - to an engineer there are definitely changes in the video signal. Another case is where the device shows a slow moving screen saver of the type that moves a logo around the screen. The average and standard deviations won't change as the actual image has the same pixels but just in a different location. StormTest won't report motion. A human would not consider this to be live video but again an engineer would ignore the content and consider the image as moving.

ImagesCompared Integer : The number of images compared during execution of the motion region. It will be None if no motion took place. In all other cases it is the actual number of images, that were processed. A value of 1 means that the first image after the initial reference image produced motion.

3.15.1 Member Function Documentation

3.15.1.1 __getattr__ (self, attr)

3.15.1.2 __setattr__ (self, attr, value)

3.15.1.3 __checkObject (self) [private]

3.15.1.4 __resetObject (self, handle) [private]

3.15.1.5 __init__ (self, name = '', rect = None, verify = True, motionThreshold = 5, timeout = None)

Description

The constructor makes a MotionRegion object. The empty constructor NotionRegion() makes a region to detect motion on the whole screen with a threshold of 5.

Parameters:

← **name String** : Sets the Name property of the constructed region

← **rect Tuple**, List or NamedRegion : Sets the Rect property of the constructed region

← **verify Boolean** : Sets the Verify property of the constructed region

← **motionThreshold Integer** : Sets the MotionThreshold property of the constructed region

← **timeout Float** : Sets the Timeout property of the constructed region

Returns:

MotionRegion : The constructed MotionRegion object

Example:

```
>>region = MotionRegion("highlight area", (100,20,80,25))
```

3.15.1.6 __del__ (self)

Description

The MotionRegion has a finalizer as a last resort clean up when the object goes out of scope. Normally the MotionRegion is part of a ScreenDefinition - in that case, you don't need to take any precautions over MotionRegion clean up - the owning ScreenDefinition manages it. If you create 'orphaned' MotionRegions, see the Close() method.

Parameters:

None

Returns:

None

3.15.1.7 __dir__ (self)

3.15.1.8 def Close (self)

Description

Cleans up the MotionRegion releasing any internal images. Do NOT call this if the MotionRegion is owned by a ScreenDefinition - the ScreenDefinition will manage the lifetime and cleanup of all owned objects. However, if you create an orphaned MotionRegion (either directly or by removing it from the ScreenDefinition) then you should call Close() when finished with the object.

Parameters:

None

Returns:

None

Example:

```
# This is somewhat contrived example which loads a ScreenDefinition from a file
# executes it to get a result and then 'orphans' the MotionRegion (assuming it
# is the first region in the screen definition
>>>sdo = ScreenDefinition()
>>>sdo.Load("file.stscreen")
>>>result = WaitScreenDefMatch(sdo, slotNo=4)[0]
>>>r = result.Regions[0]
>>>result.Regions = []    # now the result sdo has no regions but r is an MotionRegion and 'orphaned'
# do stuff with r
# this frees the potentially large image in r. Without Close() you would wait for Python to garbage collect it.
>>>r.Close()
```

3.15.1.9 def Clone (self)

Description

This clones a MotionRegion returning an exact copy.

Parameters:

None

Returns:

MotionRegion: the cloned regions. The original region is unmodified.

Example:

```
>>>r1 = MotionRegion(rect=(0,0,100,10))
>>>r2 = r1.Clone()
>>>print r2.Rect
>>>(0, 0, 100, 10)
```

3.16 OCRRegion Enum Reference

Public Member Functions

- __getattr__
- __setattr__
- __init__
- __del__
- __dir__
- Close
- Clone

Public Attributes

- Name
- Rect
- Verify
- ExpectedText
- MaxDistance
- AutoCorrect
- Languages
- LegacyInvert

Private Member Functions

- __checkObject
- __resetObject

Private Attributes

- __cache

Static Private Attributes

- __handle = None
- dictionary __cache = {}
- __closed = False

Description

This object encapsulates the OCR processing. All properties have default values unless explicitly set by the script after construction.

Parameters:

Name String : The name of the region - this is a human readable name.

Comment String : A general comment. It is not used by StormTest and is provided for the user.

DesignSize Tuple : This is a List of 2 float values representing the width and height of the coordinate system used when designing the object. Given this information it is possible to combine objects designed at different times by different people at different resolutions and reuse the components. It effectively makes the embedded coordinates proportional coordinates.

Parent ScreenDefinition : The parent ScreenDefinition. It will be set on all objects returned from the StormTest API and should be set on objects passed into the StormTest API (but the API is tolerant of users who forget to set the Parent or choose not to set it).

Rect Tuple: The rectangle to use - a tuple / List of left, top, width, height as used elsewhere in the API. It can also be a NamedRegion object.

Verify Boolean : A bool to indicate whether the region is for verification purposes (True) or just reading (False).

PassOnNoMatch Boolean: A bool, defaulting to False, which allows the reversal of the condition - that is, the 'pass' status is when the OCR of the region does NOT match the ExpectedText. This property is ignored if Verify is False.

ExpectedText String or List : This is the expected text. It can be a simple string, or a list of strings. Each string can also be a NamedString object. If a list of strings, then each one is tested when attempting to verify the region. The text can contain ∗ and ? as wildcards. When ExpectedText is supplied and Verify is false, the rules are used to find the best match. Best match is defined as closest Levenshtein distance. In this case, ExpectedTest should be a list of more than 1 item. Since it makes no sense to read a region and supply 1 expected text, (the answer would be that text), if you do supply just one item, it is ignored and the text from the captured image is returned. On the other hand to supply 10 items and have the function return the closest makes a lot of sense.

Note:

There is a change of behaviour in release 3.3.4 - On earlier releases, if 1 item was specified for ExpectedText AND Verify was False then the ExpectedText was always returned. This, although logical, confused users so in release 3.3.4 the behaviour has changed as described above.

Parameters:

MaxDistance Integer: This is the maximum allowed Levenshtein distance as currently used in WaitOCRMatch when verification.

AutoCorrect OCRStringCorrection : This is the auto correction algorithm to apply immediately after OCR capture as described in WaitOCRMatch.

Languages List : A list of languages to use for the OCR region - the format is the same as OCRSetLanguage API and overrides the global setting. It can be None to mean use the global default.

ImageFilters List : A list of image filters to apply - the format is the same as OCRSetImageFiltering and overrides the global setting. It can be None to mean use the global default.

LegacyInvert Boolean : Whether to apply the legacy invert filter. This can be used with the ImageFilters but not recommended. This overrides the OCRSetDefaultImageProcessing global setting. It can be None to mean use the global default.

ReturnScreenImage ReturnScreen : An enumerated type to indicate whether the return should include the cropped area of the screen used for the OCR. It is an enumeration of type ReturnScreen. The default is Inherit which means the image return will be controlled by the enclosing ScreenDefinition

ResultText String : The returned text of the OCR, corrected by the auto correct and, if Verify is false, the closest value from the dictionary, if it is supplied. Will be None if no OCR was performed. The combination of RawText, ResultText, Verify and ExpectedText can be summarized by the table below

Verify	ExpectedText
Actual Screen	VerifyStatus
RawText	ResultText
True	"Options"
"Options"	True
"Options"	"Options"
True	"Options"
"Opts"	False
"Opts"	"Options"
False	"Options"
"Options"	True
"Options"	"Options"
False	"Options"
"Settings"	True
"Settings"	"Options"
False	"Options"
""	True
""	"Options"
False	["Options",""]
""	True
""	""

RawText String: The raw text from the underlying OCR engine before correction or dictionary lookup. Will be None if no OCR was performed.

VerifyStatus Boolean : The output of the verification process. It will be True if this object' verification criteria passed, else False. If Verify is False then it is True as nothing can fail.

Image StormTestImageObject : This is a StormTestImageObject holding the image of the region used for OCR. It can be None. The table below summarises when this property is not None

ReturnScreenImage value	Condition for Image to be not None
Always	Always
Never	Never
OnSuccess	VerifyStatus is True
OnFail	VerifyStatus is False
Inherit	Depends on enclosing ScreenDefinition. If no such parent, then as per OnFail

Parameters:

OCRStatistics Tuple : A tuple of the OCR statistics as normally returned by OCRSlot. None if no OCR was performed.

3.16.1 Member Function Documentation

3.16.1.1 __getattr__ (self, attr)

3.16.1.2 __setattr__ (self, attr, value)

3.16.1.3 __checkObject (self) [private]

3.16.1.4 __resetObject (self, handle) [private]

3.16.1.5 __init__ (self, expectedText = None, name = '', rect = None, verify = True, maxDistance = 0, autoCorrect = OCRStringCorrection.NoCorrection)

Description

The constructor makes an OCRRegion object. The empty constructor OCRRegion() makes a region to OCR the whole screen but has no expected text so you would need to set that separately. All parameters to the constructor are optional

Parameters:

← **expectedText String** or List: Sets the ExpectedText property of the constructed region

← **name String** : Sets the Name property of the constructed region

← **rect Tuple**, List or NamedRegion : Sets the Rect property of the constructed region

← **verify Boolean** : Sets the Verify property of the constructed region

← **maxDistance Integer** : Sets the MaxDistance property of the constructed region

← **autoCorrect OCRStringCorrection** : Sets the AutoCorrect property of the constructed region

Returns:

OCRRegion : The constructed OCRRegion object

Example:

```
>>region = OCRRegion("Options", "Region to Test for Options", (100,20,80,25), autoCorrect = OCRStringCorrection.AllChara
```

3.16.1.6 __del__ (self)

Description

The OCRRegion has a finalizer as a last resort clean up when the object goes out of scope. Normally the OCRRegion is part of a ScreenDefinition - in that case, you don't need to take any precautions over OCRRegion clean up - the owning ScreenDefinition manages it. If you create 'orphaned' OCRRegions, see the Close() method.

Parameters:

None

Returns:

None

3.16.1.7 __dir__ (self)

3.16.1.8 def Close (self)

Description

Cleans up the OCRRegion releasing any internal images. Do NOT call this if the OCRRegion is owned by a ScreenDefinition - the ScreenDefinition will manage the lifetime and cleanup of all owned objects. However, if you create an orphaned OCRRegion (either directly or by removing it from the ScreenDefinition) then you should call Close() when finished with the object.

Parameters:

None

Returns:

None

Example:

```
# This is somewhat contrived example which loads a ScreenDefinition from a file
# executes it to get a result and then 'orphans' the OCR region (assuming it
# is the first region in the screen definition
>>>sdo = ScreenDefinition()
>>>sdo.Load("file.stscreen")
>>>result = WaitScreenDefMatch(sdo, slotNo=4)[0]
>>>r = result.Regions[0]
>>>result.Regions = []    # now the result sdo has no regions but r is an OCRRegion and 'orphaned'
# do stuff with r
# this frees the potentially large image in r. Without Close() you would wait for Python to garbage collect it.
>>>r.Close()
```

3.16.1.9 def Clone (self)

Description

This clones an OCRRegion returning an exact copy.

Parameters:

None

Returns:

OCRRegion: the cloned regions. The original region is unmodified.

Example:

```
>>>r1 = OCRRegion("Option", rect=(0,0,100,10))
>>>r2 = r1.Clone()
>>>print r2.ExpectedText
>>>'Option'
```

3.17 OCRStringCorrection Enum Reference

Static Public Attributes

- string NoCorrection = 'None'
- string AllCharacters = 'AllCharacters'
- string AllNumbers = 'AllNumbers'

Description

OCRStringCorrection Enum. Defines the valid values for OCR string correction. It uses a fixed algorithm for common mistakes (eg 5 and S, 1 and l, 0 and O). It does not guarantee total conversion (for instance if you request AllNumbers and one character is M, it has no idea of a close number so leaves it alone)

Parameters:

NoCorrection : No correction - the string is unchanged.

AllCharacters : Convert the string to characters where possible

AllNumbers : Convert the string to numbers where possible

3.18 OffHookMode Enum Reference

Static Public Attributes

- string Silence = 'Silence'
- string DialTone = 'DialTone'
- string AudioNoise = 'AudioNoise'

Description

OffHookMode Enum. Defines the valid values for off-hook mode.

Parameters:

Silence : Present no tone when off-hook

DialTone : Present dial tone when off-hook

AudioNoise : Present audio noise when off-hook

3.19 OutputTo Enum Reference

Static Public Attributes

- Console = DebugLog.LOG_TO_CONSOLE
- SerialLog = DebugLog.LOG_TO_SERIAL
- File = DebugLog.LOG_TO_FILE

Deprecated

3.20 OverWriteAction Enum Reference

OverWriteAction Enum.

OverWriteAction Enum.

Defines the valid values that can be passed as an argument to CaptureImageEx().

Parameters:

OverWrite	Overwrites an existing file when saving an image with same name
Error	Returns an error when a file exists with the same name of the image being captured
NewName	Saves the file with a new name if the name already exists

Example:

```
ret = CaptureImageEx (None, "file.bmp",overwriteAction=OverWriteAction.OverWrite)
print ret
>>> [[8, True, <StormTestImageObject instance>, u'file.jpg']]
ret = CaptureImageEx (None,  "file.bmp",overwriteAction=OverWriteAction.Error)
print ret
>>> [[8, True,<StormTestImageObject instance >,'Error']]
ret = CaptureImageEx (None,  "file.bmp",overwriteAction=OverWriteAction.NewName)
print ret
>>> [[8, True,<StormTestImageObject instance>, u'file(1).bmp']]
```

3.21 PerceptualParameters Enum Reference

Public Member Functions

- __init__
- ToString

Public Attributes

- videoFormat
- allowJerkiness

Deprecated

3.21.1 Member Function Documentation

3.21.1.1 __init__ (self, VideoFormat = "H264", AllowJerkiness = True)

3.21.1.2 def ToString (self)

3.22 ProcessImage Enum Reference

Description

ProcessImage Enum. Defines the valid values for OCR image pre-processing.

Parameters:

Invert : Invert the image.

Greyscale : Convert the image to greyscale. DEPRECATED, do not use. Use the OCR Filters. Retained for legacy applications.

Example:

```
>>print OCRSetDefaultImageProcessing(ProcessImage.Invert)
>>True
```

3.23 RectangleMatrix Enum Reference

Public Member Functions

- __init__

Public Attributes

- Bounds
- Rows
- Columns
- HorizontalGap
- VerticalGap

Description

This represents a rectangle matrix object. This is n x m sub rectangles, each of which is treated as a separate rectangle. This is available in release 3.4 and is only supported in a CustomRegion

Parameters:

Bounds List : A list of 4 integers representing the bounding rectangle of the matrix. This allows a border to be defined outside of the rectangle matrix. The integers are the same as for any Rectangle in the API.

Rows Integer : The number of rows in the matrix. It must be greater than 0.

Columns Integer : The number of columns in the matrix. It must be greater than 0.

HorizontalGap Float : The gap between columns.

VerticalGap Float : The gap between rows. The horizontal and vertical gaps are defined as floating point to allow calculation of rectangles to floating point accuracy - most algorithms will then need to round to the nearest pixel but the flexibility has been designed in now to allow fractional pixel processing.

3.23.1 Member Function Documentation

3.23.1.1 __init__ (self, bounds = [0, rows = 1, columns = 1, horizontalGap = 0, verticalGap = 0)

3.24 ReturnScreen Enum Reference

Static Public Attributes

- int Never = 0
- int Always = 1
- int OnSuccess = 2
- int OnFail = 4
- int Inherit = 128

Description

This enumeration controls the return of images in the ScreenDefintion and all sub objects.

Parameters:

Never : Never return an image

Always : Always return an image

OnSuccess : Return an image of success only

OnFail : Return an image on Fail only

Inherit : Use the parent's setting. This cannot be applied to a top level ScreenDefinition

3.25 SaveScreenCap Enum Reference

Deprecated

3.26 ScreenDefinition Enum Reference

Public Member Functions

- __getattr__
- __setattr__
- __del__
- __dir__
- __init__
- Close
- Save
- Serialize
- DeSerialize
- Load
- Clone
- ToNamedResource
- WaitMatch
- WaitNoMatch
- Match
- TestRawZapFrames
- FindRawZapMatch
- FindRawZapNoMatch
- GetExtensionAlgorithms

Private Member Functions

- __checkObject
- __resetObject
- __deserialize

Private Attributes

- __cache

Static Private Attributes

- __handle = None
- dictionary __cache = {}
- __closed = False

Description

The object defines the screen. It is a collection of sub objects which define areas of a screen and tests to perform. The tests can involve verification of simply read and return values. A test can in fact be another screen definitino object to construct a hierarchical screen definition. During execution, all the tests with verification state (Verify property is True) are tested and if all the validation criteria are met then the read criteria are read. With the exception of a fatal system type error, a read operation cannot fail. Where Unicode is specified, a simple Python string (assumed UTF-8) can be used. Where float is specified, an integer can be used. Where a list is specified, a tuple may be used instead.

Parameters:

Name String : The name of the screen definition - this is a human readable name.

Comment String: A general comment. It is not used by StormTest and is provided for the user.

DesignSize Tuple: This is a List of 2 float values representing the width and height of the coordinate system used when designing the screen definition. Given this information it is possible to combine objects designed at different times by different people at different resolutions and reuse the components. It effectively makes the embedded coordinates proportional coordinates.

Parent ScreenDefinition: The parent ScreenDefinition. It is None on the top level screen definition. It will be set on all screen defintion sub objects returned from the StormTest API and should be set on objects passed into the StormTest API (but the API is tolerant of users who forget to set the Parent or choose not to set it).

ReturnScreenImage ReturnScreen: An enumerated type to indicate whether the return should include the full screen used for the evalutation. It is an enumeration of type ReturnScreen. The default is OnFail which means the image will be returned only on a failure.

ModelList List : Read Only. When the screen definition is loaded from a NamedScreenDefinition then the list of models that the NamedScreenDefinition supported is listed here. It will be empty for generic NamedScreenDefinition. This property is for informational purposes.

Regions List : List of regions owned by the screen definition. The default is empty. You can add/remove items or replace the list entirely. You should not share a list between screen definitions because this confuses the clean up code. When the screen definition is executed the types and values of the Regions property are checked - until that moment, you may be able to put erroneous items in the Regions list.

CoordConverters List : List of CoordConverter objects to perform coordinate conversions. Each object specifies how to convert coordinates from one size to another. If any sub objects have a design size different from the captured video size then coordinate transformation must occur. Default is an empty list. See discussion on how this works _coordtransforms

VerifyStatus Boolean: The output of the verification process. It will be True if all of this screen defintions object's verification criteria passed, else False.

Image StormTestImageObject: This is a StormTestImageObject holding the image of the region used for evalutation. If a MotionRegion or an AudioRegion is contained in the screen definition then it will be the last image used for motion detection or audio detection - not the image used for the image, color or OCR tests. It can be None. The table below summarises when this property is not None

ReturnScreenImage value	Condition for Image to be not None
Always	Always
Never	Never
OnSuccess	VerifyStatus is True
OnFail	VerifyStatus is False
Inherit	Depends on enclosing ScreenDefinition. If no such parent, then as per OnFail

Parameters:

Errors List : A list of strings holding the errors encountered udring evaluation of the screen definition. These are errors such as a failure to capture an image, invalid parameters detected on the server etc. They **do not** include a normal failure to verify a region. The Errors will be None if no errors are found (the usual condition), else a list of strings (usually only 1 string as the first serious error stops evaluation). When screen definitions are nested, the outermost screen definition includes all the inner errors as a master list of errors.

3.26.1 Member Function Documentation

3.26.1.1 __getattr__ (self, attr)

3.26.1.2 __setattr__ (self, attr, value)

3.26.1.3 __del__ (self)

Description

The Screen Definition has a finalizer to clean up when the object goes out of scope. However, you should call Close() when finished with the screen definition. The finalizer simply calls Close().

Parameters:

None

Returns:

None

3.26.1.4 __dir__ (self)

3.26.1.5 __checkObject (self) [private]

3.26.1.6 __resetObject (self, handle) [private]

3.26.1.7 __init__ (self)

Description

The constructor makes an emtpy screen definition object, ready for you to add regions.

Parameters:

None

Returns:

ScreenDefinition : The constructed empty screen definition

Example:

```
>>>sdo = ScreenDefinition()
>>>print len(sdo.Regions)
>>>0
```

3.26.1.8 __deserialize (self, serializedData) [private]

3.26.1.9 def Close (self)

Description

This cleans up the screen definition, calling Close() on all the regions. It also disposes of the Image in the screen definition. After calling Close() any other access to the screen definition, a sub object or an image (even if a reference is held elsewhere) is likely to cause an exception. This is the key reason why you must not share Regions between screen definitions - the Close() will close the shared list. If you wish to share data, use the Clone() method on the regions.

Parameters:

None

Returns:

None

Example:

```
>>>sdo = ScreenDefinition()
>>>sdo.Close()
```

3.26.1.10 def Save (self, filename)

Description

Saves the screen definition to a file on the local file system. If there is an error during save (privilege, disk full etc) then an exception is thrown.

Parameters:

← **filename String** : The filename to save to, including extension. You can use any extension but if you wish to edit it within StormTest Developer Suite, you **must** use .stscreen as the extension. The filename can be absolute or relative. If it is relative then it is saved relative to the 'images' directory as set by the SetDirectories() and SetProjectDir() functions. The format of the file is JSON text with images stored as base64 encoded such that if the base64 is decoded and saved as binary to a file then Windows can read the file correctly.

Returns:

None

Example:

```
>>>sdo = ScreenDefinition()
>>>sdo.Save("mysdo.stscreen")
>>>sdo.Save("c:\\users\\me\\mysdo.stscreen")
```

3.26.1.11 def Serialize (self)

Description

Serializes the screen definition to a string. The format is the same as Save(). You could use Serialize and then choose how/when to save the screen definition using normal python code.

Parameters:

None

Returns:

string : The serialized data

Example:

```
>>>sdo = ScreenDefintion()
>>>s = sdo.Serialize()
```

3.26.1.12 def DeSerialize (self, jsonData)

Description

De-Serializes a string into the screen definition. It reverses Serialize()

Parameters:

← **jsonData String** : the serialized data (from Serialize or other source)

Returns:

None

Example:

```
>>>sdo = ScreenDefintion()
>>>s = sdo.Serialize()
>>>sdo2 = ScreenDefintion()
>>>sdo2.DeSerialize(s)
```

3.26.1.13 def Load (self, filename)

Description

Loads the screen definiton from the specified filename, destroying all the contents of the pre existing screen definition object. If there is an error loading the file, then and exception si thrown and the object is in an indeterminate state (Close() will work to clean it up). It should be noted that even on a non-existent file, the user **cannot** assume the object was unchanged and can still be used.

Parameters:

← **filename String** : The file name to load the screen definition from. It can be absolute or relative and if relative then it is relative to the 'images' directory as set by the SetDirectories() and SetProjectDir() functions.

Returns:

None

Example:

```
>>>sdo = ScreenDefintion()
>>>sdo.Load("mysdo.stscreen")
```

3.26.1.14 def Clone (self)

Description

This clones a ScreenDefiniton object returning an exact copy. All sub regions are also cloned. This is a 'deep copy'.

Parameters:

None

Returns:

ScreenDefinition: the cloned screen definition. The original screen definition is unmodified.

Example:

```
>>>sdo = ScreenDefinition()
>>>sdo.Regions.append(ImageRegion())
>>>print len(sdo.Regions)
>>>1
>>>sdo2 = sdo.Clone()
>>>sdo.Regions.append(OCRRegion())
>>>print len(sdo.Regions), len(sdo2.Regions)
>>>2 1
```

3.26.1.15 def ToNamedResource (self, path, comment = None, dutModelId = 0)

Description

This is the same as WriteNamedScreenDefinition but provides an instance method (in fact the WriteNamedScreenDefinition is implemented as a call to ToNamedResource). path is always case sensitive and SetNamedResourceMatchCase is ignored. There is no requirement for the path to end in the same name as the embedded name of the screenDefinition. Likewise the comment is not related to the embedded comment property.

Parameters:

← **path String:** the full absolute path to the screen definition (may begin with a / but not necessary), path elements separated by /. Final element is the screen definition name in the database.

← **comment String:** (optional) Free form text of a user defined comment.

← **dutModelId Integer**, String or List : The model for which the data is to be written. The integer value of 0 means the generic model only will be created/updated. It can also be an explicit model number, a string representing a single model or a list of strings representing multiple models.

Returns:

None : An exception is thrown on an error (for example bad parameters, no communication).

Example:

```
>>>sdo = ScreenDefinition()
>>>sdo.ToNamedResource('myprojects/empty sdo')
```

3.26.1.16 def WaitMatch (self, timeToWait = 5, waitGap = 1, slotNo = 0)

Description

Executes the verification stages of the screen definiton until they pass, then executes the read stages. On a failure, a wait of waitGap occurs between retries and if the time taken exceeds timeToWait then the call returns.

Parameters:

← **timeToWait Float** : Optional. The maximum time to wait in seconds for the VerifyStatus of the screen definition to become true. Default is 5 seconds.

← **waitGap Float** : Optional. The time between successive attempts at verification in seconds. This is the time to wait after a failure before trying again.

← **slotNo Integer** : Optional. The slot to use for the evaluation of the screen definition object. If slotNo is non zero then all reserved slots are used.

Returns:

List : If slotNo is non zero then the returned list is [screen definition, time waited, number of captures made]. If slotNo is zero then the return is a list of lists, each sub list being [slotNo, [screen definition, time waited, number of captures made]]

Note:

that in all cases the execution time may exceed timeToWait. For example the 4th test at time 4.5 seconds fails so the code waits 1 second, tries again at 5.5 seconds and passes.
If the screenDef includes a motion detection region with a long timeout then the overall timeToWait may need to be adjusted upwards if failures are anticipated and retries are desired. The returned screenDefinition is a **copy** of the input with output fields filled in. The original input screen not altered in any way and can thus be reused in many API calls.

Example:

```
>>>sdo.WaitMatch(4, 0.1)
>>>[[5,[<ClientAPI.ScreenDefinition object>, 1.2388, 4]]]
>>>sdo.WaitMatch(slotNo=5)
>>>[<ClientAPI.ScreenDefinition object>, 1.2388, 6]
```

3.26.1.17 def WaitNoMatch (self, timeToWait = 5, waitGap = 1, slotNo = 0)

Description

Executes the verification stages of the screen definiton until one of them fails. This means the screen on the DUT no longer matches the screen definition object. On a pass, a wait of waitGap occurs between retries and if the time taken exceeds timeToWait then the call returns. Whenever the verification stages pass, the read stages are executed. This can result in unnecessary reads so the user should ise this function with screen definitions whcih have either no read regsions or fast to execute read regions (such as ColorRegion and ImageRegion). OCRRegions with Verify as False should be avoided because they take time and use up OCR license characters. The purpose of the function is to wait until the DUT screen changes away from a known state where you don't care what happens next so long as something happens.

Parameters:

← **timeToWait Float** : Optional. The maximum time to wait in seconds for the VerifyStatus of the screen definition to become false. Default is 5 seconds.

← **waitGap Float** : Optional. The time between successive attempts at non verification in seconds. This is the time to wait after a success before trying again.

← **slotNo Integer** : Optional. The slot to use for the evaluation of the screen definition object. If slotNo is non zero then all reserved slots are used.

Returns:

List : If slotNo is non zero then the returned list is [screen definition, time waited, number of captures made]. If slotNo is zero then the return is a list of lists, each sub list being [slotNo, [screen definition, time waited, number of captures made]]

Note:

that in all cases the execution time may exceed timeToWait. For example the 4th test at time 4.5 seconds passes so the code waits 1 second, tries again at 5.5 seconds and fails.
If the screenDef includes a motion detection region with a long timeout then the overall timeToWait may need to be adjusted upwards if successes are anticipated and retries are desired. The returned screenDefinition is a **copy** of the input with output fields filled in. The original input screen not altered in any way and can thus be reused in many API calls.

Example:

```
>>>sdo.WaitNoMatch(4, 0.1)
>>>[[5,[<ClientAPI.ScreenDefinition object>, 1.2388, 4]]]
>>>sdo.WaitNoMatch(slotNo=5)
>>>[<ClientAPI.ScreenDefinition object>, 1.2388, 6]
```

3.26.1.18 def Match (self, whatToMatch = 0, slotNo = 0)

Description

The runs a single evaluation of the screen definition on a slot or an image (depending on the parameter supplied).

Parameters:

← **whatToMatch PIL** Image, StormTestImageObject, NamedImage or String: The image to use for matching. If this is 0 (the integer value 0) then this is functionally equivalent to calling WaitMatch(0, 0, slotNo) but with a simplified return value. If whatToMatch is a valid image then an evaluation occurs using that image. In this case, MotionRegions and AudioRegions in the screen definition will result in a StormTestExceptions being raised as it is impossible to detect motion or audio in a single image. If whatToMatch is a string then it is assumed to be a file and an image will be loaded using the same semantics as the StormTestImageObject.FromFile() method. This option is available on Release 3.3.3 and later.

← **slotNo Integer** : The slot number to use for evaluation when whatToMatch is 0. If whatToMatch is an image (or string) then the slotNo is used to determine how resources are resolved. If slotNo is 0 then generic resources are used, otherwise resources for the specific model in the specified slot are used. There is no requirement here that the specified slot is reserved when using this method on an image (however some slot must be reserved for the server to accept the request).

Returns:

ScreenDefinition or List : If whatToMatch is 0 and slotNo is non zero then a single ScreenDefinition is returned. If whatToMatch is 0 and slotNo is 0 then a List is returned. Each item in the list is a list of [slotNo, ScreenDefinition] where slotNo is the reserved slot. If whatToMatch is an image then the retuen is a ScreenDefinition. In all cases the returned ScreenDefinition is a copy and the original screen definition is unchanged.

Example:

```
>>>sdo = ScreenDefinition()
>>>print sdo.Match(0, 0)
>>>[[4, <ClientAPI.ScreenDefinition object>], [5, <ClientAPI.ScreenDefinition object>]]
>>>print sdo.Match(0, 4)
>>><ClientAPI.ScreenDefinition object>
>>>im = Image.open("c:\\users\\me\\myimg.png")
>>>print sdo.Match(im)          # use generic lookup of resources
>>><ClientAPI.ScreenDefinition object>
>>>print sdo.Match(im, 7)       # use resources for dut model in slot 7
>>><ClientAPI.ScreenDefinition object>
>>>print sdo.Match("c:\\users\\me\\myimg.png")   # read a file and use generic lookup of resources
>>><ClientAPI.ScreenDefinition object>
```

3.26.1.19 def TestRawZapFrames (self, frameIndex, slotNo = 0)

Description

Evaluates the screen definition over one or more arbitrary frames captured from the high speed timer. The screen definition cannot contain a MotionRegion or an AudioRegion and a StormTestExceptions exception is raised if it does.

Parameters:

← **frameIndex Integer** or List : The frame of frames to test. It can be a single integer for a single frame test or a list of integers. If a list then multiple tests are performed. Frame indexes are zero based.

← **slotNo Integer** : Optional. The slot number. There must have been a prior execution of the high speed timer on this slot with frames captured.

Returns:

ScreenDefinition or List : see example code for the combinations of return types and input parameters.

Note:

There is no requirement for any particular order of frames with specifying a list: [6, 10, 56] is as valid as [99, 3, 78].

Example:

```
>>>ReserveVideoTimer()
>>>EnableCaptureZapFrames()
>>>TriggerHighSpeedTimer()
>>>WaitSec(5)
>>>StopHighSpeedTimer()
# now we can work on the zap frames
>>>sdo = ScreenDefinition()
>>>sdo.Load("c:\\users\\me\\mysdo.stscreen")
>>>print sdo.TestRawZapFrames(5, slotNo=1)
>>><ClientAPI.ScreenDefinition object>
>>>print sdo.TestRawZapFrames([6,7,8], slotNo=1)
>>>[(6, <ClientAPI.ScreenDefinition object>), (7, <ClientAPI.ScreenDefinition object>),(8, <ClientAPI.ScreenDefinition
>>>print sdo.TestRawZapFrames(5)
>>>[[1,<ClientAPI.ScreenDefinition object>]]
>>>print sdo.TestRawZapFrames([6,7,8])
>>>[[1, [(6, <ClientAPI.ScreenDefinition object>), (7, <ClientAPI.ScreenDefinition object>),(8, <ClientAPI.ScreenDefini
```

3.26.1.20 def FindRawZapMatch (self, startIndex, maxFrames, slotNo = 0)

Description

Evaluates the screen definition over a range of contiguous previously captured frames from the high speed timer looking for a match of the screen definition (VerifyStatus returned as True). A MotionRegion may be used in the screen definition - it will only examine frames up to the last specified in the range. It will calculate the timeout based on the frame rate of the original capture not on real time to do the testing.

Parameters:

← **startIndex Integer** : Zero based start frame to scan for screen definition match.

← **maxFrames Integer** : The maximum number of frames to scan for a match.

← **slotNo Integer** : Optional. The slot number. There must have been a prior execution of the high speed timer on this slot with frames captured. If omitted or 0 then the scan occurs on all reserved slots.

Returns:

Tuple : If slotNo is non zero then a tuple of ScreenDefinition that matched and index where a match was found. If slotNo is zero then a List of slot, tuples. If no match is found then the matched frame is None instead of an Integer.

Note:

The operation of the motion region can be confusing regarding the return value from FindRawZapMatch. If you specify a timeout on the motion region of, for example, 10 seconds then FindRawZapMatch starts at startIndex and looks for motion within 10 seconds of that point. If motion is found in the 10 seconds then startIndex will be returned. If no motion is found then FindRawZapMatch starts at startIndex + 1 and searches for 10 seconds. If motion is found then startIndex + 1 is returned. If no motion is found then the process is repeated until startIndex + maxFrames is reached. If you wish to measure **when** motion starts after a static picture then you should set the timeout on the motion region to be large, verify that startIndex is returned from FindRawZapMatch and examine the ImagesCompared property of the returned MotionRegion - this will tell you how many images were needed to find motion and thus how long before motion started. The ImagesCompared property is new in release 3.2.6

Example:

```
>>>ReserveVideoTimer()
>>>EnableCaptureZapFrames()
>>>TriggerHighSpeedTimer()
>>>WaitSec(5)
>>>StopHighSpeedTimer()
# now we can work on the zap frames
>>>sdo = ScreenDefinition()
>>>sdo.Load("c:\\users\\me\\mysdo.stscreen")
>>>print sdo.FindRawZapMatch(0, 25, slotNo=6)
>>>(<ClientAPI.ScreenDefinition object>, 12)
>>>print sdo.FindRawZapMatch(25, 100)
>>>[[6, (<ClientAPI.ScreenDefinition object>, None)]]
```

3.26.1.21 def FindRawZapNoMatch (self, startIndex, maxFrames, slotNo = 0)

Description

Evaluates the screen definition over a range of contiguous previously captured frames from the high speed timer looking for a NON match of the screen definition (VerifyStatus returned as False). A MotionRegion may be used in the screen definition - it will only examine frames up to the last specified in the range. It will calculate the timeout based on the frame rate of the original capture not on real time to do the testing.

Parameters:

← **startIndex Integer** : Zero based start frame to scan for screen definition non match.

← **maxFrames Integer** : The maximum number of frames to scan for a non match.

← **slotNo Integer** : Optional. The slot number. There must have been a prior execution of the high speed timer on this slot with frames captured. If omitted or 0 then the scan occurs on all reserved slots.

Returns:

Tuple : If slotNo is non zero then a tuple of ScreenDefinition that failed to match and index where the match was not found. If slotNo is zero then a List of slot, tuples. If all frames match then the matched frame is None instead of an Integer.

Example:

```
>>>ReserveVideoTimer()
>>>EnableCaptureZapFrames()
>>>TriggerHighSpeedTimer()
>>>WaitSec(5)
>>>StopHighSpeedTimer()
# now we can work on the zap frames
>>>sdo = ScreenDefinition()
>>>sdo.Load("c:\\users\\me\\mysdo.stscreen")
>>>print sdo.FindRawZapNoMatch(0, 25, slotNo=6)
>>>(<ClientAPI.ScreenDefinition object>, 12)
>>>print sdo.FindRawZapNoMatch(25, 100)
>>>[[6, (<ClientAPI.ScreenDefinition object>, None)]]
```

3.26.1.22 def GetExtensionAlgorithms ()

Description

Gets the list of Sdo Extension Algorithms installed on the config server being used by the Script. This is a live read of the list of algorithms - if an administrator is adding or removing algorithms while a script is running, this will be reflected in the result of this method. Getting the list can also be time consuming, requiring a round trip on the network so script writers are advised to cache the result in a variable unless live updates are needed. This method is available in Release 3.4

Parameters:

None

Returns:

List : Each item in the list is an AlgorithmDescriptor describing one SdoExtension algorithm. If no extensions are installed then the result is an empty list.

Example:

```
>>>print ScreenDefinition.GetExtensionAlgorithms()
>>>[<ClientAPI.AlgorithmDescriptor object>, <ClientAPI.AlgorithmDescriptor object>]
>>>algorithms = ScreenDefinition.GetExtensionAlgorithms()
>>>print algorithms[0].Id, algorithms[0].Name, algorithms[0].Version
>>>12453B27-4736-4EAA-BC92-655DE7716871 My Algorithm V1.0
```

3.27 SignalRecoveryMode Enum Reference

Static Public Attributes

- int Safe = 0
- int Fast = 1

Description

SignalRecoveryMode Enum. Defines the names for signal recovery mode. See SetSignalRecoveryMode()

Parameters:

Safe : The hardware performs in a safe manner, assuming the worst case scenario

Fast : The hardware performs in a fast manner. This assumes that the video resolution on signal recovery was the same as when it was lost.

3.28 SystemCapability Enum Reference

Static Public Attributes

- IR = stormtest_client.SystemCapability.IR
- Power = stormtest_client.SystemCapability.Power
- Serial = stormtest_client.SystemCapability.Serial
- BasicAV = stormtest_client.SystemCapability.BasicAV
- HD = stormtest_client.SystemCapability.HD
- OCR = stormtest_client.SystemCapability.OCR
- VTA = stormtest_client.SystemCapability.VTA
- AVAnalysis = stormtest_client.SystemCapability.AVAnalysis
- NamedResources = stormtest_client.SystemCapability.NamedResources
- DUTSpecificResources = stormtest_client.SystemCapability.DUTSpecificResources
- SDO = stormtest_client.SystemCapability.SDO
- GPUVideoCompression = stormtest_client.SystemCapability.GPUVideoCompression

Description

SystemCapability Enum. Defines the names for all the system and server capabilities. The supported list is:

Parameters:

IR : The server has IR device installed

Power : The server has RPS power device installed

Serial : The server has serial controller device installed

BasicAV : The server has audio video capture card installed and can do basic AV functions

HD : The server has HD audio video capture card installed and can do basic AV functions at High Definition

OCR : The server has OCR and a valid license for OCR

VTA : The server can do High Speed Video Timing (this implies BasicAV)

AVAnalysis : The server can do audio/video analysis (is licensed for it)

NamedResources : The System supports named resources.

DUTSpecificResources : The System supports named resources per DUT model

SDO : The server supports screen definition objects

GPUVideoCompression : The server is doing Video Compression using the GPU (host based s/w encoding) rather than hardware encoding. Some servers cannot do hardware encoding. If GPU encoding is being used, there is a higher CPU cycle usage so video streaming (eg log file recording or viewing) should only be used when absolutely necessary. When GPU encoding is used in HD it is possible to control the streaming resolution with SetResolution. This capability was added in release 3.3.0

3.29 TestResult Enum Reference

Deprecated

3.30 TLSStatus Enum Reference

Static Public Attributes

- string OnHook = 'OnHook'
- string OffHook = 'OffHook'

Description

TLSStatus Enum. Defines the status of a telephone line simulator port.

Parameters:

OnHook	On-hook
OffHook	Off-hook

3.31 TM Enum Reference

Static Public Attributes

- int PASS = 111
- int PARTIAL_PASS = 110
- int FAIL = 100
- int NOT_RUN = 101

Description

TM Enum. Defines the valid values expected by the StormTest Test Manager after a test runs.

Parameters:

PARTIAL_PASS : Test passed on some of the slots it ran on.

FAIL : Test failed.

PASS : Test passed.

NOT_RUN,: The test did not run. This is never returned by a test script but can be returned by a test step to indicate that the test step did not run.

Example:

```
>>ReturnTestResult(TM.PASS)
```

3.32 TouchType_iOS Enum Reference

Static Public Attributes

- string BEGAN = "TOUCHESBEGAN"
- string MOVED = "TOUCHESMOVED"
- string ENDED = "TOUCHESENDED"
- string STATIONARY = "TOUCHESSTATIONARY"
- string CANCELLED = "TOUCHESCANCELLED"

Deprecated

3.33 TriggerCondition Enum Reference

Public Member Functions

- __init__
- __getattr__
- __setattr__
- __dir__
- FromImage
- FromIcon
- FromColor

Public Attributes

- Delay
- FrameCount
- DroppedFrames
- IRKey
- IRKeyRepeat
- Sdo
- SdoRepeat
- FromNow

Private Attributes

- __cache

Static Private Attributes

- dictionary __cache = {}

Description

The trigger condition. This is used by several functions within VQA API to indicate when a function should take action (be triggered). Not all properties are appropriate for all functions and the individual API calls will indicate which properties are ignored. It should be assumed that a property is valid unless specified otherwise. It is possible to specify multiple conditions on one trigger object and in this case the condition is satisfied when any one of the conditions is met. If any property is None then it is ignored in considering the trigger. This is the default value for many properties.

Parameters:

Delay float : A time in seconds. Default value is None.

FrameCount integer : Number of frames as an integer. Default value is None.

DroppedFrames integer : Number of dropped frames. Useful to trigger on an error by setting to 1. Default value is None. This is not supported in release 3.2.3.

IRKey String : An IRKey stroke to use to trigger the analysis. Default value is None. This is not supported at version 3.2.3 and must be set to None

IRKeyRepeat integer : The number of times that the IRKey must be present before it is considered the trigger. Default value is 1 meaning the first occurrence of the IRKey will cause the trigger to be considered triggered.

Sdo ScreenDefinition : A screen definition to match each frame. The Sdo may only include image, and color tests. Motion, audio and OCR are forbidden as they take too long. You should choose small regions to test so that the SDO can execute in less than 1 frame time. A full image compare cannot operate in real time. In this release, you cannot use NamedRegions, NamedColors or NamedImages within an SDO. Default value is None

SdoRepeat integer : Number of frames that must match the SDO for it to be considered a valid trigger. Default value is 1. Only a value of 1 is supported in release 3.2.3.

FromNow bool : True to consider the delay, frame count etc to start 'now'. If False, then the reference point is the most recent logical point in the past. This is not always sensible. A useful case is setting it to False on the Stop() call with a fixed delay. This means the Stop() will be exactly the specified value after Start() even if the script is a bit slow in calling Stop(). Useful for precise timing. Default value is True.

3.33.1 Member Function Documentation

3.33.1.1 __init__ (self)

Description

Constructor to make a default trigger object

Parameters:

None

Returns:

TriggerCondition : the default trigger object. This will simply trigger immediately.

Example:

```
>>>create a trigger to trigger after 1000 frames
>>>trigger = TriggerCondition()
>>>trigger.FrameCount = 1000
```

3.33.1.2 __getattr__ (self, attr)

3.33.1.3 __setattr__ (self, attr, value)

3.33.1.4 __dir__ (self)

3.33.1.5 def FromImage (image, rect = None, threshold = 98)

Description

Generate a trigger condition from an image. This will make the sdo for you and set it up to have one image compare region. You should consider the FromIcon method for optimum performance as full screen images take time to send over the network.

Parameters:

← **image** StormTestImageObject : the image to use as the reference. It should be the same size as the frames you are analyzing.

← **rect** List or Tuple : (optional) the region of interest within the image. It must be a list or tuple of 4 elements representing left, top, width and height of the rectangle to use for comparison. It can be None to mean the whole image should be compared.

← **threshold** double : (optional) the threshold for the image match as per CompareImageEx.

Returns:

TriggerCondition : the completed trigger condition ready to use.

Example:

```
>>>create a trigger to trigger on image but using rect at (0,0,400,300)
>>>trigger = TriggerCondition.FromImage(image, [0,0,400,300])
```

3.33.1.6 def FromIcon (image, location, threshold = 98)

Description

Generate a trigger condition from an icon. This will make the sdo for you and set it up to have one image compare region.

Parameters:

← **image** StormTestImageObject : the image to use as the reference icon.

← **location** List or Tuple : the location of the icon within the image. It must be a list or tuple of 2 elements representing left, top of the icon's location.

← **threshold** double : (optional) the threshold for the icon match as per CompareIconEx.

Returns:

TriggerCondition : the completed trigger condition ready to use.

Example:

```
>>>create a trigger to trigger on icon but using location at (89,77)
>>>trigger = TriggerCondition.FromIcon(image, [89,77])
```

3.33.1.7 def FromColor (rect, color, tolerance = (4,4, flatness = 98, peakError = 0)

Description

Generate a trigger condition from a color. This will make the sdo for you and set it up to have one color compare region.

Parameters:

← **rect** List or Tuple : the region of interest within the screen capture. It must be a list or tuple of 4 elements representing left, top, width and height of the rectangle to use for color comparison. It can be None to mean the whole image should be compared.

← **color** List or Tuple : the color as a List or Tuple of Red, Green, Blue values.

← **tolerance** List or Tuple : (optional) the tolerance to apply to color comparison as per CompareColorEx function. Default is (4,4,4)

← **flatness** double : (optional) the flatness of the color as per CompareColorEx function. Default is 98.

← **peakError** double : (optional) the peak error of the color as per CompareColorEx funcion. Default is 0.

Returns:

TriggerCondition : the completed trigger condition ready to use.

Example:

```
>>>create a trigger to trigger on color black at (0,0, 400,100)
>>>trigger = TriggerCondition.FromColor((0,0,400,100), (0,0,0))
```

3.34 VideoFormat Enum Reference

Static Public Attributes

- string MPEG2 = "MPEG2"
- string H264 = "H264"

Description

VideoFormat Enum. Defines the valid values of video format supported by VQA (and any other future API that requires knowledge of video format)

Parameters:

MPEG2 : Video compressed to MPEG-2

H264 : Video compressed to ITU.T H.264 (also known as MPEG-4 Part 10 and AVC)

3.35 VideoSignalState Enum Reference

VideoSignalState Enum.

Static Public Attributes

- VidSigLoss = stormtest_client.SlotState.VID_SIG_LOSS
- VidSigGain = stormtest_client.SlotState.VID_SIG_GAIN

VideoSignalState Enum.

Defines the status of a video signal. This is used when monitoring the video from a DUT to see if the video signal disappears or reappears.

Parameters:

VidSigLoss	: video signal lost
VidSigGain	: video signal regained

3.36 VideoStandard Enum Reference

Description

VideoStandard Enum. Defines the valid values that can be passed to the SetVideoStandard() function.

Parameters:

PAL	PAL video standard
NTSC	NTSC video standard

Example:

```
>>SetVideoStandard(VideoStandard.NTSC)
```

3.37 VQAAudioResult Enum Reference

Public Member Functions

- __init__
- __getattr__
- __setattr__
- __dir__

Public Attributes

- FrameNumber
- TimeStamp
- MOS
- MOSDescription
- AudioQuality
- AudioQualityTemporal
- CustomSilences
- Silences
- SilenceDuration
- Saturations
- Breaks
- EstimatedBandwidth
- HighFrequencyPresent

Private Attributes

- __cache

Static Private Attributes

- dictionary __cache = {}

Description

The VQA audio result object. The properties here indicate various metrics about the audio analyzed. Audio is analyzed in chunks of 1 second, not video frame lengths. However, not all properties are necessarily valid for all result records - in that case the value will be None instead of the value described here. If there is a long period of pure silence, the MOS score is None (a human would not say silence is 'good' or 'bad' but rather 'there is no audio'.
The low level hardware supplies audio in packets of 8192 bytes (42.67 mS at 48 kHz audio) so 1 second is not a complete number of packets. For this reason, you do not get exactly 1 result per 1 second of 30 frames of video (at 30 frames per second). The audio is processed continually without error - the first audio result is generated when 24 packets of 42.67 mS have been received (1.024 seconds).

Parameters:

FrameNumber integer : The frame number of this result. The first result analyzed is frame 30 +/-1 at a video rate of 30 frames per second. Frame numbers are counted from the captured video frame and will increment at the raw video capture frame rate (+/- 1), due to 1 result per second.

TimeStamp double : the timestamp of the result. This is in seconds since the Unix epoch (1/1/1970 00:00:00 UTC) and is the time on the server when the result was generated. This may increase unevenly due to threading and CPU load. It is useful for debugging should problems occur.

MOS double : The raw MOS score. This is a number normally between 0 and 100. However, if the XXXInfluence parameters are set to high values and lots of errors occur, the result can be negative. The scores are:

MOS Score	Meaning
< 20	Bad Quality
20 - 40	Poor Quality
40 - 60	Rather Good Quality
60 - 80	Good Quality
80 - 100	Excellent Quality

These terms are defined in standards such as BT.500 and BT.710. The MOS score is an average of the qualities over time. It uses a 10 second window for the averaging so the first 10 seconds worth of values after starting are not valid.

MOSDescription string : The String equivalent of the MOS score without the word Quality (i.e. Bad, Poor, Rather Good, Good or Excellent)

AudioQuality double : The quality of a single 1 second piece of audio. The range and meaning is the same as MOS score. MOS score is the average of the raw audio qualities. The AudioQuality ignores the effect of prior frames.

AudioQualityTemporal double : The quality of a single 1 second piece of audio taking into account the previous pieces' scores. This takes into account a property of the human audio system which is that we are "quick to criticize, slow to forgive". In order to understand this sentence, consider human observers listening to a good quality audio track. Now imagine that an audible distortion appears. When the distortion appears, the quality judgment of the human listeners decreases nearly instantly ("quick to criticize"). However, when this distortion disappears, their human judgment takes several seconds to come back to its previous state that was judging that the audio has a good quality ("slow to forgive"). Humans are generally more sensitive to audio distortion than video, so a longer (10 second) window is used for temporal calculations. This is the temporal effect implemented in this score computation. It indicates the quality of each piece of audio, taking into account the previous judgments.

CustomSilences double : Number of silences detected based on the custom silence detection algorithm selected. It will be a number >= 0.

Silences double : The number of audio silences detected using the standard silence detection algorithm.

SilenceDuration double : The duration of audio silences detected using the standard silence detection algorithm (each silence is >= 100mS, this value is the sum of all silences).

Saturations double : The number of times audio saturation was detected in the 1 second piece of audio.

Breaks double : The number of breaks detected in the 1 second piece of audio.

EstimatedBandwidth double : The estimaed bandwidth of the audio sample. The maximum bandwidth is the audio sample rate/2. A low value of the estimated bandwidth indicates possible distortion (but it may also simply be a feature of the source audio).

HighFrequencyPresent double : A value indicating the presence of high frequencies in the 1 second of audio. It is an adaptive calculation which takes into account the ratio between low and high frequencies. Values > 0.01 are generally OK, values lower than 0.0005 indicate heavy distortion.

3.37.1 Member Function Documentation

3.37.1.1 __init__ (self)

3.37.1.2 __getattr__ (self, attr)

3.37.1.3 __setattr__ (self, attr, value)

3.37.1.4 __dir__ (self)

3.38 VQAEngine Enum Reference

Public Member Functions

- __init__
- __getattr__
- __setattr__
- __dir__
- Start
- Stop
- Close
- WaitEvent

Private Attributes

- __cache

Static Private Attributes

- dictionary __cache = {}

Description

The VQA Engine class. This is the core engine. You can create this object directly or use VQACreateEngine function. This engine can be passed to functions such as VQAStart or you can call methods directly on this object.

Parameters:

VideoOptions VQAVideoOptions : The video options used to create the engine. This is read only. You cannot update the options once the engine has been created.

AudioOptions VQAAudioOptions : The audio options used to create the engine. This is read only. You cannot update the options once the engine has been created.

Status VQAEngineStatus : The status of the engine. It is a value from the VQAEngineStatus. You cannot write to the status - it is read only and updated by the engine.

Slots List : List of slots that this engine is using. It will be None until the engine has been started and after that it will be the list of slots in use. You cannot replace this list and if you try to update the contents, then you will be confused (Python won't stop you modifying the contents). The values are the logical slots when under the daemon and can be passed to other API functions that take a slot number.

3.38.1 Member Function Documentation

3.38.1.1 __init__ (self, videoOptions = None, audioOptions = None)

Description

The constructor makes a VQAEngine object.

Parameters:

← **videoOptions VQAVideoOptions** : The video options to use. This may be None (the default) to get the default settings. You cannot change the options once the engine has been made.

← **audioOptions VQAAudioOptions** : The audio options. This may be None (the defautl) to get the default settings. You cannot change the options once the engine has been made.

Note:

vqa = VQAEngine(videoOptions, audioOptions) is the same as vqa = VQACreateEngine(videoOptions, audioOptions) for those who prefer a function based API.

Returns:

VQAEngine : The completely constructed VQAEngine object.

Example:

```
>>>voptions = VQAVideoOptions()
>>>voptions.Backlog = 1500
>>>aoptions = VQAAudioOptions()
>>>aoptions.SilenceInfluence = 10
>>>vqa = VQAEngine(voptions, aoptions)
>>>print vqa.Status
>>>Idle
```

3.38.1.2 __getattr__ (self, attr)

3.38.1.3 __setattr__ (self, attr, value)

3.38.1.4 __dir__ (self)

3.38.1.5 def Start (self, trigger = None, callbackFunction = None, slotNo = 0)

Description

Start VQA analysis on one or more slots. Multi slot VQA is an advanced issue and this release of StormTest does not support that. Only 1 slot may be used. Once started, the VQA analysis proceeds asynchronously. You may wait for results or be notified using a callback. The VQA will run until you stop it or until the slot is released.

Parameters:

← **trigger TriggerCondition** : (optional) The trigger condition to start VQA analysis. If omitted analysis starts immediately. You can set up a trigger condition to start VQA analysis on a specific keystroke or when a specific video frame is detected.

callbackFunction[in] VQACallback : (optional) The callback function to be called whenever an event of interest occurs. The callback is called on another thread and you must be aware of that when handling the events. You should not rely upon calling arbitrary StormTest functions. However, you can call Stop and Close methods on the VQAEngine.

← **slotNo Integer** : (optional) The slot number to use. If omitted or set to 0 then all reserved slots will be used. In this release, if 0 is used for slotNo, then an exeption will be thrown if your script has reserved 2 or more slots.

Returns:

None

Exceptions:

StormTestExceptions is thrown on an error (for example bad parameters, not enough memory, invalid slot, no communication).

Example:

```
>>>vqa = VQAEngine()
>>>vqa.Start()              # start on all reserved slots immediately and without a callback
```

callbackFunction(slotNo, engine, event).

Parameters:

← **slotNo Integer:** The slot number related to the event

← **engine VQAEngine:** The engine that is calling the function (this allows a single callback to manage many engines)

← **event VQAEvent:** The event object

Returns:

None

Note:

You should return None from your callback - future versions of StormTest may define the return value to mean something but it will know that a value of None means the script was written for version 3.2.3 and later versions will ensure correct behavior for such scripts.

3.38.1.6 def Stop (self, trigger = None)

Description

Stop a running VQA analysis. This will stop further frames being queued up for analysis but will not stop any pending frames from being analysed. It is normal for there to be some frames waiting analysis - these will be analysed after Stop is called. Once you have called Stop() you may not call Start() again. To start a new analysis, create a new VQAEngine.

Parameters:

← **trigger TriggerCondition** : (optional) the trigger condition to stop the analysis. If omitted analysis stops immediately. In release 3.2.3, only the FrameCount and Delay options of triggering are supported. In release 3.2.5, if you are only doing audio VQA Analysis then only the Delay option is supported - FrameCount must be None. This function is asynchronous and returns immediately.

Returns:

None

Exceptions:

StormTestExceptions is thrown on an error (engine not running, invalid paramters, no communication)

Note:

If you create a new VQAEngine and start it on the same slot after calling Stop() but before calling Close() then the system will automatically call Close() for you on the VQAEngine() which had been stopped but not closed.

Example:

```
>>>vqa = VQAEngine()
>>>vqa.Start()
>>>stopCondition = TriggerCondition()
>>>stopCondition.Delay = 30
>>>vqa.Stop(stopCondition)    # stop on a trigger. That is set to 30 seconds so the stop will stop 30 seconds in the fut
```

3.38.1.7 def Close (self)

Description

Closes a VQAEngine, stopping it if necessary and freeing all resources associated with the engine. Any pending frames will not be analyzed and after calling Close(), you cannot call any other method on the engine. You should call this once you are finished with the analysis and you have got all your results. This ensures all memory used by the VQA has been freed. You may receive a callback after Close() is called due to the asynchronouse nature of callbacks.

Parameters:

None

Returns:

None

Note:

If you create a new VQAEngine and start it on the same slot after calling Stop() but before calling Close() then the system will automatically call Close() for you on the VQAEngine() which had been stopped but not closed.

Example:

```
# example to just run for a while, get no results and stop everything. Not the most useful program ever written.
>>>vqa = VQAEngine()
>>>vqa.Start()
>>>WaitSec(5)
vqa.Close()                    # stop everything and clean up
```

3.38.1.8 def WaitEvent (self, eventList)

Description

Wait for one or more events to occur. You should not use this function for an event type AND also process the same event type in the callback - this will cause confusion. You should choose one method of the other. If you specify a callback then events are posted to the callback. If the callback handles the event then it is no longer available to be retrieved from WaitEvent(). If an event is retrieved via WaitEvent() then it will be still posted to the callback if a callback exists.

Parameters:

← **eventList** List of VQAEventType: list of events. The call will return when any one of those events occur. You can set the list to be empty to mean 'any event'. The call will return on the first event.

Returns:

events List : List of VQAEvent objects. Each event object identifies its type and some optional parameters.

Note:

If you specify an event list which can never occur then this function will never return.
It is the script writers responsibility to process events in a timely manner. The queue of events is limited - an event will be held for at least 15 seconds.

Example:

```
>>>vqa = VQAEngine()
>>>vqa.Start()
>>>WaitSec(2)
>>>events = vqa.WaitEvent([])
>>>print "We have {0} events".format(len(events))
>>>We have 25 events
```

3.39 VQAEngineStatus Enum Reference

Static Public Attributes

- string Idle = "Idle"
- string Run = "Run"
- string Stop = "Stop"
- string Complete = "Complete"
- string Closed = "Closed"

Description

VQAEngineStatus Enum. This indicates the status of the VQAEngine.

Parameters:

Idle : The engine is idle, it has not started analysis (this occurs from creation until the trigger condition specified in Start() has been met.

Run : The engine is running and actively analyzing video samples.

Stop : The engine has been stopped via a call to Stop() has been called. The engine sets this state immediately upon you calling Stop(). However, if a trigger condition with a delay has been specified, the Stop event will be received when the delay expires. The reason is that you can check the engine status to avoid calling stop multiple times. The actual stopping of the engine is asynchronous because it is on the server. The engine will continue to analyze samples previously captured that remain in the backlog queue.

Complete : The engine has completed its analysis, including all the samples in the backlog queue. No further action will be taken by the VQA engine

Closed : The engine has been closed by using the Close() function. Any further calls to the engine are forbidden and will raise an exception. You can use this state to check if Close() has been called, perhaps in a different thread by your application.

3.40 VQAEvent Enum Reference

Public Member Functions

- __init__
- __getattr__
- __setattr__
- __dir__

Public Attributes

- Type
- TimeStamp
- Value
- FrameNumber

Private Attributes

- __cache

Static Private Attributes

- dictionary __cache = {}

Description

The event class. An instance of this is returned to the user script for every event of interest that occurs during VQA processing. It is not a requirement that all scripts process all events, many are for information which might be useful in some situations. A script can choose which events to process.

Parameters:

Type VQAEventType : type of event as an enumeration. This indicates what the event signifies.

Value object : extra values about the event. This may be None but for some events, extra data

TimeStamp double : The time stamp when the event was generated. This is the time, in seconds since the Unix epoch (1/1/1970 00:00:00 UTC) and is related to the value returned from time.time() function. This timestamp is generated on the server so can be slightly different from time.time() if the client and server machine are not synchronised.

FrameNumber Integer : The frame number associated with the event. Frame 0 is the first frame AFTER the trigger condition is satisfied. Frames then increment sequentially as determined by the source video stream.

3.40.1 Member Function Documentation

3.40.1.1 __init__ (self)

3.40.1.2 __getattr__ (self, attr)

3.40.1.3 __setattr__ (self, attr, value)

3.40.1.4 __dir__ (self)

3.41 VQAEventType Enum Reference

Static Public Attributes

- string Start = "Start"
- string Stop = "Stop"
- string Finish = "Finish"
- string DroppedFrame = "DroppedFrame"
- string ResChange = "ResChange"
- string VideoResult = "VideoResult"
- string AudioResult = "AudioResult"

Description

VQAEventType Enum. Defines the types of event - it is a field in the VQAEvent object. Depending on the value of this, the Value field of VQAEvent may have useful information. If nothing is specified below then the Value will be None

Parameters:

Start : The analysis has started after any trigger condition has been satisfied.

Stop : The analysis has stopped accepting new video frames but is still processing any pending frames.

Finish : The analysis has processed all frames in the queue after Stop. No new data will be generated.

DroppedFrame : The engine has been forced to drop frames due to overload. The Value is an Tuple indicating the number dropped since the last DroppedFrame event and the total dropped since the start of analysis. Frames discarded due to the user setting a low frame rate in the options, are NOT counted or reported.

ResChange : The resolution of the video has changed (the source maybe has switched video size). The Value is a tuple of (width, height, frameRate) of the new resolution. After a resolution change, video analysis stops. Any pending frames at the original resolution are processed but no new frames are processed.

VideoResult : Results of video VQA are available. The Value is a List of VQAVideoResult objects. This event is typically generated every few seconds during analysis. In any case, it will not be generated more often than once per second.

AudioResult : Results of audio VQA are available. The Value is a List of VQAAudioResult objects. This event is typically generated every few seconds during analysis. In any case, it will not be generated more often than once per second.

3.42 VQASilenceMode Enum Reference

Static Public Attributes

- string Disabled = "Disabled"
- string Amplitude = "Amplitude"
- string StdDev = "StdDev"

Description

VQASilenceMode Enum. Defines the valid values of custom audio silence detector supported by VQA

Parameters:

Disabled : The custom audio silence detector is disabled.

Amplitude : The custom audio silence detector examines the value of audio samples (the amplitude) looking for values below a threshold.

StdDev : The custom audio silence detector examines the standard deviation of a group of samples (1 frame, equal in length to the video frame time) and calculates the standard deviation of the amplitudes and examines that in order to determine silence.

3.43 VQAVideoOptions Enum Reference

Public Member Functions

- __init__
- __getattr__
- __setattr__
- __dir__

Public Attributes

- Enabled
- Format
- CaptureIsUpscaled
- Backlog
- FrameRate
- Jerkiness
- Activity
- ChangingPixels
- MeanBrightness
- DesignSize
- Crop

Private Attributes

- __cache

Static Private Attributes

- dictionary __cache = {}

Description

The VQA Video options class. This holds all the options that can be configured for VQA video analysis. You can create a default object with the default parameters and adjust those that are necessary.

Parameters:

Enabled Bool : Whether video analysis should be performed. This is a new property in 3.2.5 to allow the options of audio only VQA analysis. Default is True.

Format VideoFormat : The format of the source of the video that was supplied to StormTest. Most HD Video uses H.264 and this is the default value for the Format value. You should change this to MPEG2 ONLY if you know for certain that the video was originally encoded using MPEG2. StormTest processes raw video frames but these are supplied by a device - that device will have decoded the video from either H.264 or MPEG2 video. Getting this value 'wrong' will alter how the VQA engine assesses the quality of the video and thus may generate unexpected scores for the quality. If another format was originally used, then the VQA results from StormTest are not necessarily representative of human perception of quality.

CaptureIsUpscaled Bool : Whether the captured image is upscaled from the source image. The VQA engine is designed to work on captured images which are the same size as the original source material. However, it may be that StormTest is capturing images from the device under test at a higher resolution than the original source material. In order to get more realistic MOS scores in this situation, set CaptureIsUpscaled to True. Default is False. This setting was introduced in release 3.3.0

Backlog Integer : The number of frames used as a backlog queue. It is possible that video is captured more quickly than it can be analyzed. In this case, StormTest keeps a queue of pending frames. When that queue is full, then frames are dropped. This affects the MOS scores. Setting a high value uses more memory but allows a longer period from the start of video analysis before frames are dropped. The minimum value is 16. The maximum depends on available memory: for an HD system with 8GB of RAM then the maximum is around 1500 for 1920 x 1080 frames. The default value is 500.

FrameRate Float : The frame rate to use for analysis. If the actual frame rate is greater than this value then frames are dropped in a deterministic manner to bring the frame rate down to the FrameRate value. if you are monitoring video for an extended period and the engine cannot process all frames then a deterministic drop of frames will typically give better results (more consistent) than the non deterministic dropping on a full queue. Best results are achieved where the number of frames dropped can be expressed as 1 in n where n is integer, for example 20 for a 25 fps input video (1 in 5 frames dropped) A value of 0 should be used to mean 'same as input rate'

Jerkiness Bool : Set to true to permit 'jerkiness' or 'image freezing' of the video to affect MOS score. If you expect smooth video then you should set this to True. If, however, you expect your video to be jerky with a lot of static frames then you may get a more realistic MOS score by setting this to False. The default is True. You can adjust the detection parameters of what is considered a 'static picture'. A picture is moving when it exceeds the Activity OR the ChangingPixels OR the MeanBrightness thresholds.

Activity Float : A threshold indicating temporaral activity, above which, we consider the video to be moving. It ranges from 0 to 255. Values lower than 5 are consider low motion, 5 to 15 is normal motion. Default value for Activity is 0.9 - only very slow moving pictures are considered 'static'. This parameter has no effect if Jerkiness is False

ChangingPixels Float : The percentage of pixels (0 - 100) which have a significant change compared to the previous frame. If the number exceeds this value, then the picture is considered moving. The default is 0.01. This parameter has no effect if Jerkiness is False

MeanBrightness Float : The average value of brighness, above which the image is considered moving. The default value is 21 meaning that if the image is not black, it is considered to be moving. This means that jerkiness is likely to be detected only on very dark pictures or black screens. Setting this to a high value would allow freezing to be detected on any brightness of screen. This parameter has no effect if Jerkiness is False

Crop List : A list of 4 integers representing a rectangle of the video in which the video analysis is performed. The rectangle is specified as left, top, width, height. It can be None to indicate no cropping and analysis on the full frame. If the specified rectangle is greater than the video size then the full frame will be used. After the VQA Engine has started, the Crop value on the VideoOptions property of the engine can be read to find out the region of the video being analyzed. The final crop region will be adjusted so that it is an even number of pixels in size in each direction and the top left corner is on even pixels in each direction. The minium final size of the Crop region is 176 x 144. A StormTestExceptions exception will be raised if the crop region is too small. The default value for Crop is None. This property was introduced in version 3.3.3

DesignSize List : A list of 2 integers representing the size of the video for which the options are designed. This affects how the Crop is interpreted. If the DesignSize is None then the Crop is considered to be an absolute rectangle. If the DesignSize is not None then the VQA engine will scale the Crop rectangle as necessary so that the same intended region will be analyzed if the actual video is not the same as DesignSize. As an example, consider the case of DesignSize being [1920, 1080] and the Crop being [100, 50, 600, 300] but the raw video is 1280 x 720. In this case analysis will be performed on the rectangle [66, 34, 400, 200] because 1280 x 720 is 2/3 the size of 1920 x 1080 (The region is rounded to nearest even coordinates). The default value for DesignSize is None. This property was introduced in version 3.3.3

3.43.1 Member Function Documentation

3.43.1.1 __init__ (self)

Description

The constructor provides an object with the default VQA options. You can then modify any option as needed.

Parameters:

None

Returns:

VQAVideoOptions : The completely constructed VQAVideoOptions object.

Example:

```
>>>vqaOptions = VQAVideoOptions()
>>>print vqaOptions.Jerkiness, vqaOptions.Backlog, vqaOptions.Format
>>>True 500 H264
>>>vqaOptions.Backlog = 1500
>>>print vqaOptions.Jerkiness, vqaOptions.Backlog, vqaOptions.Format
>>>True 1500 H264
```

3.43.1.2 __getattr__ (self, attr)

3.43.1.3 __setattr__ (self, attr, value)

3.43.1.4 __dir__ (self)

3.44 VQAVideoResult Enum Reference

Public Member Functions

- __init__
- __getattr__
- __setattr__
- __dir__

Public Attributes

- FrameNumber
- TimeStamp
- MOS
- MOSDescription
- FrameQuality
- FrameQualityTemporal
- Jerkiness
- Blockiness
- Blur
- Contrast
- TemporalActivity
- MotionVector
- YLevels
- ULevels
- VLevels

Private Attributes

- __cache

Static Private Attributes

- dictionary __cache = {}

Description

The VQA video result object. The properties here indicate various metrics about the video frames analyzed. There will be one result per frame of video processed. However, not all properties are necessarily valid for all frames - in that case the value will be None instead of the value described here.

Parameters:

FrameNumber integer : The frame number of this result. The first frame analyzed is frame 0. Frame numbers are counted from the captured frame and include all dropped frames (whether intentionally or not). To be clear, if the queue is NOT overloaded and VQAVideoOptions.FrameRate is set to 15 for a 30 frame/sec source then you will get results with frameNumber value: 0, 2, 4, 6, 8 ... due to the deliberate dropping of frames.

TimeStamp double : the timestamp of the result. This is in seconds since the Unix epoch (1/1/1970 00:00:00 UTC) and is the time on the server when the result was generated. This may increase unevenly due to threading and CPU load. It is useful for debugging should problems occur.

MOS double : The raw MOS score. This is a number between 0 and 100. The scores are:

MOS Score	Meaning
0 - 20	Bad Quality
20 - 40	Poor Quality
40 - 60	Rather Good Quality
60 - 80	Good Quality
80 - 100	Excellent Quality

These terms are defined in standards such as BT.500 and BT.710. The MOS score is an average of the frame qualities over time (as no human can determine the quality of a single frame at 30 frames per second). It uses a 5 second window for the averaging so the first 5 seconds worth of values after starting are not valid.

MOSDescription string : The String equivalent of the MOS score without the word Quality (i.e. Bad, Poor, Rather Good, Good or Excellent)

FrameQuality double : The quality of a single frame. The range and meaning is the same as MOS score. MOS score is the average of the raw frame qualities. The FrameQuality ignores the effect of prior frames.

FrameQualityTemporal double : The quality of a single frame taking into accout the previous frames' scores. This takes into account a property of the human visual system which is that we are "quick to criticize, slow to forgive". In order to understand this sentence, consider human observers watching a good quality video. Now imagine that a visible distortion appears. When the distortion appears, the quality judgment of the human observers decreases nearly instantly ("quick to criticize"). However, when this distortion disappears, their human judgment takes several seconds to come back to its previous state that was jusging that the video has a good quality ("slow to forgive"). This is the temporarl effet implemented in this score computation. It indicates the quality of each frame, taking into accout the previous judgments.

Jerkiness double : The minimum value is 0 and no maximum. It is the duration in mS during which video was frozen.

Blockiness double : This indicates the blockiness on a range from 0 to 255. It expresses the importance of the frontier between a block which is suspected as distored and its neighbourhood. Typical values: 0 - 3 : OK, above 3 : distorted.

Blur double : Blur ranges from 0 to the diagonal length of the picture (sqrt(width $*$ width + height $*$ height)). It indicates the bluriness of the image. Values below 2 are OK, above 2 indicates a blurry picture. This of course, may be the intention of the source video for artistic effect.

Contrast double : This ranges from 0 to 2 and expresses the importance of details in the video, especially around macroblock boundaries where they are submitted to the H.264 deblocking filter. This value only has meaning when the original video is H.264. Values less than 0.15 indicate not enough detail, good video has values above 0.15

TemporalActivity double : This ranges from 0 to 255 and indicates how the pixel values change between 2 successive frames. Typical values are 0 - 5 : low motion, 5 - 15 : normal motion and >15: important motion.

MotionVector Tuple : A tuple of the global motion vectors in (X,Y) directions indicating the general motion of this frame relative to the previous frame.

YLevels Tuple : Tuple of minimum Y, maximum Y, average Y and standard deviation of Y. Y is the brightness (luminance) of the video frame. All values are doubles between 0 and 255. The mean values can be interpreted as 0 - 50 : dark content, 50 - 150 : normal content, and above 150 : light content. For the standard deviation values less the 20 indicate a homogeneous content.

ULevels Tuple : Tuple of minimum U, maximum U, average U and stardard deviation of U. The U value is the difference between Blue and the Brightness (part of YUV color space). In this release the average and standard deviation will always be None as it is not supported.

VLevels Tuple : Tuple of minimum V, maximum V, average V and stardard deviation of V. The V value is the difference between Red and the Brightness (part of YUV color space). In this release the average and standard deviation will always be None as it is not supported.

3.44.1 Member Function Documentation

3.44.1.1 __init__ (self)

3.44.1.2 __getattr__ (self, attr)

3.44.1.3 __setattr__ (self, attr, value)

3.44.1.4 __dir__ (self)

3.45 ClientAPI.py File Reference

The ClientAPI module provides APIs for user to script tests and talks to server.

Enums

- enum SignalRecoveryMode
- enum TM
- enum TestResult
- enum SystemCapability
- enum LogRegionStyle
- enum OverWriteAction

    OverWriteAction Enum.

- enum CompareAlgorithm
- enum SaveScreenCap
- enum HDMIColorSpace
- enum VideoStandard
- enum VideoSignalState

    VideoSignalState Enum.

- enum CoordConverter
- enum MetaDataType
- enum OutputTo
- enum DebugLevel
- enum ReturnScreen
- enum OCRStringCorrection
- enum OCRRegion
- enum ImageRegion
- enum ColorRegion
- enum MotionRegion
- enum AudioRegion
- enum Color
- enum RectangleMatrix
- enum CustomRegion
- enum AlgorithmDescriptor
- enum ScreenDefinition
- enum ProcessImage
- enum AudioChannel
- enum OffHookMode
- enum TLSStatus
- enum PerceptualParameters
- enum VideoFormat
- enum VQAVideoOptions
- enum VQASilenceMode
- enum VQAAudioOptions
- enum TriggerCondition
- enum VQAEventType

- enum VQAEvent
- enum VQAVideoResult
- enum VQAAudioResult
- enum VQAEngineStatus
- enum VQAEngine
- enum MeasurementTime

  MeasurementTime Enum.

- enum TouchType_iOS

Functions

- GetFacilityData
- GetServerNames
- GetServerField
- GetStbField
- ConnectToServer
- ReleaseServerConnection
- ReserveSlot
- ReserveStb
- SetCurrentSubslot
- GetCurrentSubslot
- FreeSlot
- KillSlot
- ResetSlot
- SetSignalRecoveryMode
- GetSignalRecoveryMode
- GetLogicalReservedSlots
- GetPhysicalAllocations
- IsUnderDaemon
- ReturnTestResult
- BeginTestStep
- EndTestStep
- GetClientVersion
- CheckSlots
- GetReservedStatus
- WaitSec
- HasCapability
- GetCapabilities
- BeginLogRegion
- EndLogRegion
- PressButton
- EnablePressDigitsCriticalSection
- PressDigits
- GetCountIRInterfaces
- SetIRControlInterface
- GetIRControlInterface
- IRSetSignalPower

- OpenNavigator
- OpenNavigatorFile
- CloseNavigator
- GetNavigatorScreens
- ValidateNavigatorScreen
- NavigateTo
- ValidateNavigator
- StopNavigator
- GetValidNavigators
- GetCurrentNavigator
- ConvertNavigatorToScreenDef
- UploadPublicNavigator
- DownloadPublicNavigator
- ListPublicNavigators
- PowerOnSTB
- PowerOffSTB
- IsSlotPowerOn
- AddObserve
- RemoveObserve
- GetObserveLog
- GetSerialLog
- StartSerialLog
- StopSerialLog
- SetExtParser
- ChangeSerialParameters
- SendCommandViaSerial
- SetRawDataReceiver
- SendRawData
- CommentToLog
- SetVideoCpuUsage
- ShowVideo
- CloseVideo
- HideVideo
- DetectMotionEx
- CaptureImageEx
- CompareImageEx
- CompareIconEx
- WaitImageMatch
- WaitImageNoMatch
- WaitIconMatch
- WaitIconNoMatch
- CompareColorEx
- WaitColorMatch
- WaitColorNoMatch
- SetVideoOffset
- GetVideoOffset
- CalibrateVideoOffset
- StartVideoLog

- StopVideoLog
- EnableRollingRecord
- DisableRollingRecord
- SaveRecording
- BookmarkVideoLog
- GetAudioLevel
- WaitAudioPresence
- WaitAudioAbsence
- ResyncAudio
- IsAudioResyncNeeded
- SetResolution
- GetResolution
- GetRawInputInfo
- SetMaxBitRate
- GetMaxBitRate
- SetFrameRate
- GetFrameRate
- SetVideoStandard
- ResetDefaultParameters
- RegisterVideoLossCallback
- SetDesignToWorldTransform
- GetDesignToWorldTransform
- SetWorldToDesignTransform
- GetWorldToDesignTransform
- SetProjectDir
- SetDirectories
- GetProjectDir
- GetDirectories
- GetLogFileDirectory
- SetResourcePath
- GetResourcePath
- SetNamedResourceMatchCase
- GetNamedResourceMatchCase
- FindNamedRegion
- FindNamedColor
- FindNamedString
- FindNamedImage
- FindNamedScreenDefinition
- WriteNamedRegion
- DeleteNamedRegion
- ReadNamedRegion
- RenameNamedRegion
- WriteNamedImage
- DeleteNamedImage
- ReadNamedImage
- RenameNamedImage
- WriteNamedString
- DeleteNamedString

- ReadNamedString
- RenameNamedString
- WriteNamedColor
- DeleteNamedColor
- ReadNamedColor
- RenameNamedColor
- WriteNamedScreenDefinition
- DeleteNamedScreenDefinition
- ReadNamedScreenDefinition
- RenameNamedScreenDefinition
- CreateNamedFolder
- DeleteNamedFolder
- RenameNamedFolder
- ListNamedResources
- AddMetaData
- RemoveMetaData
- ReadMetaData
- SetDebugLevel
- SetLogPageSize
- WriteDebugLine
- SetHtmlThumbnailSize
- GetHtmlThumbnailSize
- WaitScreenDefMatch
- WaitScreenDefNoMatch
- ScreenDefMatch
- ScreenDefMatchImage
- TestRawZapFrames
- FindRawZapMatch
- FindRawZapNoMatch
- OCRSlot
- WaitOCRMatch
- WaitOCRNoMatch
- OCRFile
- OCRImage
- OCRGetRemainingChars
- OCRSetLanguage
- OCRGetLanguage
- OCRSetDefaultImageProcessing
- OCRGetDefaultImageProcessing
- OCRSetImageFiltering
- OCRGetImageFiltering
- CompareStrings
- ComputeStringsDistance
- StringsAutoCorrection
- ReserveVideoTimer
- FreeVideoTimer
- StartZapMeasurement
- TriggerHighSpeedTimer

- StopHighSpeedTimer
- GetZapTimes
- GetRawZapMeasurements
- EnableCaptureZapFrames
- GetCountImagesSaved
- GetRawZapFrame
- TriggerOsdMask
- StopOsdMask
- TriggerOsdString
- StopOsdString
- SelectInputFeed
- GetInputFeed
- SetAVInput
- GetAVInput
- GetCapsVideoInput
- GetCapsAudioInput
- GetRfFormats
- GetRfTunerConfiguration
- ConfigureRfTuner
- ConfigureHdDownConverter
- SetTLSOffHookMode
- GetTLSOffHookMode
- RetrieveTLSStatus
- RetrieveTLSLastDialed
- ClearTLSLastDialed
- StormTestLicenseDetails
- OCRLicenseDetails
- SetMasterConfigurationServer
- CreateDutInstanceInSlot
- UpdateDutInstance
- UpdateDutInstanceInSlot
- GetDutInstanceInSlot
- GetDutDetails
- GetSoftwareComponentsSupported
- AddSwComponentToDut
- GetSwComponentFromDut
- GetAllSmartcards
- SetSmartcardNumber
- GetSmartcardNumber
- SetSmartcardAttribute
- GetSmartcardAttribute
- GetHwComponents
- FindModelsByHardware
- CreateDutModel
- UpdateDutModel
- DeleteDutModel
- ListDutModels
- VQACreateEngine

- VQAStartAnalysis
- VQAStopAnalysis
- VQACloseAnalyzer
- VQAWaitEvent
- VQACreateTrigger
- VQACreateTriggerFromImage
- VQACreateTriggerFromIcon
- VQACreateTriggerFromColor
- VQACreateVideoOptions
- VQACreateAudioOptions
- VQAGetLicenseInfo
- VQAIsSlotLicensed
- StartVideoAnalysis
- StopVideoAnalysis
- SetVideoAnalysisParameters
- ReadVideoCalibrationData
- WriteVideoCalibrationData
- DeleteVideoCalibrationData
- GetVideoAnalysisResult
- GetVideoAnalysisData
- ClearVideoAnalysisResults
- ResetAudioAnalysisParameters
- SetAudioAnalysisParameters
- ReadAudioCalibrationData
- WriteAudioCalibrationData
- DeleteAudioCalibrationData
- SetAudioMeasurementTime
- GetAudioMeasurementTime
- SetAudioAnalysisChannel
- GetAudioAnalysisChannel
- SetAudioToneParameters
- GetAudioToneParameters
- SetAudioImpulseParameters
- GetAudioImpulseParameters
- SetSubBandPowerBandList
- GetSubBandPowerBandList
- SetOutOfBandPowerBandList
- GetOutOfBandPowerBandList
- SetSpuriousToneBandList
- GetSpuriousToneBandList
- StartAudioAnalysis
- StopAudioAnalysis
- GetOverallAudioAnalysisStatus
- GetResultPowerSpectrum
- GetResultToneDetection
- GetResultAverageToneAmplitude
- GetResultSpuriousTone
- GetResultSubbandPower
- GetResultOutOfBandPower
- GetResultImpulseNoise
- GetJobConfigurator

Variables

- STORMTEST_PUBLIC = stormtest_client.public_area
- STORMTEST_PRIVATE = stormtest_client.private_area
- _use_legacy_server_colors = False
- AllowIgnoredScreens = False

3.45.1 Detailed Description

The ClientAPI module provides APIs for user to script tests and talks to server.