SCSI TOOLBOX, LLC

VCPSSL Tape Threaded Functions - Basic

Contents

The APIs that every VCPSSL Tape Application Must Have

1. ***BOOL VCSCSIPrepareForNewTapeTestSequence();***
2. ***BOOL VCSCSIAddTapeDeviceToBeTested(int nHA,int nTid,int nLun);***
3. ***"APIs to add tests to your test sequence (for example VCSCSIAddTapeWriteTest_Time)"***
4. ***BOOL VCSCSIStartTapeTestSequence();***
5. ***"APIs to retrieve status of the tests in your test sequence (for example VCSCSIGetTapeTestStatus)"***

The above outline contains the five basic parts of any VCPSSL tape application.

Tape API Function names and usage

API: VCSCSIPrepareForNewTapeTestSequence

```
BOOL VCSCSIPrepareForNewTapeTestSequence()
```

VCSCSIPrepareForNewTapeTestSequence is called to prepare the library to start threaded testing. Basically you call this function "to start everything from a fresh start"

***API*: VCSCSIAddTapeDeviceToBeTested**

```
BOOL VCSCSIAddTapeDeviceToBeTested(int nHA,int nTid,int nLun)
```

You call VCSCSIAddTapeDeviceToBeTested to tell the library which devices you want to test.

Return Values:

TRUE – The tape device identified by nHA, nTid, nLun was added to the array of devices to be tested

FALSE – The tape device identified by nHA, nTid, nLun was either not found, or the device was previously added so was NOT added a second time

***API*: VCSCSIStartTapeTestSequence**

```
BOOL VCSCSIStartTapeTestSequence()
```

After you have added all the devices to be tested via API VCSCSIAddTapeDeviceToBeTested, and after you've added all the tests (these APIs are covered below), you would then tell the library to begin testing by calling API VCSCSIStartTapeTestSequence.

THE AVAILABLE TESTS

***API*: VCSCSIAddTapeWriteTest_Time**

```
BOOL VCSCSIAddTapeWriteTest_Time(
                        int nNumberOfMinutes = 1,
                        enum ePATTERN_TYPE ePatType =  eIncrementing,
                        char * pstrPatternFile = NULL)
```

This adds a Write Test to your test sequence. It will start writing to the tape to wherever the tape happens to be – so if it is important for the writing to take place at the beginning of the tape, you must rewind the tape by calling VCSCSIAddTapeRewindTest.

nNumberOfMinutes:

Pass in how many minutes you want the library to write to the tape in this parameter

ePatType:

The available patterns are as follows:

- eAllZeroes (0x00)
- eAllOnes (0xFF)
- eAltZeroAndOne (0x55)
- eAltOneAndZero (0xAA)
- eLBA
- eRandom
- eUserPat (pattern file must contain ascii characters)
- eUserPatBinary (pattern file can contain any type of data)
- eWalkingZeros (all bits are '1' except for one '0' which "walks to the right", the pattern generated is 0x7F 0xBF 0xDF 0xEF 0xF7 0xFB 0xFD 0xFE)

- eWalkingOnes (all bits are '0' except for one '1' which "walks to the left", the pattern generated is 0x01 0x02 0x04 0x08 0x10 0x20 0x40 0x80)
- eAlt1And0ThenAlt0And1 (0xA5)
- eAlt0And1ThenAlt1And0 (0x5A)
- e2To1Compressable

pstrPatternFile:

If you are using pattern type eUserPat or eUserPatBinary then you must pass in the fully-qualified path and filename that has the pattern

***API*: VCSCSIAddTapeWriteTest_MBytes**

```
BOOL VCSCSIAddTapeWriteTest_MBytes(
                        int nNumMBytesToWrite = 1,
                        enum ePATTERN_TYPE ePatType =  eIncrementing,
                        char * pstrPatternFile = NULL)
```

This adds a Write Test to your test sequence. It will start writing to the tape to wherever the tape happens to be – so if it is important for the writing to take place at the beginning of the tape, you must rewind the tape by calling VCSCSIAddTapeRewindTest.

nNumMBytesToWrite:

Pass in how many MegaBytes you want the library to write to the tape in this parameter. 1 MegaByte is equal to 1048576 bytes.

ePatType:

The available patterns are as follows:

- eAllZeroes (0x00)
- eAllOnes (0xFF)
- eAltZeroAndOne (0x55)
- eAltOneAndZero (0xAA)
- eLBA
- eRandom
- eUserPat (pattern file must contain ascii characters)
- eUserPatBinary (pattern file can contain any type of data)
- eWalkingZeros (all bits are '1' except for one '0' which "walks to the right", the pattern generated is 0x7F 0xBF 0xDF 0xEF 0xF7 0xFB 0xFD 0xFE)

- eWalkingOnes (all bits are '0' except for one '1' which "walks to the left", the pattern generated is 0x01 0x02 0x04 0x08 0x10 0x20 0x40 0x80)
- eAlt1And0ThenAlt0And1 (0xA5)
- eAlt0And1ThenAlt1And0 (0x5A)
- e2To1Compressable

pstrPatternFile:

If you are using pattern type eUserPat or eUserPatBinary then you must pass in the fully-qualified path and filename that has the pattern

***API*: VCSCSIAddTapeReadTest_Time**

```
BOOL VCSCSIAddTapeReadTest_Time(
                        int nNumberOfMinutes = 1,
                        enum ePATTERN_TYPE ePatType = eIncrementing,
                        char * pstrPatternFile = NULL,
                        BOOL bCompareData = FALSE)
```

This will add a Read Test to your test sequence. It will start reading from wherever the tape happens to be – so if you intended to start reading from the beginning of the tape you must add a Rewind Test by calling VCSCSIAddTapeRewindTest.

nNumberOfMinutes:

Pass in how many minutes you want the library to read from the tape in this parameter

ePatType:

The available patterns are as follows:

- eAllZeroes (0x00)
- eAllOnes (0xFF)
- eAltZeroAndOne (0x55)
- eAltOneAndZero (0xAA)
- eLBA
- eRandom
- eUserPat (pattern file must contain ascii characters)
- eUserPatBinary (pattern file can contain any type of data)
- eWalkingZeros (all bits are '1' except for one '0' which "walks to the right", the pattern generated is 0x7F 0xBF 0xDF 0xEF 0xF7 0xFB 0xFD 0xFE)
- eWalkingOnes (all bits are '0' except for one '1' which "walks to the left", the pattern generated is 0x01 0x02 0x04 0x08 0x10 0x20 0x40 0x80)
- eAlt1And0ThenAlt0And1 (0xA5)

- eAlt0And1ThenAlt1And0 (0x5A)
- e2To1Compressable

pstrPatternFile:

If you are using pattern type eUserPat or eUserPatBinary then you must pass in the fully-qualified path and filename that has the pattern

bCompareData:

Pass in TRUE for bCompareData if you want the library to compare the data Read from the drive with the data pattern specified in parameter ePatType. If you specify TRUE for bCompareData, after the data is read from the tape, the library will generate the expected pattern based on ePatType. Then a comparison of the "actual data" and the "expected data" is done. If the two buffers are not identical, the test fails with status eMiscompare. Once a data miscompare is found, no more reading from the tape is done!

***API*: VCSCSIAddTapeReadTest_MBytes**

```
BOOL VCSCSIAddTapeReadTest_MBytes(
                        int nNumMBytesToRead = 1,
                        enum ePATTERN_TYPE ePatType = eIncrementing,
                        char * pstrPatternFile = NULL,
                        BOOL bCompareData = FALSE)
```

This will add a Read Test to your test sequence. It will start reading from wherever the tape happens to be – so if you intended to start reading from the beginning of the tape you must add a Rewind Test by calling VCSCSIAddTapeRewindTest.

nNumMBytesToRead:

Pass in how many MegaBytes you want the library to read from the tape in this parameter. 1 MegaByte is equal to 1048576 bytes.

ePatType:

The available patterns are as follows:

- eAllZeroes (0x00)
- eAllOnes (0xFF)
- eAltZeroAndOne (0x55)
- eAltOneAndZero (0xAA)
- eLBA
- eRandom

- eUserPat (pattern file must contain ascii characters)
- eUserPatBinary (pattern file can contain any type of data)
- eWalkingZeros (all bits are '1' except for one '0' which "walks to the right", the pattern generated is 0x7F 0xBF 0xDF 0xEF 0xF7 0xFB 0xFD 0xFE)
- eWalkingOnes (all bits are '0' except for one '1' which "walks to the left", the pattern generated is 0x01 0x02 0x04 0x08 0x10 0x20 0x40 0x80)
- eAlt1And0ThenAlt0And1 (0xA5)
- eAlt0And1ThenAlt1And0 (0x5A)
- e2To1Compressable

pstrPatternFile:

If you are using pattern type eUserPat or eUserPatBinary then you must pass in the fully-qualified path and filename that has the pattern

bCompareData:

Pass in TRUE for bCompareData if you want the library to compare the data Read from the drive with the data pattern specified in parameter ePatType. If you specify TRUE for bCompareData, after the data is read from the tape, the library will generate the expected pattern based on ePatType. Then a comparison of the "actual data" and the "expected data" is done. If the two buffers are not identical, the test fails with status eMiscompare. Once a data miscompare is found, no more reading from the tape is done!

***API*: VCSCSIAddTapeWriteFMTest**

```
BOOL VCSCSIAddTapeWriteFMTest()
```

This API adds the "Write Filemark Test" to your test sequence. If you need to write multiple filemarks, you will need to add this test multiple times.

***API*: VCSCSIAddTapeReadFMTest**

```
BOOL VCSCSIAddTapeReadFMTest()
```

This API adds the "Read Filemark Test" to your test sequence. If you need to read multiple filemarks, you will need to add this test multiple times.

***API*: VCSCSIAddTapeRewindTest**

```
BOOL VCSCSIAddTapeRewindTest()
```

This API adds the “Tape Rewind Test” to your test sequence.

***API*: VCSCSIAddTapeSpaceTest**

```
BOOL VCSCSIAddTapeSpaceTest(
                    enum eSPACE_TYPE_TAPE eSpaceType = eForwardToEODTape)
```

This API adds the “Tape Space Test” to your test sequence. With this test, you can space forward or backwards or to the “end of data” marker

eSpaceType:

The valid “spacing types” are as follows:

- eForwardToBlockTape
- eBackwardToBlockTape
- eForwardToFilemarkTape
- eBackwardToFilemarkTape
- eForwardToEODTape

If you need to space, for example, 3 blocks forward, you need to add 3 of these tests. The VCSCSIAddTapeSpaceTest only spaces forward/backward 1 block/filemark!

***API*: VCSCSIAddTapeCompressionTest**

```
BOOL VCSCSIAddTapeCompressionTest(
              enum eCOMPRESSION_TYPE_TAPE eCompType = eCompressionONTape)
```

eComptype:

The values for the compression type are

- eCompressionONTape
- eCompressionOFFTape

With compression OFF, writing a 64K block of data to the drive results in 64K of data being written to the tape. With compression ON, the drive will compress the data and result in most likely a significantly less amount of data being written to the tape. If the data is exactly 2-to-1 compressable, then a 64K block of data gets compressed to 32K of tape being written to the tape.

***API*: VCSCSIAddTapeLogPageTest**

```
BOOL VCSCSIAddTapeLogPageTest(
                    enum eLOGPAGE_TYPE_TAPE eLogType = eLogPageClearTape)
```

eLogType:

The values for eLogtype are as follows:

- eLogPageClearTape
- eLogPageSaveTape

eLogPageClearTape clears all the log pages. eLogPageSaveTape causes the library to read all the log pages, and parse each log page for all the parameters and stores the information to the drives logfile. The format of the data logged is

"Log Page #, Log Page Parameter, Parameter Data"

***API*: VCSCSIAddTapeSynchronizeTest**

```
BOOL VCSCSIAddTapeSynchronizeTest()
```

This test causes the library to not start the next test in the sequence until ALL drives have completed all the tests prior to the synchronize test step. So for example, if the synchronize test step is test #7 in your test sequence, and you have 2 drives running the test sequence, and say Drive#2 finishes Step 6 but Drive#1 is still executing test#5, then Drive#2 will "wait" until Drive#1 completes test#5 and test#6, and then both drives will start test#8 at the exact same time. To say it another way, the "synchronize test" causes all drives to start the next test step at the exact same time, waiting for slower drives to finish their testing.

***API*: VCSCSIAddTapeExternalProgramTest**

```
BOOL VCSCSIAddTapeExternalProgramTest(char * pProgramName)
```

The above API adds the external program test to your test sequence. What this means is that you can have the library call an application that you have written to perform any tasks that you want. As an example, suppose you want to read a certain mode page and store that information into the logfile. Once you have your application written, you can call it as a test step.

Important: The library will pass your application the following information so that you will know which device your application is being called by. The information passed in the command-line parameters is

"ha=nn,tid=mm,lun=xx". An example would be "ha=4,tid=5,lun=0". Notice there are no spaces in the information we pass on the command line.

pProgramName:

This parameter should contain the fully qualified path and application name. The path should have no spaces in it. For example, avoid names like
"c:\Program Files\MyApp\MyApplication.exe". Rather, store the application in a folder like
"c:\Workdir\MyApp\MyApplication.exe" (notice there are no spaces)

***API*: VCSCSIGetTapeTestStatus**

```
eTEST_STATUS VCSCSIGetTapeTestStatus(int nHA,int nTid,int nLun,
                                     int nTestNumber)
```

nTestNumber:

The nTestNumber parameter is 0-based. This means that the first test you add to your sequence has value 0 for nTestNumber.

Return Value:

The possible return values are:

0 (or eTestInProgress)
1 (or eCompleteOnSuccess)
2 (or eCompleteOnFailure)
3 (or eTestNotStartedYet)
4 (or eTestIsPaused)
5 (or eTestStopped)
6 (or eErrorOnParamsPassed)
7 (or eMisCompare)
8 (or eInvalidBlock)
9 (or ePendingIOOutstanding)
10 (or eUnknownStatus)

These values need some more discussion.

eTestInProgress: This means that when you made the call to VCSCSIGetTapeTestStatus the library was currently executing that test on your drive.

eCompleteOnSuccess: The test has completed and it completed successfully

eCompleteOnFailure: The test has completed and it failed. You can get more information on what caused the test to fail by calling VCSCSIGetTapeTestStatusWData to get any sense information or you

can call VCSCSIGetTapeTestStatusWStruct. You can also look in the logfile that is created (the filename has format "HA-Target-Lun-SerialNumber.Log", for example 04-014-000-2A0007F.Log).

eTestNotStartedYet: This means that the test has not been started yet. For example, if you have 5 tests in your test sequence, and test #2 is currently executing and you ask for the status of test #4, the library will return eTestNotStartedYet.

eTestIsPaused: This means you called VCSCSIPauseTapeTest to pause the test and it is currently in the "pause state". You must call VCSCSIResumeTapeTest to resume the test

eTestStopped: This means you called VCSCSIStopTapeTest to stop the test. Once you stop a test, there is no way to resume this test

eErrorOnParamsPassed: This means one of the parameters you passed in for the test was invalid and the test will not execute. An example of how you can get eErrorOnParamsPassed is the number of MegaBytes to write to a drive – it must be positive. But if you pass in say -5 then you will get eErrorOnParamsPassed.

eMisCompare: This means that data retrieved from the drive is not what the library was expecting. The logfile will have more information about which block has the incorrect information, and what was the expected and actual data.

***API*: VCSCSIGetTapeTestStatusWData**

```
eTEST_STATUS VCSCSIGetTapeTestStatusWData(
                                    int nHA,int nTid,int nLun,
                                    int nTestNumber,
                                    BYTE * pSenseBuf,int nSenseBufLen)
```

nSenseBufLen:

The nSenseBufLen parameter is the total size of your buffer pSenseBuf. It should have 128 BYTEs of space. BUT if you only want say 18 bytes of sense data, pass in nSenseBufLen equal to 18 and the library will only store at most 18 bytes in your buffer.

***API*: VCSCSIGetTapeTestStatusFull**

```
eTEST_STATUS VCSCSIGetTapeTestStatusFull(
                                int nHA,int nTid,int nLun,
                                int nTestNumber,
```

```
                                BYTE * pCDBBuf,int nCDBBufLen,
                                BYTE * pSenseBuf,int nSenseBufLen,
                                int * pRunningTime,int * pTimeRemaining)
```

pCDBBuf:

This is a pointer to a buffer where the library can store the CDB that caused the test to fail (if the test completed in failure). This parameter can be NULL.

nCDBBufLen:

This is the size of your buffer pCDBBuf. It should be 6 or 10

pSenseBuf:

This is a pointer to a buffer where the library can store the sense data (if any). This parameter can be NULL.

nSenseBufLen:

The nSenseBufLen parameter is the total size of your buffer pSenseBuf. It should have 128 BYTEs of space. BUT if you only want say 18 bytes of sense data, pass in nSenseBufLen equal to 18 and the library will only store at most 18 bytes in your buffer.

pRunningTime:

This is a pointer to an integer that will be filled in by the library with the time the test has been running (in seconds). This parameter can be NULL.

pTimeRemaining:

This is a pointer to an integer that will be filled in by the library with how much longer the time will be running (in seconds). This parameter can be NULL.

***API*: VCSCSIPauseTapeTest**

```
BOOL VCSCSIPauseTapeTest(int nHA,int nTid,int nLun,int nTestNumber)
```

Return Value:

Returns TRUE if the test was successfully paused

Returns FALSE if the test could not be paused (either because the test has not started, or the test has completed, or there is no nTestNumber test)

***API*: VCSCSIResumeTapeTest**

```
BOOL VCSCSIResumeTapeTest(int nHA,int nTid,int nLun,int nTestNumber)
```

Return Value:

Returns TRUE if the test was successfully resumed

Returns FALSE if the test could not be resumed (either because the test has not started, or the test has completed, or the test was not in a pause state, or there is no nTestNumber test)

***API*: VCSCSIStopTapeTest**

```
BOOL VCSCSIStopTapeTest(int nHA,int nTid,int nLun,int nTestNumber)
```

Return Value:

Returns TRUE if the test was successfully stopped

Returns FALSE if the test could not be stopped (either because the test has not started, or the test has completed, or there is no nTestNumber test)

***API*: VCSCSIStopAllTapeTest**

```
BOOL VCSCSIStopAllTapeTest()
```

This API stops the current test that is executing, and also prevents all other tests from running. It is different from VCSCSIStopTapeTest in that it stops all testing on all drives (whereas VCSCSIStopTapeTest stops only a single test on a single drive).