SCSI TOOLBOX, LLC

VCPSSL Threaded Functions - Getting Started

CREATING A VCPSSL DTB PROJECT 4

Step 1: 4

Step 2: 4

Step 3: 4

Step 4: 4

The APIs that every VCPSSL application must have 5

API: VCSCSIPrepareForNewDiskTestSequence 5

API: VCSCSIAddDiskDeviceToBeTested 5

API: VCSCSIStartDiskTestSequence 5

Available Tests 6

API: VCSCSIAddDiskWriteTest_Time 6

API: VCSCSIAddDiskWriteTest_Blocks 7

API: VCSCSIAddDiskReadTest_Time 8

API: VCSCSIAddDiskReadTest_Blocks 8

API: VCSCSIAddDiskWriteReadTest_Time 9

API: VCSCSIAddDiskWriteReadTest_Blocks 9

API: VCSCSIAddDiskVerifyTest_Time 9

API: VCSCSIAddDiskVerifyTest_Blocks 10

API: VCSCSIAddDiskWriteVerifyTest_Time 10

API: VCSCSIAddDiskWriteVerifyTest_Blocks 10

API: VCSCSIAddDiskSeekTest_Time 11

API: VCSCSIAddDiskSeekTest_Blocks 11

API: VCSCSIAddDiskFormatTest_SameSize 11

API: VCSCSIAddDiskFormatTest_NewSize 11

API: VCSCSIAddDiskFWDownloadTest 11

API: VCSCSIAddDiskSpinTest 12

API: VCSCSIAddDiskSynchronizeTest 12

API: VCSCSIAddDiskStreamingTest_Time 12

API: VCSCSIAddDiskStreamingTest_Blocks 13

API: VCSCSIAddDiskOLTPTest_Time 13

API: VCSCSIAddDiskOLTPTest_Blocks 14

API: VCSCSIAddDiskFileServerTest_Time 15

API: VCSCSIAddDiskFileServerTest_Blocks 15

API: VCSCSIAddDiskWebServerTest_Time 16

API: VCSCSIAddDiskWebServerTest_Blocks 16

API: VCSCSIAddDiskWorkStationTest_Time 17

API: VCSCSIAddDiskWorkStationTest_Blocks 18

API: VCSCSIAddDiskClearLogPagesTest 18

API: VCSCSIAddDiskSaveLogPagesTest 19

API: VCSCSIAddDiskDSTTest 19

API: VCSCSIAddDiskDelayTest 19

API: VCSCSIAddDiskAdjTrkIntfTest 19

API: VCSCSIAddDiskSMARTTest 20

API: VCSCSIAddDiskSATAFWDLTest 21

Getting the Status of a Test 21

API: VCSCSIGetDiskTestStatus 21

API: VCSCSIGetDiskTestStatusWData 22

API: VCSCSIGetDiskTestStatusWStruct 23

Additional APIs: Pausing, Stopping, Resuming 23

API: VCSCSIPauseDiskTest 23

API: VCSCSIResumeDiskTest 24

API: VCSCSIStopDiskTest 24

API: VCSCSIStopAllDiskTest 24

EXAMPLE CODE 25

CREATING A VCPSSL DTB PROJECT

Step 1: Copy the file VCPSSLImports.h into the project folder (this file will be in c:\Program Files\STB\DTB\VisualC++)

Step 2: In your StdAfx.h file, add the line

#include "VCPSSLImports.h

Step 3: Insure that STBVCBase.dll and VCPSSL.dll are in the System32 folder

Step 4:

- Copy the file VCPSSL.lib into your project folder (the file VCPSSL.lib will be in c:\Program Files\STB\DTB\VisualC++)
- Select the Visual Studio Main Menu ‘Project’ choice
- Select the ‘Settings’ choice
- Highlight your project name in the left pane, then click the ‘Link’ tab
- Choose the ‘General’ selection from the ‘Category’ combo selection box
- In the ‘Object/Library Modules’ edit box type “VCPSSL.LIB”

IMPORTANT: When debugging, you'll need to tell VC++ the relative path to VCPSSL.lib by going to the Link tab, insure the "Category" combobox has "Input" selected, and then in the "Additional library path" editbox, enter the relative path. In most cases, you should enter "..\" without the quotation marks.

The APIs that every VCPSSL application must have

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

1. BOOL VCSCSIPrepareForNewDiskTestSequence();
2. BOOL VCSCSIAddDiskDeviceToBeTested(int nHA,int nTid,int nLun);
3. "APIs to add tests to your test sequence (for example VCSCSIAddDiskWriteTest_Time)"
4. BOOL VCSCSIStartDiskTestSequence();
5. "APIs to retrieve status of the tests in your test sequence (for example VCSCSIGetDiskTestStatus)"

API: **VCSCSIPrepareForNewDiskTestSequence**

```
BOOL VCSCSIPrepareForNewDiskTestSequence()
```

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

API: **VCSCSIAddDiskDeviceToBeTested**

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

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

Return Values:

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

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

API: **VCSCSIStartDiskTestSequence**

```
BOOL VCSCSIStartDiskTestSequence()
```

After you have added all the devices to be tested via API VCSCSIAddDiskDeviceToBeTested, 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 VCSCSIStartDiskTestSequence.

Available Tests

API: VCSCSIAddDiskWriteTest_Time

```
BOOL VCSCSIAddDiskWriteTest_Time(
                                   int nNumberOfMinutes = 1,
                                   int nRandomOrSeq = 1,
                                   int nStartLBA = 0,
                                   ePATTERN_TYPE ePatType = eIncrementing,
                                   char * pstrPatternFile = NULL,
                                   BOOL bOverlayLBA = FALSE,
                                   SPECIAL_OPTIONS * pSO = NULL)
```

Discussion: First note all the parameters have default values. The possible values for nRandomOrSeq are as follows:

nNumberOfMinutes:

Pass in the number of minutes you want to test to run for. If you want to run the test by seconds, you must enter the number of seconds in the high 16 bits of the integer!

Example: 0x00000003 runs the test for 3 minutes

Example: 0x00070003 runs the test for 3 minutes and 7 seconds

Example: 0x00070000 runs the test for 7 seconds

nRandomOrSeq:

- 0 for Random Access
- 1 for Sequential Access
- 2 for Butterfly Access
- 3 for CPAM Access

The available patterns are as follows:

ePatType:

- 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 (for tape drives)

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

bOverlayLBA:

If you pass in TRUE for bOverlayLBA then for each block of data the LBA will be overlayed on the first 4 bytes (if the Capacity of the drive is less than 2TeraBytes) and on the first 8 bytes (if the Capacity of the drive is more than 2TeraBytes)

pSO:

The SPECIAL_OPTIONS data structure has the following fields:

int nVersionNumber;
int nSizeOfThisStruct;
__int64 i64HighBlockForThisTest;
BOOL   bOverlayLBAOnEndOfBlock; //Can overlay the LBA on the last part of each block
BOOL   bDataCompareOnOnlyOverlays; //On data compare you can do the compare only on the overlays

BOOL   bOverlayKey; //If set to TRUE, we overlay the “key” after the LBA overlay
int   nOverlayKey; //This contains the “key” you want to overlay

API: **VCSCSIAddDiskWriteTest_Blocks**

```
BOOL VCSCSIAddDiskWriteTest_Blocks(
                             long lNumBlocksToWrite = -1,
                             int nRandomOrSeq = 1,
                             int nStartLBA = 0,
                             ePATTERN_TYPE ePatType = eIncrementing,
                             char * pstrPatternFile = NULL,
                             BOOL bOverlayLBA = FALSE)
```

We've already discussed most of the parameters when going over the API VCSCSIAddDiskWriteTest_Time, but the parameter lNumBlockstoWrite needs special mention.

lNumBlockstoWrite:

Pass in the number of blocks you want to test to run for. The special number "-1" means to run the test for the number of blocks on the entire drive

API: VCSCSIAddDiskReadTest_Time

```
BOOL VCSCSIAddDiskReadTest_Time(
                     int nNumberOfMinutes = 1,
                     int nRandomOrSeq = 1,
                     int nStartLBA = 0,
                     ePATTERN_TYPE ePatType = eIncrementing,
                     char * pstrPatternFile = NULL,
                     BOOL bOverlayLBA = FALSE,
                     BOOL bCompareData = FALSE,
                     SPECIAL_OPTIONS * pSO = NULL)
```

bCompareData:

Pass in TRUE for bCompareData to cause the library to do a data compare on the data read from the drive. The question asked many times is: what data are you comparing it to? The answer is to the ePatType you pass in and bOverlayLBA you pass in. From these 2 parameters, the library generates the expected pattern. It then compares this expected pattern to the actual data retrieved from the drive. If there is a mismatch, the test fails with a "miscompare".

API: VCSCSIAddDiskReadTest_Blocks

```
BOOL VCSCSIAddDiskReadTest_Blocks(
                             long lNumBlocksToWrite = -1,
                             int nRandomOrSeq = 1,
                             int nStartLBA = 0,
                             ePATTERN_TYPE ePatType = eIncrementing,
                             char * pstrPatternFile = NULL,
                             BOOL bOverlayLBA = FALSE,
                             BOOL bCompareData = FALSE,
                             SPECIAL_OPTIONS * pSO = NULL)
```

API: **VCSCSIAddDiskWriteReadTest_Time**

```
BOOL VCSCSIAddDiskWriteReadTest_Time(
                             int nNumberOfMinutes = 1,
                             int nRandomOrSeq = 1,
                             int nStartLBA = 0,
                             ePATTERN_TYPE ePatType = eIncrementing,
                             char * pstrPatternFile = NULL,
                             BOOL bOverlayLBA = FALSE,
                             BOOL bCompareData = FALSE,
                             SPECIAL_OPTIONS * pSO = NULL)
```

ePatType:

On the Write command, this determines the pattern that the library writes to the drive. On the Read command, it is used to created the "expected pattern" (and if you set bCompareData to TRUE the expected pattern will be compared to the actual data retrieved from the drive)

API: **VCSCSIAddDiskWriteReadTest_Blocks**

```
BOOL VCSCSIAddDiskWriteReadTest_Blocks(
                            long lNumBlocksToWrite = -1,
                            int nRandomOrSeq = 1,
                            int nStartLBA = 0,
                            ePATTERN_TYPE ePatType = eIncrementing,
                            char * pstrPatternFile = NULL,
                            BOOL bOverlayLBA = FALSE,
                            BOOL bCompareData = FALSE,
                            SPECIAL_OPTIONS * pSO = NULL)
```

lNumBlocksToWrite:

This parameter indicates how many blocks to write. Since every write is accompanied by a Read, the actual number of blocks transferred to and from the drive will be twice the number you pass in for lNumBlockstoWrite

API: **VCSCSIAddDiskVerifyTest_Time**

```
BOOL VCSCSIAddDiskVerifyTest_Time(
                                 int nNumberOfMinutes = 1,
                                 int nRandomOrSeq = 1,
                                 int nStartLBA = 0)
```

The above test issues the Verify scsi command (opcode 0x2F). Notice there is no transfer of data – this means that the verify commands that the library sends out has the ByteChk bit set to 0. As such, the verification criteria is determined by your mode page settings.

API: **VCSCSIAddDiskVerifyTest_Blocks**

```
BOOL VCSCSIAddDiskVerifyTest_Blocks(
                                    long lNumBlocksToVerify = -1,
                                    int nRandomOrSeq = 1,
                                    int nStartLBA = 0)
```

The above test issues the Verify scsi command (opcode 0x2F). Notice there is no transfer of data – this means that the verify commands that the library sends out has the ByteChk bit set to 0. As such, the verification criteria is determined by your mode page settings.

API: **VCSCSIAddDiskWriteVerifyTest_Time**

```
BOOL VCSCSIAddDiskWriteVerifyTest_Time(
                            int nNumberOfMinutes = 1,
                            int nRandomOrSeq = 1,
                            int nStartLBA = 0,
                            ePATTERN_TYPE ePatType = eIncrementing,
                            char * pstrPatternFile = NULL,
                            BOOL bOverlayLBA = FALSE,
                            BOOL bByteCheck = FALSE)
```

bByteCheck:

Pass in TRUE for bByteCheck if you want the library to send out the “Write Verify” command with the “Byte Check” bit set in the CDB that goes to the drive. Passing in bByteCheck as TRUE results in the data being written to the drive, then the drive reading the data written to the drive and a byte-by-byte comparision being done to insure the data is correct.

API: **VCSCSIAddDiskWriteVerifyTest_Blocks**

```
BOOL VCSCSIAddDiskWriteVerifyTest_Blocks(
                            long lNumBlocksToVerify = -1,
                            int nRandomOrSeq = 1,
                            int nStartLBA = 0,
                            ePATTERN_TYPE ePatType = eIncrementing,
                            char * pstrPatternFile = NULL,
                            BOOL bOverlayLBA = FALSE,
                            BOOL bByteCheck = FALSE)
```

bByteCheck:

Pass in TRUE for bByteCheck if you want the library to send out the “Write Verify” command with the “Byte Check” bit set in the CDB that goes to the drive. Passing in bByteCheck as TRUE results in the data being written to the drive, then the drive reading the data written to the drive and a byte-by-byte comparision being done to insure the data is correct.

API: VCSCSIAddDiskSeekTest_Time

```
BOOL VCSCSIAddDiskSeekTest_Time(
                                  int nNumberOfMinutes = 1,
                                  int nRandomOrSeq = 1,
                                  int nStartLBA = 0)
```

API: VCSCSIAddDiskSeekTest_Blocks

```
BOOL VCSCSIAddDiskSeekTest_Blocks(
                                  long lNumBlocksToWrite = -1,
                                  int nRandomOrSeq = 1,
                                  int nStartLBA = 0)
```

API: VCSCSIAddDiskFormatTest_SameSize

```
BOOL VCSCSIAddDiskFormatTest_SameSize()
```

As the name of the above API implies, this test formats the drive to the exact same blocksize as it is currently (so, for example, if the drive's current blocksize is 520 bytes then it re-formats the drive to have blocksize 520).

API: VCSCSIAddDiskFormatTest_NewSize

```
BOOL VCSCSIAddDiskFormatTest_NewSize(int nNewBlockSize)
```

nNewBlockSize:

Pass in the new blocksize for the drive in the above parameter. For example, if the drive is currently at blocksize 512, and you want to re-format the drive to have blocksize 520, then you pass in 520 for nNewBlockSize

API: VCSCSIAddDiskFWDownloadTest

```
BOOL VCSCSIAddDiskFWDownload(char * pFirmwareFileName)
```

pFirmwareFileName:

Make sure you pass in the fully qualified path and filename for the firmware file

API: **VCSCSIAddDiskSpinTest**

```
BOOL VCSCSIAddDiskSpinTest(int nNumberOfMinutes = 1)
```

The "Spin" test powers down the drive, and then powers the drive back up over-and-over for nNumberOfMinutes. The final state of the drive will be that the drive is powered up (so if the number of minutes has elapsed and the test is in the middle of powering down the drive, it will complete the test by powering up the drive).

API: **VCSCSIAddDiskSynchronizeTest**

```
BOOL VCSCSIAddDiskSynchronizeTest()
```

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: **VCSCSIAddDiskStreamingTest_Time**

```
BOOL VCSCSIAddDiskStreamingTest_Time(
                                       int nNumberOfMinutes,
                                       int nStartLBA,
                                       enum ePATTERN_TYPE ePatType,
                                       char * pstrPatternFile,
                                       BOOL bOverlayLBA,
                                       int nPercentWriteVersusRead,
                                       int nMaxTransLen,
                                       BOOL bCompareData,
                                       SPECIAL_OPTIONS * pSO = NULL)
```

The "Streaming Test" is a test which combines Writes and Reads at a particular "distribution". With the "Streaming Test" you can have say 78% Writes and 22% Read. You have complete freedom to choose your distribution by modifying the parameter nPercentWriteVersusRead. In terms of accessing the drive, the access is sequential. So, for example, you may Write to LBA 0 with a transfer length of 0x4C, then Write to LBA 0x4C with a transfer length of 0x71, then Read from LBA 0xBD with a transfer length of 0x5A. The accesses are NOT random – it is sequential with Writes and Reads being done according to the ratio you specify.

nPercentWriteVersusRead:

The parameter nPercentWriteVersusRead determines the distribution of Writes and Reads. If you pass in the integer 78, then this will cause the library to issue 78% Writes with 22% Reads. If say you want only 10% of the IOs to be Writes, then you would pass in 10 for nPercentWriteVersusRead (and so you would necessarily get 90% Reads).

nMaxTransLen:

The default behavior of the Streaming Test is to transfer a random amount between 64 and 128 blocks. But if you want to have the test do a random amount between 64 and say 100, then pass in 100 for nMaxTransLen. Any value less than 64 causes every IO to be exactly nMaxTransLen.

API: VCSCSIAddDiskStreamingTest_Blocks

```
BOOL VCSCSIAddDiskStreamingTest_Blocks(
                                       long lNumberOfBlocks,
                                       int nStartLBA,
                                       enum ePATTERN_TYPE ePatType,
                                       char * pstrPatternFile,
                                       BOOL bOverlayLBA,
                                       int nPercentWriteVersusRead,
                                       int nMaxTransLen,
                                       BOOL bCompareData,
                                       SPECIAL_OPTIONS * pSO = NULL)
```

The “Streaming Test” is a test which combines Writes and Reads at a particular “distribution”. With the “Streaming Test” you can have say 78% Writes and 22% Read. You have complete freedom to choose your distribution by modifying the parameter nPercentWriteVersusRead. In terms of accessing the drive, the access is sequential. So, for example, you may Write to LBA 0 with a transfer length of 0x4C, then Write to LBA 0x4C with a transfer length of 0x71, then Read from LBA 0xBD with a transfer length of 0x5A. The accesses are NOT random – it is sequential with Writes and Reads being done according to the ratio you specify.

nPercentWriteVersusRead:

The parameter nPercentWriteVersusRead determines the distribution of Writes and Reads. If you pass in the integer 78, then this will cause the library to issue 78% Writes with 22% Reads. If say you want only 10% of the IOs to be Writes, then you would pass in 10 for nPercentWriteVersusRead (and so you would necessarily get 90% Reads).

nMaxTransLen:

The default behavior of the Streaming Test is to transfer a random amount between 64 and 128 blocks. But if you want to have the test do a random amount between 64 and say 100, then pass in 100 for nMaxTransLen. Any value less than 64 causes every IO to be exactly nMaxTransLen.

API: VCSCSIAddDiskOLTPTest_Time

```
BOOL VCSCSIAddDiskOLTPTest_Time(
                                 int nNumberOfMinutes,
                                 int nStartLBA,
                                 enum ePATTERN_TYPE ePatType,
                                 char * pstrPatternFile,
                                 BOOL bOverlayLBA,
                                 int nPercentWriteVersusRead,
                                 int nMaxTransLen,
                                 BOOL bCompareData,
                                 SPECIAL_OPTIONS * pSO = NULL)
```

The "Online Transaction Processing Test" is designed to randomly access the drive, with mixing Writes and Reads at a particular ratio. The transfer lengths are randomly chosen from between 4 and 32 blocks.

nPercentWriteVersusRead:

The parameter nPercentWriteVersusRead determines the distribution of Writes and Reads. If you pass in the integer 78, then this will cause the library to issue 78% Writes with 22% Reads. If say you want only 10% of the IOs to be Writes, then you would pass in 10 for nPercentWriteVersusRead (and so you would necessarily get 90% Reads).

nMaxTransLen:

The default behavior of the "Online Transaction Processing Test" is to transfer a random amount between 4 and 32 blocks. But if you want to have the test do a random amount between 4 and say 100, then pass in 100 for nMaxTransLen. Any value less than 4 causes every IO to be exactly nMaxTransLen.

***API*: VCSCSIAddDiskOLTPTest_Blocks**

```
BOOL VCSCSIAddDiskOLTPTest_Blocks(
                                 long lNumberOfBlocks,
                                 int nStartLBA,
                                 enum ePATTERN_TYPE ePatType,
                                 char * pstrPatternFile,
                                 BOOL bOverlayLBA,
                                 int nPercentWriteVersusRead,
                                 int nMaxTransLen,
                                 BOOL bCompareData,
                                 SPECIAL_OPTIONS * pSO = NULL)
```

nPercentWriteVersusRead:

The parameter nPercentWriteVersusRead determines the distribution of Writes and Reads. If you pass in the integer 78, then this will cause the library to issue 78% Writes with 22% Reads. If say you want only 10% of the IOs to be Writes, then you would pass in 10 for nPercentWriteVersusRead (and so you would necessarily get 90% Reads).

nMaxTransLen:

The default behavior of the “Online Transaction Processing Test” is to transfer a random amount between 4 and 32 blocks. But if you want to have the test do a random amount between 4 and say 100, then pass in 100 for nMaxTransLen. Any value less than 4 causes every IO to be exactly nMaxTransLen.

API: **VCSCSIAddDiskFileServerTest_Time**

```
BOOL VCSCSIAddDiskFileServerTest_Time(
                                       int nNumberOfMinutes,
                                       int nStartLBA,
                                       enum ePATTERN_TYPE ePatType,
                                       char * pstrPatternFile,
                                       BOOL bOverlayLBA,
                                       int nPercentWriteVersusRead,
                                       int nMaxTransLen,
                                       BOOL bCompareData,
                                       SPECIAL_OPTIONS * pSO = NULL)
```

The “File Server Test” is designed to randomly access the drive with a mixture of Writes and Reads. The transfer lengths are randomly chosen from between 8 and 128 blocks.

nPercentWriteVersusRead:

The parameter nPercentWriteVersusRead determines the distribution of Writes and Reads. If you pass in the integer 78, then this will cause the library to issue 78% Writes with 22% Reads. If say you want only 10% of the IOs to be Writes, then you would pass in 10 for nPercentWriteVersusRead (and so you would necessarily get 90% Reads).

nMaxTransLen:

The default behavior of the “File Server Test” is to transfer a random amount between 8 and 128 blocks. But if you want to have the test do a random amount between 8 and say 100, then pass in 100 for nMaxTransLen. Any value less than 8 causes every IO to be exactly nMaxTransLen.

API: **VCSCSIAddDiskFileServerTest_Blocks**

```
BOOL VCSCSIAddDiskFileServerTest_Blocks(
                                       long lNumberOfBlocks,
                                       int nStartLBA,
                                       enum ePATTERN_TYPE ePatType,
                                       char * pstrPatternFile,
                                       BOOL bOverlayLBA,
                                       int nPercentWriteVersusRead,
                                       int nMaxTransLen,
                                       BOOL bCompareData,
                                    SPECIAL_OPTIONS * pSO = NULL)
```

The “File Server Test” is designed to randomly access the drive with a mixture of Writes and Reads. The transfer lengths are randomly chosen from between 8 and 128 blocks.

nPercentWriteVersusRead:

The parameter nPercentWriteVersusRead determines the distribution of Writes and Reads. If you pass in the integer 78, then this will cause the library to issue 78% Writes with 22% Reads. If say you want only 10% of the IOs to be Writes, then you would pass in 10 for nPercentWriteVersusRead (and so you would necessarily get 90% Reads).

nMaxTransLen:

The default behavior of the “File Server Test” is to transfer a random amount between 8 and 128 blocks. But if you want to have the test do a random amount between 8 and say 100, then pass in 100 for nMaxTransLen. Any value less than 8 causes every IO to be exactly nMaxTransLen.

API: **VCSCSIAddDiskWebServerTest_Time**

```
BOOL VCSCSIAddDiskWebServerTest_Time(int nNumberOfMinutes,
                                     int nStartLBA,
                                     enum ePATTERN_TYPE ePatType,
                                     char * pstrPatternFile,
                                     BOOL bOverlayLBA,
                                     int nMaxTransLen,
                                     BOOL bCompareData,
                                     SPECIAL_OPTIONS * pSO = NULL)
```

The “Web Server Test” is designed to randomly access the drive with Reads. The transfer lengths are randomly chosen from between 1 and 1024 blocks.

nMaxTransLen:

The default behavior of the “Web Server Test” is to transfer a random amount between 1 and 1024 blocks. But if you want to have the test do a random amount between 1 and say 100, then pass in 100 for nMaxTransLen.

API: **VCSCSIAddDiskWebServerTest_Blocks**

```
BOOL VCSCSIAddDiskWebServerTest_Blocks(long lNumberOfBlocks,
                                       int nStartLBA,
                                       enum ePATTERN_TYPE ePatType,
                                       char * pstrPatternFile,
                                       BOOL bOverlayLBA,
                                                  int nMaxTransLen,
                                                  BOOL bCompareData,
                                                  SPECIAL_OPTIONS * pSO = NULL)
```

The “Web Server Test” is designed to randomly access the drive with Reads. The transfer lengths are randomly chosen from between 1 and 1024 blocks.

nMaxTransLen:

The default behavior of the “Web Server Test” is to transfer a random amount between 1 and 1024 blocks. But if you want to have the test do a random amount between 1 and say 100, then pass in 100 for nMaxTransLen.

API: **VCSCSIAddDiskWorkStationTest_Time**

```
BOOL VCSCSIAddDiskWorkStationTest_Time(int nNumberOfMinutes,
                                       int nStartLBA,
                                       enum ePATTERN_TYPE ePatType,
                                       char * pstrPatternFile,
                                       BOOL bOverlayLBA,
                                       int nPercentWriteVersusRead,
                                       int nMaxTransLen,
                                       BOOL bCompareData,
                                       SPECIAL_OPTIONS * pSO = NULL)
```

The “Work Station Test” is our most complex test in that it is designed to randomly access the drive 80% of the time, and then sequential 20% of the time. What this means is that over a long run of the “Work Station Test”, 80% of the time the LBA for the next IO will be chosen at random, but then 20% of the time the LBA for the next IO will be “next to” or “adjacent” to the previous IO. By way of an example, the LBAs might be 0x125AD, 0x0A437, 0xAFD007, 0xAFD017. Notice the first 3 IOs the LBAs are random, but then the 4th IO (with LBA 0xAFD017) is right next to the previous IO (which has LBA 0xAFD007). In addition to the mixture of the random and sequential access of the drive, the test also has a mixture of Writes and Reads (the distribution of Writes and Reads is determined by the parameter nPercentWriteVersusRead. The transfer lengths are randomly chosen from between 1 and 32 blocks.

nPercentWriteVersusRead:

The parameter nPercentWriteVersusRead determines the distribution of Writes and Reads. If you pass in the integer 78, then this will cause the library to issue 78% Writes with 22% Reads. If say you want only 10% of the IOs to be Writes, then you would pass in 10 for nPercentWriteVersusRead (and so you would necessarily get 90% Reads).

nMaxTransLen:

The default behavior of the “Work Station Test” is to transfer a random amount between 1 and 32 blocks. But if you want to have the test do a random amount between 1 and say 10, then pass in 10 for nMaxTransLen.

API: **`VCSCSIAddDiskWorkStationTest_Blocks`**

```
BOOL VCSCSIAddDiskWorkStationTest_Blocks(long lNumberOfBlocks,
                                         int nStartLBA,
                                         enum ePATTERN_TYPE ePatType,
                                         char * pstrPatternFile,
                                         BOOL bOverlayLBA,
                                         int nPercentWriteVersusRead,
                                         int nMaxTransLen,
                                         BOOL bCompareData,
                                         SPECIAL_OPTIONS * pSO = NULL)
```

The “Work Station Test” is our most complex test in that it is designed to randomly access the drive 80% of the time, and then sequential 20% of the time. What this means is that over a long run of the “Work Station Test”, 80% of the time the LBA for the next IO will be chosen at random, but then 20% of the time the LBA for the next IO will be “next to” or “adjacent” to the previous IO. By way of an example, the LBAs might be 0x125AD, 0x0A437, 0xAFD007, 0xAFD017. Notice the first 3 IOs the LBAs are random, but then the 4th IO (with LBA 0xAFD017) is right next to the previous IO (which has LBA 0xAFD007). In addition to the mixture of the random and sequential access of the drive, the test also has a mixture of Writes and Reads (the distribution of Writes and Reads is determined by the parameter nPercentWriteVersusRead. The transfer lengths are randomly chosen from between 1 and 32 blocks.

nPercentWriteVersusRead:

The parameter nPercentWriteVersusRead determines the distribution of Writes and Reads. If you pass in the integer 78, then this will cause the library to issue 78% Writes with 22% Reads. If say you want only 10% of the IOs to be Writes, then you would pass in 10 for nPercentWriteVersusRead (and so you would necessarily get 90% Reads).

nMaxTransLen:

The default behavior of the “Work Station Test” is to transfer a random amount between 1 and 32 blocks. But if you want to have the test do a random amount between 1 and say 10, then pass in 10 for nMaxTransLen.

API: **`VCSCSIAddDiskClearLogPagesTest`**

```
BOOL VCSCSIAddDiskClearLogPagesTest()
```

This test clears all the log pages maintained by the drive

API: **VCSCSIAddDiskSaveLogPagesTest**

BOOL VCSCSIAddDiskSaveLogPagesTest()

This test reads all the log pages maintained by the drive and stores this data into the drive's logfile (this logfile is created by the library and has the format "HA-Target-Lun-SerialNumber.Log"). The format of the data stored in the logfile will be "Log Page Number, Log Page Parameter, Data".

API: **VCSCSIAddDiskDSTTest**

BOOL VCSCSIAddDiskDSTTest()

This test runs the Short Drive Self Test. This is a self test done by the drive itself – the library does NOT determine what is performed in the Drive Self Test (the library simply initiates the Drive Self Test and waits for the test to complete).

API: **VCSCSIAddDiskDelayTest**

BOOL VCSCSIAddDiskDelayTest(int nNumSecondsToDelay)

This test simply "delays" the testing on a drive by the number of seconds you specify in nNumSecondsToDelay.

API: **VCSCSIAddDiskAdjTrkIntfTest**

```
BOOL VCSCSIAddDiskAdjTrkIntfTest(int nTargetCyl,int nTargetHead,
                                 int nNumWritesToMiddleTrk)
```

The "Adjacent Track Interference Test" is designed to test the effects on a track due to the vibrations on neighboring tracks. Adjacent tracks are considered to be adjacent tracks on the same platter (NOT adjacent tracks on the same cylinder). So given a track, which is uniquely characterized by its Cylinder:Head values, the track "to the left" is (Cylinder – 1):Head, while the track to the right is (Cylinder + 1):Head. Call these 3 tracks Tr_L, Tr_M, Tr_R. We write "All Zeroes" to all 3 tracks. Then we write a different pattern 30,000 times to Tr_M. After writing track Tr_M 30,000 times, we then look to track Tr_L, and Tr_R to see if the vibration on the middle track had any effect on the left and right tracks.

If there was no effect then the pattern on these two tracks would still be “All Zeroes”. If there is all zeroes, the test succeeds. If any bit is different, the test fails.

nTargetCyl:

This is the “Cylinder” value. Since the library needs to go to the track to the left, which means we must subtract 1 from the “Cylinder” value, the “Cylinder” value must be atleast 1.

nTargetHead:

This is which “platter” on the drive you want tested. As a general rule of thumb, since many drives only have around 12 platters, this value should be fairly small.

***API*: VCSCSIAddDiskSMARTTest**

BOOL VCSCSIAddDiskSMARTTest(DMM_AtaSmartData * pAtaSmartData)

This test allows you to compare the SMART attributes to values you store in the DMM_AtaSmartData data structure. There structure has the following fields:

```
int HBAType;
int PercentOfThreshold;
int Param1;
int Param2;
int Param3;
int Param4;
int Param5;
int RAW1;
int RAW2;
int RAW3;
int RAW4;
int RAW5;
bool ThresholdChecked;
bool ParamsChecked;
bool Param1Checked;
bool Param2Checked;
bool Param3Checked;
bool Param4Checked;
bool Param5Checked;
int  P1CompareType;
int  P2CompareType;
int  P3CompareType;
int  P4CompareType;
int  P5CompareType;
```

API**: VCSCSIAddDiskSATAFWDLTest**

BOOL VCSCSIAddDiskSATAFWDLTest(char * pFirmwareFileName)

This test allows you to download firmware to a SATA drive. As the normal firmware download does not work on SATA drives, you must use this function to do a firmware download to your SATA drive (that is do NOT use VCSCSIAddDiskFWDownloadTest).

API: VCSCSIAddDiskSATAinfoTest

BOOL VCSCSIAddDiskSATAinfoTest()

This test retrieves the SATA IDENTIFY information from the SATA drive and stores this information in the drive's logfile.

Getting the Status of a Test

API**: VCSCSIGetDiskTestStatus**

eTEST_STATUS VCSCSIGetDiskTestStatus(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 VCSCSIGetDiskTestStatus 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 VCSCSIGetDiskTestStatusWData to get any sense information or you can call VCSCSIGetDiskTestStatusWStruct. 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 VCSCSIPauseDiskTest to pause the test and it is currently in the "pause state". You must call VCSCSIResumeDiskTest to resume the test

eTestStopped: This means you called VCSCSIStopDiskTest 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 High Capacity of the drive is only 0x05000000 and you pass in a Start Block of 0x06000000.

eMisCompare: This means that data retrieved from the drive and 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*: VCSCSIGetDiskTestStatusWData**

eTEST_STATUS VCSCSIGetDiskTestStatusWData(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*: VCSCSIGetDiskTestStatusWStruct**

eTEST_STATUS VCSCSIGetDiskTestStatusStruct(int nHA,int nTid,int nLun,int nTestNumber,TEST_PROGRESS_INFO * pTestProgressInfo);

You call VCSCSIGetDiskTestStatusWStruct to get information about the test as it is progressing. The types of information you can retrieve from this API are, for example, "how much longer is the test going to take to finish", "how many IOs have been issued", "how much data has been transferred by the test".

pTestProgressInfo:

The fields in the TEST_PROGRESS_INFO structure are as follows:

```
int    nSizeOfThisStruct;
DWORD  dwTestStatus;
int    nNumBlocksTransSoFar;
double dNumBytesTransSoFar;
time_t testStartTime;
time_t testEndTime;
time_t testPauseTime;
int    nNumTotBlocksToTransfer;
int    nTimeRunning;
int    nTimeRemaining;
BYTE   cCDBOnFailure[16];
__int64 i64NumIOIssued;
BYTE   cMiscompareInfoExp[16];
BYTE   cMiscompareInfoAct[16];
int    nMiscompareInfoOff;
__int64 i64MiscompareLBA;
```

Notice that in addition to being able to get information about the test as it is proceeding, you also call this API to get error information. For example, if a miscompare occurred, you call this API to get the expected data (in field cMiscompareInfoExp), the actual data retrieved from the drive (this is field cMiscompareInfoAct), and where in the block of data does the miscompare start (this is stored in field nMiscompareInfoOff).

Additional APIs: Pausing, Stopping, Resuming

***API*: VCSCSIPauseDiskTest**

BOOL VCSCSIPauseDiskTest(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: `VCSCSIResumeDiskTest`

BOOL VCSCSIResumeDiskTest(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: `VCSCSIStopDiskTest`

BOOL VCSCSIStopDiskTest(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: `VCSCSIStopAllDiskTest`

BOOL VCSCSIStopAllDiskTest()

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

EXAMPLE CODE

The following example outlines the normal usage of the APIs contained in this document. You should use it as a guideline for developing for own application using the VCPSSL library. The general outline of any application using VCPSSL is as follows:

1. Initialize the library by calling VCSCSIPrepareForNewDiskTestSequence
2. Add the disk drives to the pool of drives that the tests will be run against by calling VCSCSIAddDiskDeviceToBeTested
3. Add all the tests to be performed by calling one of the “VCSCSIAddDisknnnnnnnTest”
4. Start the testing by calling VCSCSIStartDiskTestSequence
5. Determine the status of the tests by calling VCSCSIGetDiskTestStatus

```
int nHA,nTid,nLun;
eTEST_STATUS eTestStatus;

//STEP 1) Initialize the library
VCSCSIPrepareForNewDiskTestSequence();

//STEP 2) Add devices to be tested.  In this example we are just going
//to add 1 drive, but if you need to add say 10 drives then you need
//to have 10 calls to VCSCSIAddDiskDeviceToBeTested
nHA = 4;
nTid = 5;
nLun = 0;
bSuccess = VCSCSIAddDiskDeviceToBeTested(nHA,nTid,nLun);

//STEP 3)
VCSCSIAddDiskWriteTest_Blocks(-1, //Write the entire drive
                               1, //Sequential access
                               0, //Start LBA
                               eIncrementing, //pattern type
                               NULL, //Pattern File,
                               TRUE //Overlay LBA);

VCSCSIAddDiskWorkStationTest_Blocks(50000000,      //Number of blocks
                                    0,             //Start LBA
                                    eIncrementing, //Pattern
                                    NULL,          //Pattern File
                                    TRUE,          //Overlay LBA
                                    65,            //65% Writes
                                    128,           //Max Transfer Len
                                    TRUE,          //Compare Data
                                    NULL);         //Special Options

//STEP 4)
VCSCSIStartDiskTestSequence();

//STEP 5)
BOOL bAllDrvDone;
```

```
eTEST_STATUS eTestStatus;
TEST_PROGRESS_INFO sTestProgressInfo;
int nNumTests = 2;  //We added 2 tests
int nNumDrvDone;
int nNumDrvsInTest = 1;  //We are testing 1 drive

bAddDrvDone = FALSE;

while (bAllDrvDone == FALSE)
{
  Sleep(1000);  //Check status every once-per-second
  VCSCSICMQ();  //Make sure Window's messages are being processed
  nNumDrvDone = 0;

  //The below code checks the status of each drive – in our
  //example we are only testing 1 drive, but if you are testing
  //more than 1 drive you would have a "for" loop checking each
  //drive

  //Check the status of the last test
  eTestStatus = VCSCSIGetDiskTestStatus(nHA,nTid,nLun,
                                        nNumTests-1);

  if (eTestStatus != eTestNotStartedYet &&
      eTestStatus != eTestInProgress)
  {
    //The drive has finished last test
    nNumDrvDone++;
  }
  if (nNumDrvDone == nNumDrvsInTest)
  {
    bAllDrvDone = TRUE;
  }
}
```