SCSI TOOLBOX, LLC

Developer Toolbox API - DTB

VC function list 110407

Contents

Function List

BAM functions

The DTB BAM functions enable you to capture all I/O to and from any device on your system, in real time.

Please refer to Appendix C "Using BAM from a program" for example code

API: VCSCSIBAMconfigure

int VCSCSIBAMconfigure(long BufSize, long PhaseSize, int Flags, int Phases);

Discussion: Called to configure BAM to capture a trace. You must call this function before calling any other BAM functions

BufSize:
is the size of the BAM buffer, in MB to be no larger than 96.

PhaseSize:
The amount of data captured for each I/O event, in bytes, to be no larger than 65536.

Flags:
Action settings:
1 =MODE_ STOP_ON_FULL – stops the trace when the buffer is full
2 = MODE_STOP_ON_RESET – stops the trace if a bus reset is detected
4=MODE_CLEAR – clears the trace buffer

Phases:
Which I/O events you want to capture, from this list. Passing 0 will choose all of these phases.
Phases = SENSE_PHASE | OK_PHASE | CDB_PHASE | DATA_IN_PHASE | DATA_OUT_PHASE | ATA_PHASE |ATA_STATUS_PHASE | SRB_PHASE | SRB_STATUS_PHASE | RESET_PHASE;

Returns:
0 = success
-1 = failure

API: VCSCSIBAMclearBuffer

int VCSCSIBAMclearBuffer();

Discussion: Called to clear the BAM trace buffer. You may call this function at any time after calling VCSCSIBAMConfigure to clear out all trace data from the trace buffer

Returns:
0 = success
-1 = failure

API: VCSCSIBAMdrive

int VCSCSIBAMdrive(int ha, int target, int lun, int capture);

Discussion: Call this function to set or reset which drives I/O will be captured.

Ha,target,lun:
The Host Adapter, Target, and LUN of the drive to set/reset

Capture:
0 = do not capture (only needed to stop capture on a drive previously set to capture)
1 = capture

Returns:
0 = success
-1 = failure

API: VCSCSIBAMstartCapture

int VCSCSIBAMstartCapture();

Discussion: call this function to start capturing I/O on the drives which have been selected via VCSCSIBAMdrive, with the settings which have been set via VCSCSIBAMconfigure

Returns:
0 = success
-1 = failure

API: VCSCSIBAMstopCapture

int VCSCSIBAMstopCapture();

Discussion: call this function to stop capturing I/O on the drives which have been selected via VCSCSIBAMdrive, with the settings which have been set via VCSCSIBAMconfigure. Always call this function before calling any other BAM functions if a trace capture has been started.

Returns:
0 = success
-1 = failure

API: VCSCSIBAMsaveCapture

int VCSCSIBAMsaveCapture(BYTE *fname,eBAM_FILE_TYPES eBamFileType);

Discussion: This function saves the BAM trace data to a file.

fname:
the filename to save the trace data to

eBamFileType:
eBAMFileRaw = BAM-formatted file. This file type can only be opened using BAM
eBAMFileExcel = Excel (CSV) comma-delimited text file. Can be opened with text editors or spreadsheets

Returns:
0 = success
-1 = failure

ATA task register command functions

The DTB ATA functions enable you to issue ATA/SATA task register commands to ATA or SATA devices. Due to limitations in Windows these functions will only work with drives which are connected to native ATA or SATA motherboard ports. These commands will NOT function with add-in (PCI) ATA or SATA cards.

The best method to issue ATA commands to multiple drives is to use the SAT protocol to issue embedded ATA commands within the SCSI SAT command. See the appendix A “Using SAT” for details on this method.

API: VCSCSIUserATACdb

int VCSCSIUserATACdb(int ha, int target, int lun, BYTE *cdb, int datadir, int buffer);

Discussion: Used to issue an 28-bit ATA Task Register command to a device connected to a native motherboard SATA port.
Note- this function does not return the ATA status register values. Use VCSCSIATACdbFull if you need to examine the ATA status register values.

Ha,target,lun:
The Host Adapter, Target, and LUN of the drive to send the command to

cdb:
the 7-byte (28-bit) task register command

datadir:
0 = out (*from* controller *to* drive)
1=in (*from* drive *to* controller)

Buffer:
0 = DTB buffer 0
1=DTB buffer 1

Returns:
0 = success
<0 = failure

API: VCSCSIUserATACdbFull

int VCSCSIUserATACdbFull(int ha, int target, int lun, BYTE *cdb, int datadir, int buffer, int datalength, int cmdlength, int timeout);

Discussion: Used to issue either 28-bit or 48-bit ATA Task Register command to a device connected to a native motherboard SATA port. The task register return values are returned to the array used to pass in the command, and a command timeout is implemented.

Ha,target,lun:
The Host Adapter, Target, and LUN of the drive to send the command to

cdb:
the 7-byte (28-bit) task register command

datadir:
0 = out (*from* controller *to* drive)
1=in (*from* drive *to* controller)

Buffer:
0 = DTB buffer 0
1=DTB buffer 1

datalength:
data transfer length – almost always should be 512

cmdlength:
number of bytes in the command. 7 for 28-bit commands, >7 for 48-bit commands.

timeout:
command timeout value – in seconds
note: it is VERY IMPORTANT to set a reasonable timeout value – 5-15 seconds is suggested

Returns:
0 = success
<0 = failure

The cdb array is populated with the task register return values

General Functions

General functions are housekeeping and buffer management functions which generally do not issue commands to specific drives.

API: VCSCSICMQ

int VCSCSICMQ(void);

Discussion: forces Windows message queue to flush events. Call this within loops to help keyboard & mouse responsiveness, screenrefreshes, etc.

returns:
always returns 0

API: VCSCSISetTimeout

int VCSCSISetTimeout(int sec);

Discussion: sets the CDB timeout of all subsequent commands

sec:
the number of seconds to set all CDB timeouts to. 0 = unlimited timeout

Returns:
always returns 0

API: VCSCSIResetHBA

int VCSCSIResetHBA(int hba);

Discussion: attempts to issue a bus reset

hba:
the bus to reset

Returns:
always returns 0

API: VCSCSIHardReset

int VCSCSIHardReset(int ha, int target, int lun);

Discussion: Used to issue a bus reset . A drive must be present on the bus which is to be reset.

Ha,target,lun:
The Host Adapter, Target, and LUN of a drive on the bus you wish to reset

Returns:
0 = success
<0 = failure

API: VCSCSIAnd

Int int VCSCSIAnd(int number1, int number2);

Discussion: used to logically AND two numbers

Number1, number 2:
The two numbers you wish to logically AND

Returns:
The result of the logical AND operation

API: VCSCSIOr

Int int VCSCSIOr(int number1, int number2);

Discussion: used to logically OR two numbers

Number1, number 2:
The two numbers you wish to logically OR

Returns:
The result of the logical OR operation

API: VCSCSIXor

Int int VCSCSIXor(int number1, int number2);

Discussion: used to logically OR two numbers

Number1, number 2:
The two numbers you wish to logically XOR

Returns:
The result of the logical XOR operation

API: VCSCSIHex2Dec

unsigned long VCSCSIHex2Dec(BYTE *hexdata);

Discussion: Used to convert a number represented as four hexadecimal bytes into an unsigned long.

Hexdata:
A pointer to a four bytes hexadecimal number

Returns:
An unsigned long

API: VCSCSIDec2Hex

int VCSCSIDec2Hex(long ul, BYTE *hexdata);

Discussion: used to convert an unsigned long number into a four-byte hexadecimal number

ul:
an unsigned long number

***hexdata:**
A pointer to four bytes of hexadecimal

Returns:
The four-byte hexadecimal representation of the number is returned in *hexdata, the function return value always returns 0

API: VCSCSIBufferSize

long VCSCSIBufferSize();

Discussion:
This function returns the size of the data buffer available.
The address of bb1->bfr[0] = this value/2

Returns:
The size of DTBs data buffer

API: VCSCSIFillRandom

int VCSCSIFillRandom(int buffer, int blocksize, long startingblock, int numberofblocks);

Discussion: Fills the specified buffer with the specified number of blocks worth of random data. The LBA number and the random seed is written at the beginning of each block.

buffer:
0=fill DTB buffer 0
1=fill DTB buffer 1

blocksize:
the value to specify what blocksize to block the data with

startingblock:
the first LBA to create the data pattern for

numberofblocks:
How many blocks of data to create

Returns:
Returns the number of blocks created

API: VCSCSICheckRandomBlock

Int VCSCSICheckRandomBlock(int buffer, int blocksize, long blocknum, long offset, int numberofblocks);

Discussion: Processes a block which was created with VCSCSIFillRandom, checks that the LBA matches, regenerates the random data based on the seed value in the block data and checks that the random data is correct.

buffer:
Not used

blocksize:
The blocksize which was used when creating the random data fill

blocknum:
The expected LBA

offset:
Not used

numberofblocks:
How many blocks to check

Returns:
-1 = success
0=block numbers do not match
>=8 = Data miscompare – value is byte offset into block where miscompare occurred
Note: Use VCSCSIGetRandomErrors for more error detail if return value != -1

API: VCSCSIGetRandomErrors

int VCSCSIGetRandomErrors(long *expected_blocknum, long *actual_blocknum, int *block, int *offset, int *expected_data, int *actual_data);

Discussion: returns detailed data of LBA or data miscompares generated by VCSCSICheckRandomBlock

***expected_blocknum:**
The blocknum (LBA) which should have been in the data

***actual_blocknum:**
The blocknum (LBA) which was in the data

***block:**
The LBA where a data miscompare occurred

***offset:**
The byte offset within the block where a data miscompare occurred

Returns:
1 = success

API: VCSCSIFileOffset2Buffer

int VCSCSIFileOffset2Buffer(int buffnum, int datalen,char *FileName, long offset);

Discussion: Opens the specified file, moves the specified number of bytes(starting from *offset* bytes) into the specified buffer.

buffnum:
0=put file data in DTB buffer 0
1=put file data in DTB buffer 1

datalen:
The number of bytes to transfer from the file into the buffer

***FileName:**
Pointer to the file name

offset:
The number of bytes into the file to start transferring from

Returns:
0=success
-1=failure

API: VCSCSIRollPattern

int VCSCSIRollPattern(int buffer, long start, long number);

Discussion:
creates a pattern in the specified buffer

buffer:
0=create pattern in DTB buffer 0
1=create pattern in DTB buffer 1
start:
The starting byte for the data pattern. Each subsequent location within the pattern will be this byte incremented by 1. The byte will roll over.

number:
The number of bytes of data pattern to create

Returns:
Always returns 1

API: VCSCSIGetDLLVersion

int VCSCSIGetDLLVersion();

returns:
returns the version of DTB

API: VCSCSIBuffer2File

int VCSCSIBuffer2File(int buffnum, int datalen, char *FileName);

Discussion: copies the specified number of bytes from the buffer to a file

buffnum:
0=DTB buffer 0
1=DTB buffer 1

datalen:
How many bytes to write to the file

***FileName**:
The name of the file to write to

Returns:

0=success
-1=failure

API: VCSCSIBuffer2FileLong

int VCSCSIBuffer2File(int buffnum, long datalen, char *FileName);

Discussion: copies the specified number of bytes from the buffer to a file

buffnum:
0=DTB buffer 0
1=DTB buffer 1

datalen:
How many bytes to write to the file

***FileName**:
The name of the file to write to

Returns:
0=success
-1=failure

API: VCSCSIFile2Buffer

int VCSCSIBuffer2File(int buffnum, int datalen, char *FileName);

Discussion: copies the specified number of bytes from the file to the buffer

buffnum:
0=DTB buffer 0
1=DTB buffer 1

datalen:
How many bytes to read from the file

***FileName**:
The name of the file to read from

Returns:
0=success
-1=failure

API: VCSCSIFileLong2Buffer

int VCSCSIBuffer2File(int buffnum, long datalen, char *FileName);

Discussion: copies the specified number of bytes from the file to the buffer

buffnum:
0=DTB buffer 0
1=DTB buffer 1

datalen:
How many bytes to read from the file

***FileName**:
The name of the file to read from

Returns:
0=success
-1=failure

API: VCSCSIFillBuffer

int VCSCSIFillBuffer(int buffer, long numbytes, int patternsize,BYTE *bufferdata);

Discussion: this function fills the specified buffer with a pattern. You specify the overall size of the fill you want, then specify the size of the pattern which you want repeated through the buffer, and then you specify what that pattern is. So for example, you could fill the entire buffer with a repeating 20-block data pattern.

buffer:
0=DTB buffer 0
1=DTB buffer 1

numbytes:
The total number of bytes to write

patternsize:
The size of the data sub-pattern

***bufferdata:**
A pointer to your data pattern

Returns:
Always returns 1

API: VCSCSIFillPattern

int VCSCSIFillPattern(int buffer, int pattern);

Discussion: this function fills the specified buffer with a pattern. You specify the overall size of the fill you want, then specify from a choice of 4 patterns. The entire buffer will be filled with the selected pattern.

buffer:
0=DTB buffer 0
1=DTB buffer 1

pattern:
0=all zero's
1=all one's
2=0xA5's
3=random

Returns:
Always returns 1

API: VCSCSIFillBlockNum

int VCSCSIFillBlockNum(int buffer, long sblock, int count, int blocksize);

Discussion: this function fills buffer # *buffer* with disk blocknumber data, starting at byte 0 of buffer, continuing for *count* X *blocksize* bytes (count blocks of data)

buffer:
0=DTB buffer 0
1=DTB buffer 1

sblock:
the starting LBA

count:
how many blocks of data to create

blocksize:
the drive blocksize you want to data to be represented in

Returns:
always returns 1

API: VCSCSISearchBuffer

int VCSCSISearchBuffer(int buffer, BYTE *searchdata, int searchsize, long searchlength);

Discussion: Searches buffer # buffer for the first occurance of searchdata. Searchlength specifies how much of the buffer to search (-1 searches the entire buffer), searchsize specifies the number of significant bytes in the pattern searchdata.

buffer:
0=DTB buffer 0
1=DTB buffer 1

***searchdata:**
Pointer to your search pattern

searchsize:
size of your search pattern

searchlength:
how much of the buffer should be searched. Passing -1 will search the entire buffer

Returns:
returns byte count of the first byte of buffer that matches pattern on success, -1 on failure

API: VCSCSILoadBuffer

int VCSCSILoadBuffer(int buffer, long numbytes, BYTE *bufferdata);

Discussion: Fills buffer # *buffer* with *count* bytes of *bufferdata*. This loads one byte of pattern, one byte at a time into the buffer.

buffer:
0=DTB buffer 0
1=DTB buffer 1

numbytes:
how many times to write your pattern byte into the buffer

***bufferdata:**
Pointer to yoursingle-byte data pattern

Returns:
Always returns 1

API: VCSCSIGetBuffer

int VCSCSIGetBuffer(int buffer, long numbytes, BYTE *bufferdata);

Discussion: Retrieves *numbytes* bytes of data from buffer # *buffer*, returns data in byte array *bufferdata*.

buffer:
0=DTB buffer 0
1=DTB buffer 1

numbytes:
How many bytes of data to retrieve from the buffer

Returns:
*bufferdata – pointer to array of BYTES to hold your data

Always returns 1

API: VCSCSICompareBuffers

int VCSCSICompareBuffers(long startbyte, long numbytes);

Discussion: Compares the contents of two buffers buffer 0 and buffer 1), starting from *startbyte*, for length *numbyte*s.

startbyte:
Location or offset into buffer where you want the compare to begin

numbytes:
How many bytes of the buffer do you want to compare

Returns:
returns 0 on success, or the byte number where the miscompare occurred on failure

API: VCSCSIHostAdapterCount

int VCSCSIHostAdapterCount();

returns:
returns the number of host bus adapters in the system

API: VCSCSITargetCount

int VCSCSITargetCount(int ha);

Discussion: returns the number of targets on the specified HBA

ha:
the HBA to count targets on

returns:
the number of targets on the HBA

API: VCSCSITargetPerBus

int VCSCSITargetsPerBus(int ha);

Discussion: returns the number of targets per bus reported by the OS

ha:
the HBA to count targets per bus

returns:
the number of targets per bus on the HBA

Tape Drive Oriented Functions

API: VCSCSIGetBufferMode

Int VCSCSIGetBufferMode(int ha, int target, int lun);

Discussion: Uses the SCSI MODE SENSE CDB to determine whether a tape device is set to buffered or non-buffered mode.

Ha,target,lun:
The Host Adapter, Target, and LUN of the tape drive in question

Returns:
0 = success
<0 = failure

API: VCSCSISetBufferMode

Int VCSCSISetBufferMode(int ha, int target, int lun, int mode);

Discussion: Uses the SCSI MODE SELECT CDB to set a tape drive into either buffered or non-buffered mode

Ha,target,lun:
The Host Adapter, Target, and LUN of the tape drive to change buffer setting

Returns:
0 = success
<0 = failure

API: VCSCSIGetTapeCapacity

int VCSCSIGetTapeCapacity(int ha, int target, int lun, long *TBS);

Discussion: Issues a SCSI MODE SENSE command to drive specified by *ha/target/lun*, returns Tape Block Length (Mode Page Block Descriptor bytes 5-7) in long *TBS*.

ha, target, lun:
The address (host bus adapter, target, and LUN) of the tape drive

Returns:
*TBS contains the Tape Block Length

returns 1 on success, -1 on failure

API: VCSCSISetTapeBlocksize

int VCSCSISetTapeBlocksize(int ha, int target, int lun, long blocksize);

Discussion: sets the blocksize of the specified drive to *blocksize*.

ha, target, lun:
The address (host bus adapter, target, and LUN) of the tape drive

Returns:
1=success
-1=failure

API: VCSCSITapeRewind

int VCSCSITapeRewind(int ha, int target, int lun, int immediate);

Discussion: rewinds the tape.

ha, target, lun:
The address (host bus adapter, target, and LUN) of the tape drive

immediate:
0=command or function will return once the rewind is physically complete
1=command/function will complete immediately

Returns:
0=success
Non-zero=failure

API: VCSCSITapeUnload

int VCSCSITapeUnload(int ha, int target, int lun, int immediate);

Discussion: unloads/ejects the tape.

ha, target, lun:
The address (host bus adapter, target, and LUN) of the tape drive

immediate:
0=command or function will return once the unload is physically complete
1=command/function will complete immediately

Returns:
0=success
Non-zero=failure

API: VCSCSITapeReadF

int VCSCSITapeReadF(int ha, int target, int lun, int count, int buffer);

Discussion: Reads one or more blocks from the tape in FIXED BLOCK mode.

ha, target, lun:
The address (host bus adapter, target, and LUN) of the tape drive

count:
the number of blocks you wish to read

buffer:
0=data is read into DTB buffer 0
1=data is read into DTB buffer 1

Returns:
0=success
Non-zero=failure

API: VCSCSITapeWriteF

int VCSCSITapeWriteF(int ha, int target, int lun, int count, int buffer);

Discussion: Writes one or more blocks to the tape in FIXED BLOCK mode.

ha, target, lun:
The address (host bus adapter, target, and LUN) of the tape drive

count:
the number of blocks you wish to write

buffer:
0=data is written from DTB buffer 0
1=data is written from DTB buffer 1

Returns:
0=success
Non-zero=failure

API: VCSCSITapeReadV

int VCSCSITapeReadV(int ha, int target, int lun, int buffer);

Discussion: Reads one block from the tape in VARIABLE mode. The CDB will use 32767 bytes as the transfer length

ha, target, lun:
The address (host bus adapter, target, and LUN) of the tape drive

buffer:
0=data is read into DTB buffer 0
1=data is read into DTB buffer 1

Returns:
0=success
Non-zero=failure

API: VCSCSITapeReadVL

int VCSCSITapeReadVL(int ha, int target, int lun, long count, int buffer);

Discussion: Reads one block from the tape in VARIABLE mode. The block size or transfer length is specified in the parameter *count*.

ha, target, lun:
The address (host bus adapter, target, and LUN) of the tape drive

count:
The transfer length to be used in the CDB

buffer:
0=data is read into DTB buffer 0
1=data is read into DTB buffer 1

Returns:
0=success
Non-zero=failure

API: VCSCSITapeWriteV

int VCSCSITapeWriteV(int ha, int target, int lun, long count, int buffer);

Discussion: Writes one block to the tape in VARIABLE mode.

ha, target, lun:
The address (host bus adapter, target, and LUN) of the tape drive

count:
the size of the block ie: the transfer length used in the CDB

buffer:
0=data is written from DTB buffer 0
1=data is written from DTB buffer 1

Returns:
0=success
Non-zero=failure

API: VCSCSITapeWFM

int VCSCSITapeWFM(int ha, int target, int lun);

Discussion: writes a file mark to the tape.

ha, target, lun:
The address (host bus adapter, target, and LUN) of the tape drive

Returns:
0=success
Non-zero=failure

API: VCSCSITapeFSF

int VCSCSITapeFSF(int ha, int target, int lun);

Discussion: Uses the SCSI SPACE command to space forward one File Mark

ha, target, lun:
The address (host bus adapter, target, and LUN) of the tape drive

Returns:
0=success
Non-zero=failure

API: VCSCSITapeFSR

int VCSCSITapeFSR(int ha, int target, int lun);

Discussion: Uses the SCSI SPACE command to space reverse one File Mark

ha, target, lun:
The address (host bus adapter, target, and LUN) of the tape drive

Returns:
0=success
Non-zero=failure

API: VCSCSITapeSpaceEOD

int VCSCSITapeSpaceEOD(int ha, int target, int lun);

Discussion: Uses the SCSI SPACE command to spaceforward to EOD (End Of Data)

ha, target, lun:
The address (host bus adapter, target, and LUN) of the tape drive

Returns:
1=success
Non-one=failure

NOTE – return value reversed from normal!!

Tape Drive Firmware Download Functions

API: VCSCSISegmented_FWDL

int VCSCSISegmented_FWDL(int ha, int target, int lun,char *FileName);

Discussion: a device (tape or disk) firmware download using the SCSI WRITE BUFFER CDB, using 0x2000 byte segments, download mode 0x07.

Ha,target,lun:
The Host Adapter, Target, and LUN of the drive to be downloaded to

FileName:
A pointer to the firmware file name

Returns:
0 = success
<0 = failure

API: VCSCSIDLT_FWDL

int VCSCSIDLT_FWDL(int ha, int target, int lun,char *FileName);

Discussion: a firmware download for DLT tape drive

Ha,target,lun:
The Host Adapter, Target, and LUN of the drive to be downloaded to

FileName:
A pointer to the firmware file name

Returns:
0 = success
<0 = failure

API: VCSCSISDLT_FWDL

int VCSCSISDLT_FWDL(int ha, int target, int lun,char *FileName);

Discussion: a firmware download for Super DLT tape drive

Ha,target,lun:
The Host Adapter, Target, and LUN of the drive to be downloaded to

FileName:
A pointer to the firmware file name

Returns:
0 = success
<0 = failure

API: VCSCSIIBMLTO_FWDL

int VCSCSIIBMLTO_FWDL(int ha, int target, int lun,char *FileName);

Discussion: a firmware download for IBM LTO tape drive

Ha,target,lun:
The Host Adapter, Target, and LUN of the drive to be downloaded to

FileName:
A pointer to the firmware file name

Returns:
0 = success
<0 = failure

API: VCSCSIHPLTO_FWDL

int VCSCSIHPLTO_FWDL(int ha, int target, int lun,char *FileName);

Discussion: a firmware download for HP LTO tape drive

Ha,target,lun:
The Host Adapter, Target, and LUN of the drive to be downloaded to

FileName:
A pointer to the firmware file name

Returns:
0 = success
<0 = failure

API: VCSCSISeagateLTO_FWDL

int VCSCSISeagateLTO_FWDL(int ha, int target, int lun,char *FileName);

Discussion: a firmware download for Seagate LTO tape drive

Ha,target,lun:
The Host Adapter, Target, and LUN of the drive to be downloaded to

FileName:
A pointer to the firmware file name

Returns:
0 = success
<0 = failure

API: VCSCSISonyAIT_FWDL

int VCSCSISonyAIT_FWDL(int ha, int target, int lun,char *FileName);

Discussion: a firmware download for Sony AIT tape drive

Ha,target,lun:
The Host Adapter, Target, and LUN of the drive to be downloaded to

FileName:
A pointer to the firmware file name

Returns:
0 = success
<0 = failure

Disk Drive Oriented Functions

API: VCSCSISegmented_FWDL

int VCSCSISegmented_FWDL(int ha, int target, int lun,char *FileName);

Discussion: a device (tape or disk) firmware download using the SCSI WRITE BUFFER CDB, using 0x2000 byte segments, download mode 0x07.

Ha,target,lun:
The Host Adapter, Target, and LUN of the drive to be downloaded to

FileName:
A pointer to the firmware file name

Returns:
0 = success
<0 = failure

API: VCSCSIDiskStartStop

int VCSCSIDiskStartStop(int ha, int target, int lun, int start);

Discussion: issues a SCSI LOAD/UNLOAD (0x1B) CDB

ha,target,lun:
The Host Adapter, Target, and LUN of the device

start:
1=start or spin up drive
0=stop or spin down drive

Returns:
0=success
Non-zero = failure

API: VCSCSIDiskUnload

int VCSCSIDiskUnload(int ha, int target, int lun);

Discussion: issues a SCSI UNLOAD (0x1B) CDB

ha,target,lun:
The Host Adapter, Target, and LUN of the device

Returns:
0=success
Non-zero = failure

API: VCSCSIDiskFormat

int VCSCSIDiskFormat(int ha, int target, int lun, int mode);

Discussion: issues a SCSI FORMAT (0x04) CDB

ha,target,lun:
The Host Adapter, Target, and LUN of the device

mode:
not used

returns:
0=success
Non-zero = failure

API: VCSCSIReadCapacity

int VCSCSIReadCapacity(int ha, int target, int lun, long * highblock_l, long * blocksize_l);

Discussion: returns the highest block and block size reported in the header of a MODE SENSE CDB.

ha,target,lun:
The Host Adapter, Target, and LUN of the device

Returns data:
*highblock_l = highest LBA reported
*blocksize_l= blocksize reported
Function returns
0=success
-1 = failure

API: VCSCSIDiskRead

int VCSCSIDiskRead(int ha, int target, int lun, int count, long sblock, long bsize, int buffer);

Discussion: Reads count number of blocks, starting a sblock, of blocks size bsize, into buffer. Uses SCSI READ 10 (0x28) CDB

ha,target,lun:
The Host Adapter, Target, and LUN of the device

count:
The number of blocks to transfer

sblock:
The block (LBA) to start transferring from

bsize:
The blocksize of the drive

buffer:
0=DTB buffer 0
1=DTB buffer 1

Returns:
0=success
Non-zero = failure

API: VCSCSIDiskWrite

int VCSCSIDiskWrite(int ha, int target, int lun, int count, long sblock, long bsize, int buffer);

Discussion: writes count number of blocks, starting a sblock, of blocks size bsize, from buffer. Uses SCSI WRITE 10 (ox2A) CDB

ha,target,lun:
The Host Adapter, Target, and LUN of the device

count:
The number of blocks to transfer

sblock:
The block (LBA) to start transferring to

bsize:
The blocksize of the drive

buffer:
0=DTB buffer 0
1=DTB buffer 1

Returns:
0=success
Non-zero = failure

API: VCSCSIDiskCorruptBlock

int VCSCSIDiskCorruptBlock(int ha, int target, int lun, long sblock, int span);

Discussion:
Uses the SCSI WRITE LONG CDB to create corrupted blocks – blocks with either correctable or non-correctable data errors, depending on the length of the ECC span specified.
Note: To "un-corrupt" any previously correpted blocks simply use the standard VCSCSIDiskWrite command or execute a sequential write test over the corrupted block range.

Ha,target,lun:
The Host Adapter, Target, and LUN of the drive to corrupt

sblock:
The block to corrupt

span:
The length (in bytes) that the data error will span. Setting this value to less-than the ECC Correction Span value of the drive will create a correctable error. Setting this value to greater-than the ECC Correctin Span value of the drive will create a non-correctable error.
Note: See the VCSCSIDiskGetECCSpan function

Returns:
0=success
Non-zero=failure

API: VCSCSIDiskGetECCSpan

int VCSCSIDiskGetECCSpan(int ha, int target, int lun);

Discussion: returns the ECC Correction Span length of the specified drive.

Ha,target,lun:
The Host Adapter, Target, and LUN of the drive

Returns:
Returns the ECC Correction Span value of the drive

API: VCSCSIDiskGetReadLongSize

Int int VCSCSIGetReadLongSize(int ha, int target, int lun);

Discussion: This function returns the Read Long size. This is the value which should be used with the VCSCSIDiskWriteLong function.

Ha,target,lun:
The Host Adapter, Target, and LUN of the drive

Returns:
Returns the READ LONG size of the drive

API: VCSCSIDiskReadLong

int VCSCSIDiskReadLong(int ha, int target, int lun, int correct, long sblock, int bsize, int buffer);

Discussion: This function executes a SCSI READ LONG CDB

Ha,target,lun:
The Host Adapter, Target, and LUN of the drive to read

sblock:
The block to read

correct:
1=set the CORRECT bit (CDB byte 1 = 0x02)
0=clear the CORRECT bit (CDB byte 1 = 0x0)

bsize:
The raw block size for this drive. Use VCSCSIGetReadLongSize to determine what this value should be for each drive.

buffer:
0=Read the data into DTB buffer 0
1=Read the data into DTB buffer 1

Returns:
0=success
Non-zero = failure

API: VCSCSIDiskWriteLong

int VCSCSIDiskWriteLong(int ha, int target, int lun, long sblock, int bsize, int buffer);

Discussion: This function executes a SCSI READ LONG CDB

Ha,target,lun:
The Host Adapter, Target, and LUN of the drive to write

sblock:
The block to write

bsize:
The raw block size for this drive. Use VCSCSIGetReadLongSize to determine what this value should be for each drive.

buffer:
0=Write the data from DTB buffer 0
1=Write the data from DTB buffer 1

Returns:
0=success
Non-zero = failure

API: VCSCSIDiskReadFUA

int VCSCSIDiskReadFUA(int ha, int target, int lun, int count, long sblock, long bsize, int buffer);

Discussion: issues a 10-byte SCSI READ 0x28 CDB with the FUA bit set to Force Unit Access.
FUA causes the read to be forced forced from the drive media, ignoring any data which may be in cache.

Ha,target,lun:
The Host Adapter, Target, and LUN of the drive to read

count:
How many blocks to read

sblock:
The starting LBA to read from

bsize:
The block size of the drive sector

buffer:
0=read into DTB buffer 0
1=read into DTB buffer 1

Returns:
0=success
Non-zero = failure

API: VCSCSIDiskWriteFUA

int VCSCSIDiskWriteFUA(int ha, int target, int lun, int count, long sblock, long bsize, int buffer);

Discussion: issues a 10-byte SCSI WRITE 0x2A CDB with the FUA bit set to Force Unit Access.
FUA causes the write to be forced forced to the drive media, ignoring any cache. The command will not complete until the data is written to the media.

Ha,target,lun:
The Host Adapter, Target, and LUN of the drive to read

count:
How many blocks to write

sblock:
The starting LBA to write to

bsize:
The block size of the drive sector

buffer:
0=written from DTB buffer 0
1=written from DTB buffer 1

Returns:
0=success
Non-zero = failure

Jukebox/Library Oriented Functions

API: VCSCSIPositionToElement

int VCSCSIPositionToElement(int ha, int target, int lun, int trans_add, int dest_add);

Discussion: Moves the hand or picker of a library/jukebox to a specified media storage cell.

ha,target,lun:
The Host Adapter, Target, and LUN of thejukebox

trans_add:
The element address for the jukebox picker/arm/transport

dest_add:
The storage element address to move the picker to

Returns:
0=success
Non-zero = failure

API: VCSCSIInitializeElementStatusRange

int VCSCSIInitializeElementStatusRange(int ha, int target,int lun, int range,int address, int num);

Discussion: sends a SCSI INTIALIZE ELEMENT STATUS w/RANGE (0xE7) CDB.

ha,target,lun:
The Host Adapter, Target, and LUN of thejukebox

range:
1=set cdb[1] = 0x01
0=set cdb[1]=0x0

address:
Loaded into cdb[2-3]

num:
Loaded into cdb[6-7]

Returns:
0=success
Non-zero = failure

API: VCSCSIInitializeElementStatus

int VCSCSIInitializeElementStatus(int ha, int target,int lun);

Discussion: sends a SCSI INTIALIZE ELEMENT STATUS (0x07) CDB.
Note: Some libraries can take a VERY long time – hours – to complete this command.

ha,target,lun:
The Host Adapter, Target, and LUN of thejukebox

Returns:
0=success
Non-zero = failure

API: VCSCSIReadElementStatus

int VCSCSIReadElementStatus(int ha, int target,int lun, int eltype, int start, int num, long length, BYTE *eldata);

Discussion: Issues a SCSI READ ELEMENT STATUS (0xB8) CDB.

ha,target,lun:
The Host Adapter, Target, and LUN of thejukebox

etype:
Which element type to return.
0x0 = All element types
0x01=Medium Transport element (picker/hand)
0x02=Storage Element
0x03=Import/Export Element (mailbox)

start:
The minimum element address to report

num:
The maximum number of elements to report

length:
The size of the data buffer in bytes

Returns:
*eldata = pointer to the requested element data
0=success
Non-zero = failure

API: VCSCSIReadElementStatusVolTag

int VCSCSIReadElementStatusVolTag(int ha, int target,int lun, int eltype, int start, int num, long length, BYTE *eldata);

Discussion: Issues a SCSI READ ELEMENT STATUS (0xB8) CDB setting the VolTag bit

ha,target,lun:
The Host Adapter, Target, and LUN of thejukebox

etype:
Which element type to return.
0x0 = All element types
0x01=Medium Transport element (picker/hand)
0x02=Storage Element
0x03=Import/Export Element (mailbox)

start:
The minimum element address to report

num:

The maximum number of elements to report

length:
The size of the data buffer in bytes

Returns:
*eldata = pointer to the requested element data
0=success
Non-zero = failure

API: VCSCSIMoveMedium

int VCSCSIMoveMedium(int ha, int target, int lun, int trans_add, int source_add, int dest_add);

Discussion: Issues a SCSI MOVE MEDIUM (0xA5) CDB to move a piece of medium from the specified source element address to the specified destination element address.

ha,target,lun:
The Host Adapter, Target, and LUN of thejukebox

trans_add:
the address of the picker to use for the move

source_add:
the address of the element containing the medium to move

dest_add:
the destination element address to move the medium to

Returns:
0=success
Non-zero = failure

General SCSI Commands Functions

API: VCSCSIGetVendor

int VCSCSIGetVendor(int ha, int target, int lun, char *VendorString);

Discussion: returns the INQUIRY VENDOR string

ha,target,lun:
The Host Adapter, Target, and LUN of the device

Returns:

*VendorString – pointer to the VENDOR ID from the inquiry data
Function returns 0 = success, non-zero = failure

API: VCSCSIGetProduct

int VCSCSIGetProduct(int ha, int target, int lun, char *ProductString);

Discussion: returns the INQUIRY PRODUCT string

ha,target,lun:
The Host Adapter, Target, and LUN of the device

Returns:
*ProductString – pointer to the PRODUCT ID from the inquiry data
Function returns 0 = success, non-zero = failure

API: VCSCSIGetVersion

int VCSCSIGetVersion(int ha, int target, int lun, char *VersionString);

Discussion: returns the INQUIRY VERSION string

ha,target,lun:
The Host Adapter, Target, and LUN of the device

Returns:
*ProductString – pointer to the VERSION from the inquiry data
Function returns 0 = success, non-zero = failure

API: VCSCSIInquiry

int VCSCSIInquiry(int ha, int target, int lun, BYTE *inqdata);

Discussion: returns up to 64 bytes of INQUIRY data

ha,target,lun:
The Host Adapter, Target, and LUN of the device

Returns:
*ProductString – pointer to the raw inquiry data
Function returns 0 = success, non-zero = failure

API: VCSCSIInquiryEVPD

int VCSCSIInquiryEVPD(int ha, int target, int lun, int page, BYTE *inqdata);

Discussion: returns up to 64 bytes of INQUIRY data from the specified VPD page

ha,target,lun:
The Host Adapter, Target, and LUN of the device

page:
The VPD page you wish to view

Returns:
*ProductString – pointer to the raw inquiry data
Function returns 0 = success, non-zero = failure

API: VCSCSITUR

int VCSCSITUR(int ha, int target, int lun);

ha,target,lun:
The Host Adapter, Target, and LUN of the device

Returns:
The READY status of the drive
0=READY
Non-zero = not ready

API: VCSCSIGetDeviceType

int VCSCSIGetDeviceType(int ha, int target, int lun);

ha,target,lun:
The Host Adapter, Target, and LUN of the device

Returns:
The DEVICE TYPE of the drive from INQUIRY byte 0

API: VCSCSIViewSense

int VCSCSIViewSense(BYTE *reqsendata);

Discussion: returns the last SENSE DATA collected – this should be from the last command which resulted in a CHECK CONDITION

returns:
*reqsendata – pointer to up to 128 bytes of sense data
Function return – always returns 1

API: VCSCSIUserCdb

int VCSCSIUserCdb(int ha, int target, int lun, BYTE *cdb, int cdblength, int datadir, long datalength, int buffer);

Discussion: Issues the SCSI CDB secified in byte array *cdb* to the device specified in *ha/target/lun*. The length of the CDB is specified in *cdblength*, data direction is specified by *datadir*(0=out from host adapter, 1 = in to host adapter), length of data transferred is specified by *datalength*, and buffer # is specified by *buffer*.

ha, target, LUN:
The address (host bus adapter, target, and LUN) of the device you want to send the CDB to.

***cdb:**
A pointer to an array of BYTES which contain your CDB bytes

cdblength:
The CDB size – 6, 10, 12, or 16 bytes

datadir:
0=data goes *out from* the HBA *to* the target
1=data comes *in from* the target *to* the HBA

datalength:
The size (if any) of any data phase your CDB will have. This size should match the CDB's Allocation Length value.

buffer:
0=use DTB buffer 0
1=use DTB buffer 1

Returns:
1 on success,
-1 on failure

API: VCSCSIUserCdbTimeout

int VCSCSIUserCdbTimeout(int ha, int target, int lun, BYTE *cdb, int cdblength, int datadir, long datalength, int buffer, int tOut);

Discussion: Issues the SCSI CDB secified in byte array *cdb* to the device specified in *ha/target/lun*. The length of the CDB is specified in *cdblength*, data direction is specified by *datadir*(0=out from host adapter, 1 = in to host adapter), length of data transferred is specified by *datalength*, and buffer # is specified by *buffer*. This function allows you to specify a timeout for the CDB – if the command has not completed within the timeout time the command will be aborted.

ha, target, LUN:
The address (host bus adapter, target, and LUN) of the device you want to send the CDB to.

***cdb:**
A pointer to an array of BYTES which contain your CDB bytes

cdblength:
The CDB size – 6, 10, 12, or 16 bytes

datadir:
0=data goes *out from* the HBA *to* the target
1=data comes *in from* the target *to* the HBA

datalength:
The size (if any) of any data phase your CDB will have. This size should match the CDB's Allocation Length value.

buffer:
0=use DTB buffer 0
1=use DTB buffer 1

tout:
timeout value in seconds. O = unlimited timeout

Returns:
1 on success,
-1 on failure

API: VCSCSIUserCdbReportUnderOverrun

int VCSCSIUserCdbReportUnderOverrun(int ha, int target, int lun, BYTE *cdb, int cdblength, int datadir, long datalength, int buffer);

Discussion: Issues the SCSI CDB secified in byte array *cdb* to the device specified in *ha/target/lun*.

The length of the CDB is specified in *cdblength*, data direction is specified by *datadir*(0=out from host adapter, 1 = in to host adapter), length of data transferred is specified by *datalength*, and buffer # is specified by *buffer*.
This function will detec t if the amount of data actually transferred during the I/O is different than the data length specified by the parameter *datalength*.

ha, target, LUN:
The address (host bus adapter, target, and LUN) of the device you want to send the CDB to.

***cdb:**
A pointer to an array of BYTES which contain your CDB bytes

cdblength:
The CDB size – 6, 10, 12, or 16 bytes

datadir:
0=data goes *out from* the HBA *to* the target
1=data comes *in from* the target *to* the HBA

datalength:
The size (if any) of any data phase your CDB will have. This size should match the CDB's Allocation Length value.

buffer:
0=use DTB buffer 0
1=use DTB buffer 1

Returns:
0 on success,
0x52 on data overrun or underrun
-1 on failure

API: VCSCSIModeSense

int VCSCSIModeSense(int ha, int target, int lun, int page, int pagecode, BYTE *modedata);

Discussion: Issues a MODE SENSE command for mode page *page*, page code *pagecode* to the drive specified by *ha/target/lun*. Mode Sense data is returned in the byte array *modedata*.
Note: this function does NOT return the Block Descriptor

ha, target, lun:
The address (host bus adapter, target, and LUN) of the device

page:
the MODE PAGE you wish to retrieve

pagecode:
The page code:
0x0= current configuration
0x01= changeable bitmap
0x02= default power-on values
0x03=saved values

Returns:
*modedata:
Pointer to an array of BYTES to contain MODE SENSE data – data will not include BYTE DESCRIPTOR

Function return:
1=success
-1=failure
NOTE – return value reversed from normal!!

API: VCSCSIModeSelect

int VCSCSIModeSelect(int ha, int target, int lun, int sp, BYTE *modedata);

Discussion: Issues a MODE SELECT command to the drive specified by *ha/target/lun*.
Note: this function does include the Block Descriptor data, start your data with the Mode Parameter Header

ha, target, lun:
The address (host bus adapter, target, and LUN) of the device

page:
the MODE PAGE you wish to send

sp:
the sp bit – CDB byte 1 bit 0

Function return:
1=success
-1=failure
NOTE – return value reversed from normal!!

API: VCSCSIModeSenseFull

int VCSCSIModeSense(int ha, int target, int lun, int page, int pagecode, BYTE *modedata);

Discussion: Issues a MODE SENSE command for mode page *page*, page code *pagecode* to the drive specified by *ha/target/lun*. Mode Sense data is returned in the byte array *modedata*.
Note: this function does return the Block Descriptor

ha, target, lun:
The address (host bus adapter, target, and LUN) of the device

page:
the MODE PAGE you wish to retrieve

pagecode:
The page code:
0x0= current configuration
0x01= changeable bitmap
0x02= default power-on values
0x03=saved values

Returns:
*modedata:
Pointer to an array of BYTES to contain MODE SENSE data – data will not include BYTE DESCRIPTOR

Function return:
1=success

-1=failure

NOTE – return value reversed from normal!!

API: VCSCSIModeSelectFull

int VCSCSIModeSelect(int ha, int target, int lun, int sp, BYTE *modedata);

Discussion: Issues a MODE SELECT command to the drive
specified by *ha/target/lun*.
Note: this function expects the Block Descriptor data, then the Mode Parameter Header, then the page data
NOTE: incorrect values in the BLOCK DESCRIPTOR can rendor your drive unusable!

ha, target, lun:
The address (host bus adapter, target, and LUN) of the device

page:
the MODE PAGE you wish to send, including Block Descriptor, Mode Parameter Header, and page data.

sp:
the sp bit – CDB byte 1 bit 0

Function return:
1=success
-1=failure

NOTE – return value reversed from normal!!

API: VCSCSILogSense

int VCSCSILogSense(int ha, int target, int lun, int page, int pagecode, BYTE *modedata);

Discussion: Issues a SCSI LOG SENSE (0x4D) CDB

ha, target, lun:
The address (host bus adapter, target, and LUN) of the device

page:
The LOG SENSE page you wish to retrieve

pagecode:
0x0=the maximum value for each log entry is returned
0x01=the current values are returned
0x02=the maximum value for each log entry is returned
0x03=the power-on values are returned

Returns:
*modedata = pointer to an array of bytes to contain the returned LOG PAGE data

Function Returns:
1=success
-1=failure

NOTE – return value reversed from normal!!

API: VCSCSILogSelect

int VCSCSILogSelect(int ha, int target, int lun, int sp, BYTE *modedata);

Discussion:

is not implemented – use VCSCSIUserCDB if you need to issue a LOG SELECT CDB.

ha, target, lun:
The address (host bus adapter, target, and LUN) of the device

sp:
the sp (save parameters) bit – CDB byte 1 bit 0

Returns:

Function Returns:

Always returns 1

API: VCSCSIGetErrorDetails

int VCSCSIGetErrorDetails(int *iostat, int *hbastat, int *scsistat);

returned values:
iostat = the most recent SRB status
hbastat = the most recent HBA status
scsistat= the most recent Target status

returns:
function always returns 0

API: VCSCSISleep

int VCSCSISleep(int nNumMilliseconds);

description: this function issues the C Sleep command.
nNumMilliseconds:
the number of ms to sleep

returns:
function always returns 1

Appendix A How to Use SAT to Access SATA drives via SAS

What is SAT?

SAT (SCSI->ATA Translation) is a mechanism whereby ATA task register commands may be sent to a device which is seen by the operating system as a SCSI device. This is most often the case when SATA drives are connected to an add-in PCI bus type of SATA controller card. Even though the card is a SATA controller in most cases Windows will see the controller as if it were a SCSI HBA, and so will not allow you to issue ATA task register level commands to the connected devices.

Documentation on SAT can be found at the T10.org site http://www.t10.org/drafts.htm#sat3

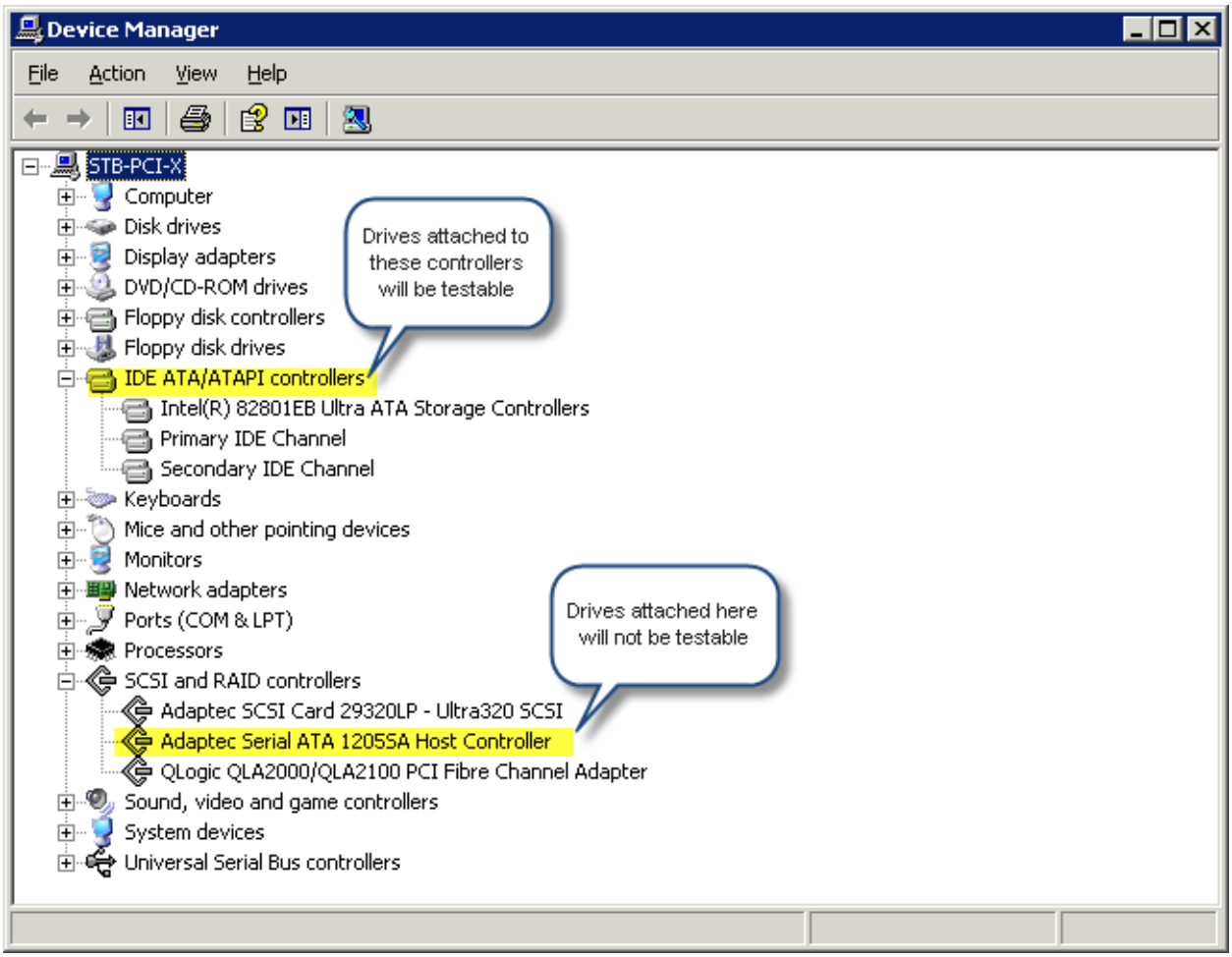
In short, SAT uses a 12 or 16 byte SCSI cdb which contains an embedded ATA task register command.

How do I know that I need to use SAT?

If your controller is seen by Windows as an ATA/IDE type of controller than you do not need to use SAT, you can simply issue normal ATA task register commands. How do you know what type of controller you have?

You can confirm how your operating system views your controller scheme by using Device Manager as shown below – note that the only drives that will be able to process actual ATA task register commands ***must*** be attached to a controller that Windows sees as an *IDE ATA/ATAPI controller*

In the example below, the second controller (Adaptec Serial ATA 1205A Host Controller) ***might*** be able to use SAT to send and ATA command to an attached drive.



Does my controller card support SAT?

The easiest way to determine of your controller supports SAT is to use the STB Suite SCSI User Defined CDB to try issuing a SAT command.

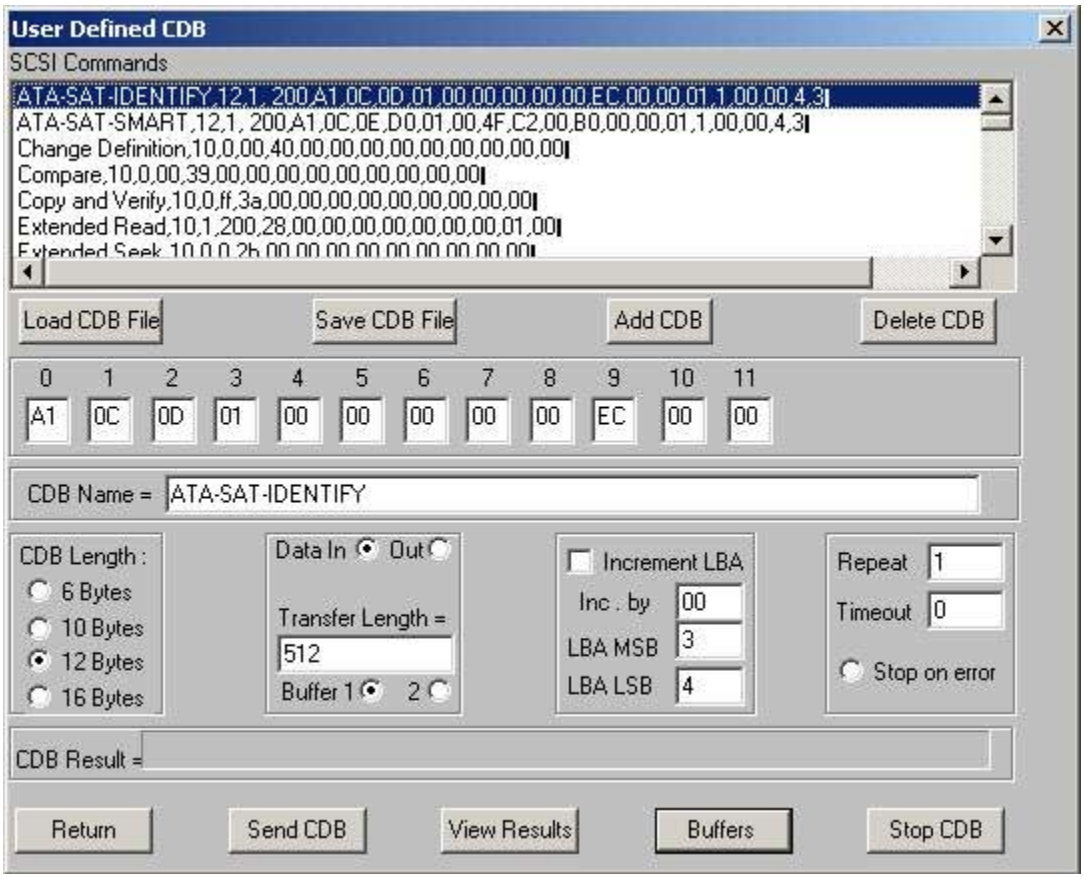
The 12-byte ATA Passthrough CDB we will use is defined as:

Byte\Bit	7	6	5	4	3	2	1	0
0	OPERATION CODE (A1h)							
1	MULTIPLE_COUNT			PROTOCOL				Reserved
2	OFF_LINE		CK_COND	Reserved	T_DIR	BYTE_BLOCK	T_LENGTH	
3	FEATURES (7:0)							
4	SECTOR_COUNT (7:0)							
5	LBA_LOW (7:0)							
6	LBA_MID (7:0)							
7	LBA_HIGH (7:0)							
8	DEVICE							
9	COMMAND							
10	Reserved							
11								

There are some obscure aspects of using this command – rather than going into detail about them right now we will instead simply describe how to issue a command which will tell us immediately if the controller supports SAT or not. We will discuss the details of each parameter of this command later on in this article.

A Simple test command

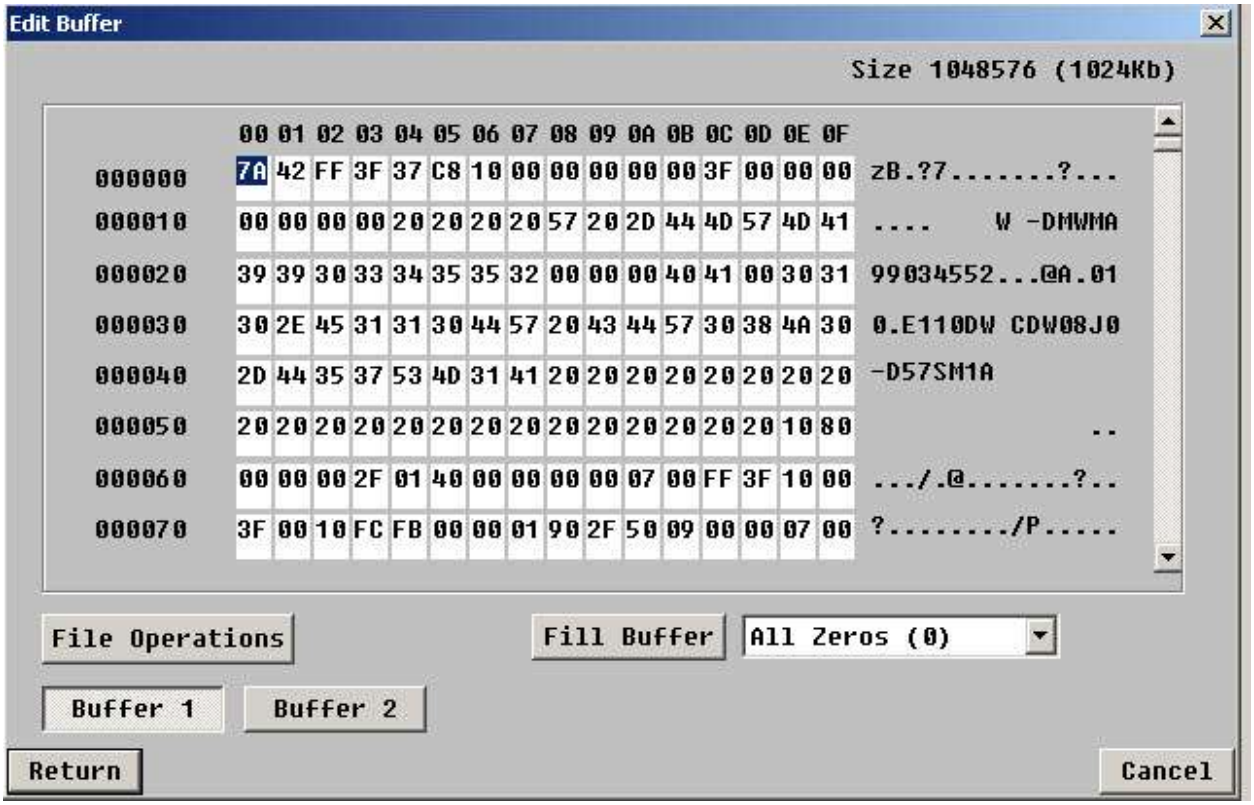
Here is a picture of the STB Suite User Defined CDB with an ATA Passthrough command defined which will issue an ATA IDENTIFY command to the attached drive



Select your target drive on the SATA controller that you hope will support SAT. Then right-click on the drive and choose ***User Defined CDBs***, then enter the command exactly as show above.

Click the ***Send CDB*** button to issue the command – the CDB Result field will tell if the command was successfully issued (congratulations, your controller implements SAT!) or if it failed (sorry, you won't be able to use SAT with this controller)

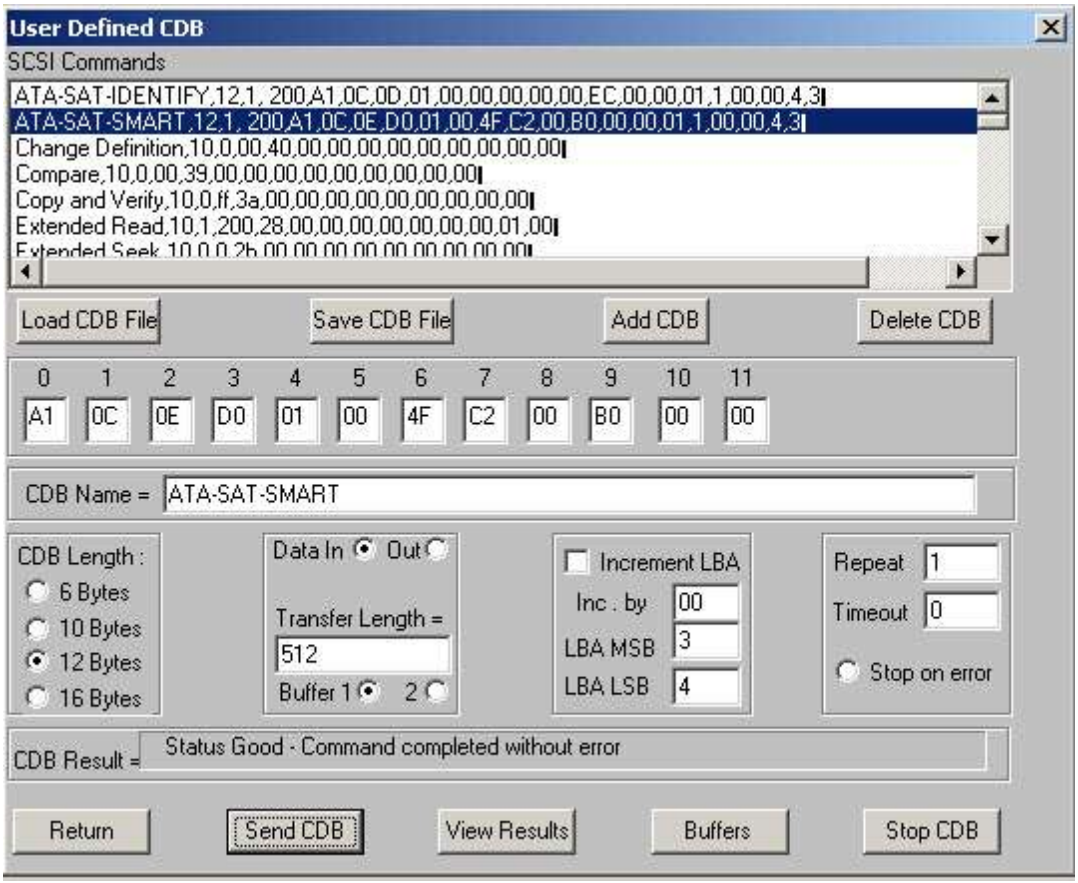
If the command completed successfully you can click the ***Buffer*** button so view the ATA IDENTIFY data returned from the drive.



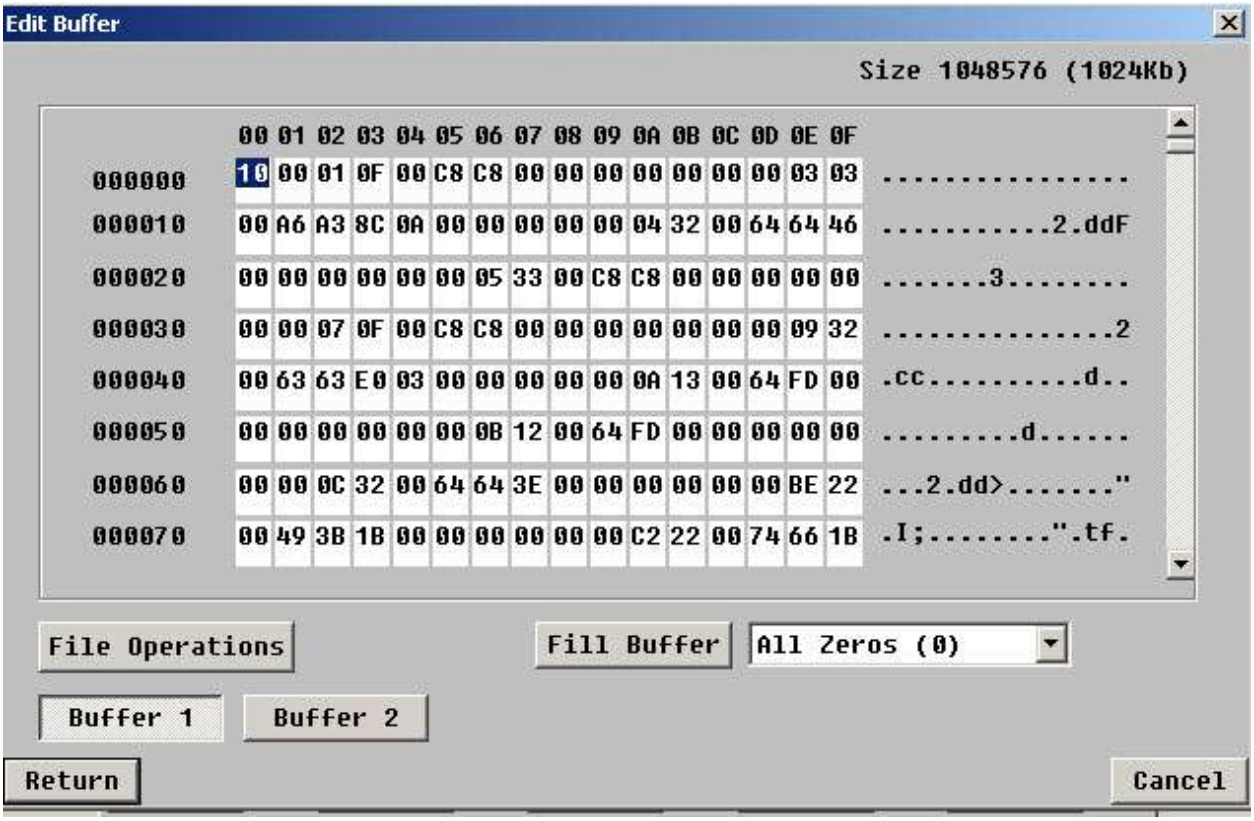
Note: ATA commands data byte-swapped.

Issuing the ATA SMART command

To retrieve SMART data from this drive define your cdb like this:



And as before, the data is available to view, edit, or save to a file



Details of defining an IDENTIFY command

Byte\Bit	7	6	5	4	3	2	1	0
0	OPERATION CODE (A1h)							
1	MULTIPLE_COUNT			PROTOCOL				Reserved
2	OFF_LINE	CK_COND		Reserved	T_DIR	BYTE_BLOCK	T_LENGTH	
3	FEATURES (7:0)							
4	SECTOR_COUNT (7:0)							
5	LBA_LOW (7:0)							
6	LBA_MID (7:0)							
7	LBA_HIGH (7:0)							
8	DEVICE							
9	COMMAND							
10	Reserved							
11								

The bytes of the command we constructed to issue an IDENTIFY command were:

0xA1, 0x0C, 0x0D, 00,00,00,00,00,0xEC,00,00

Byte 0:

Looking at the command description we see that the first byte is the SCSI op code.

Byte 1:

the next byte is used to specify the protocol, as defined in this table:

Code	Description
0	ATA hardware reset
1	SRST
2	Reserved
3	Non-data
4	PIO Data-In
5	PIO Data-Out
6	DMA
7	DMA Queued
8	Device Diagnostic
9	DEVICE RESET
10	UDMA Data In
11	UDMA Data Out
12	FPDMA[a]
13, 14	Reserved
15	Return Response Information

[a] See SATA-2.6.

In our command the 0x0C says we are requesting a DMA transfer. Note that you need to take care when defining this byte because of the offset caused by bit 0 being reserved.

Byte 2:

This byte is used to define how much data we are expecting, how the amount is specified, and which further bytes of the command will be used to specify. In the case of our IDENTIFY command we use 0x0D, which specifies that T_DIR = 1, BYTE_BLOCK=1, and T_LENGTH = 1. Referring to the SAT specification we see:

> If the T_DIR bit is set to zero, then the SATL shall transfer data from the application client to the ATA device. If the T_DIR bit is set to one, then the SATL shall transfer data from the ATA device to the application client. The SATL shall ignore the T_DIR bit if the T_LENGTH field is set to zero.

T_DIR = 1 says that the data direction will be receiving data from the drive.

> The BYTE_BLOCK (Byte/Block) bit specifies whether the transfer length in the location specified by the T_LENGTH field specifies the number of bytes to transfer or the number of blocks to transfer. If the value in the BYTE_BLOCK bit is set to zero, then the SATL shall transfer the number of bytes specified in the location specified by the T_LENGTH field. If the value in the BYTE_BLOCK bit is set to one the SATL shall transfer the number of blocks specified in the location specified by the T_LENGTH field. The SATL shall ignore the BYTE_BLOCK bit when the T_LENGTH field is set to zero.

BYTE_BLOCK=1 says that we are expecting to transfer one block (512 bytes) of data.

> The Transfer Length (T_LENGTH) field specifies where in the CDB the SATL shall locate the transfer length for the command (see table 98).

Table 98 — T_LENGTH field

Code	Description
00b	No data is transferred
01b	The transfer length is an unsigned integer specified in the FEATURES (7:0) field.
10b	The transfer length is an unsigned integer specified in the SECTOR_COUNT (7:0) field.
11b	The transfer length is an unsigned integer specified in the TPSIU (see 3.1.98).

Finally, T_LENGTH = 01 tells us that our transfer length is going to be specified by the integer placed in the ATA FEATURES byte field – which in this case is byte 3 of the CDB.

Byte 3:

-as we just said – because of our T_LENGTH setting we have specified that this byte will contain the number of blocks (because of the BYTE_BLOCK setting) of data that will be transferred, in the direction specified by T_DIR.

Byte 4:

– contains the ATA task register SECTOR COUNT data, in this case we are transferring 1 block so this must be set to 1.

Byte 5:

- contains the ATA task register LBA LOW byte – the IDENTIFY command needs this to be 0.

Byte 6:

- contains the ATA task register LBA MID byte – the IDENTIFY command needs this to be 0.

Byte 7:

- contains the ATA task register LBA HIGH byte – the IDENTIFY command needs this to be 0.

Byte 8:

- contains the ATA task register DEVICE byte – the IDENTIFY command needs this to be 0.

Byte 9:

- contains the ATA task register COMMAND byte – the IDENTIFY command is 0xEC.

Bytes 10 and 11 :

-are reserved and so set to 0.

Details of defining a SMART command

Refer to the T10 SAT specification documentation.

Note that we specify T_LENGTH = 10, which uses the SECTOR COUNT field (Byte 4) to specify our data length. Why did we need to do this, rather than use the FEATURES field (Byte 3) like we did for the IDENTIFY command?

We had to do this because the ATA SMART command needs to use the FEATURES field to define which type of SMART command we are sending – 0xD0 in this case.

Conclusion

SAT is a versatile if complicated method of issuing ATA commands to drives which are connected to controllers which Windows thinks are SCSI type. It is preferable rather than being forced to implementing controller vendor-unique pass through methods.

SAT is not universally supported or implemented. The controller that was used to illustrate this article is an LSI 8888 card, which happily does implement SAT and so allows access to “raw” ATA commands which would otherwise not be usable.

Using the above methods will allow you to access SATA functinality from within DMM, using a 12-byte User-defined SCSI CDB, or by writing a DMM-aware DTB application.

In fact, all of the canned SATA tests within DMM are implemented by using SAT.

Appendix B SCSI Sense Code/ASQ information

Disc Drive Sense Keys

Key	Sense Key Description
0h	**No Sense** - Indicates there is no specific Sense Key information to be reported for the disc drive. This would be the case for a successful command or when the ILI bit is one.
1h	**Recovered Error** - Indicates the last command completed successfully with some recovery action performed by the disc drive. When multiple recovered errors occur, the last error that occurred is reported by the additional sense bytes. **Note**: For some Mode settings, the last command may have terminated before completing.
2h	**Not Ready** - Indicates the logical unit addressed cannot be accessed. Operator intervention may be required to correct this condition.
3h	**Medium Error** - Indicates the command terminated with a nonrecovered error condition, probably caused by a flaw in the medium or an error in the recorded data.
4h	**Hardware Error** - Indicates the disc drive detected a nonrecoverable hardware failure while performing the command or during a self test. This includes SCSI interface parity error, controller failure or device failure.
5h	**Illegal Request** - Indicates an illegal parameter in the command descriptor block or in the additional parameters supplied as data for some commands (Format Unit, Mode Select, and so forth). If the disc drive detects an invalid parameter in the Command Descriptor Block, it shall terminate the command without altering the medium. If the disc drive detects an invalid parameter in the additional parameters supplied as data, the disc drive may have already altered the medium. This sense key may also indicate that an invalid IDENTIFY message was received. This could also indicate an attempt to write past the last logical block.

6h	**Unit Attention** - Indicates the disc drive may have been reset.
7h	**Data Protect** - Indicates that a command that reads or writes the medium was attempted on a block that is protected from this operation. The read or write operation is not performed.
9h	**Firmware Error** - Vendor specific sense key.
Bh	**Aborted Command** - Indicates the disc drive aborted the command. The initiator may be able to recover by trying the command again.
Ch	**Equal** - Indicates a SEARCH DATA command has satisfied an equal comparison.
Dh	**Volume Overflow** - Indicates a buffered peripheral device has reached the end of medium partition and data remains in the buffer that has not been written to the medium.
Eh	**Miscompare** - Indicates that the source data did not match the data read from the medium.

Sense Codes/ASQs

. . . . .
ASC ASCQ DT LPWROMAEBKVF Description
00h 00h DT LPWROMAEBKVF NO ADDITIONAL SENSE INFORMATION
00h 01h T FILEMARK DETECTED
00h 02h T END-OF-PARTITION/MEDIUM DETECTED
00h 03h T SETMARK DETECTED
00h 04h T BEGINNING-OF-PARTITION/MEDIUM DETECTED
00h 05h T L END-OF-DATA DETECTED
00h 06h DT LPWROMAEBKVF I/O PROCESS TERMINATED
00h 11h R AUDIO PLAY OPERATION IN PROGRESS
00h 12h R AUDIO PLAY OPERATION PAUSED
00h 13h R AUDIO PLAY OPERATION SUCCESSFULLY COMPLETED
00h 14h R AUDIO PLAY OPERATION STOPPED DUE TO ERROR
00h 15h R NO CURRENT AUDIO STATUS TO RETURN
00h 16h DT LPWROMAEBKVF OPERATION IN PROGRESS
00h 17h DT L WROMAEBKVF CLEANING REQUESTED
00h 18h T ERASE OPERATION IN PROGRESS
00h 19h T LOCATE OPERATION IN PROGRESS
00h 1Ah T REWIND OPERATION IN PROGRESS
00h 1Bh T SET CAPACITY OPERATION IN PROGRESS
00h 1Ch T VERIFY OPERATION IN PROGRESS
00h 1Dh DT B ATA PASS THROUGH INFORMATION AVAILABLE
01h 00h D W O BK NO INDEX/SECTOR SIGNAL
02h 00h D WROM BK NO SEEK COMPLETE
03h 00h DT L W O BK PERIPHERAL DEVICE WRITE FAULT
03h 01h T NO WRITE CURRENT
03h 02h T EXCESSIVE WRITE ERRORS
04h 00h DT LPWROMAEBKVF LOGICAL UNIT NOT READY, CAUSE NOT REPORTABLE

04h 01h DT LPWROMAEBKVF LOGICAL UNIT IS IN PROCESS OF BECOMING READY
04h 02h DT LPWROMAEBKVF LOGICAL UNIT NOT READY, INITIALIZING COMMAND REQUIRED
04h 03h DT LPWROMAEBKVF LOGICAL UNIT NOT READY, MANUAL INTERVENTION REQUIRED
04h 04h DT L RO B LOGICAL UNIT NOT READY, FORMAT IN PROGRESS
04h 05h DT W OMA BK LOGICAL UNIT NOT READY, REBUILD IN PROGRESS
04h 06h DT W OMA BK LOGICAL UNIT NOT READY, RECALCULATION IN PROGRESS
04h 07h DT LPWROMAEBKVF LOGICAL UNIT NOT READY, OPERATION IN PROGRESS
04h 08h R LOGICAL UNIT NOT READY, LONG WRITE IN PROGRESS
04h 09h DT LPWROMAEBKVF LOGICAL UNIT NOT READY, SELF-TEST IN PROGRESS
04h 0Ah DT LPWROMAEBKVF
LOGICAL UNIT NOT ACCESSIBLE, ASYMMETRIC ACCESS STATE
TRANSITION
04h 0Bh DT LPWROMAEBKVF LOGICAL UNIT NOT ACCESSIBLE, TARGET PORT IN STANDBY STATE
04h 0Ch DT LPWROMAEBKVF LOGICAL UNIT NOT ACCESSIBLE, TARGET PORT IN UNAVAILABLE STATE
04h 10h DT WROM B LOGICAL UNIT NOT READY, AUXILIARY MEMORY NOT ACCESSIBLE
04h 11h DT WROMAEB VF LOGICAL UNIT NOT READY, NOTIFY (ENABLE SPINUP) REQUIRED
04h 12h M V LOGICAL UNIT NOT READY, OFFLINE
05h 00h DT L WROMAEBKVF LOGICAL UNIT DOES NOT RESPOND TO SELECTION
06h 00h D WROM BK NO REFERENCE POSITION FOUND
07h 00h DT L WROM BK MULTIPLE PERIPHERAL DEVICES SELECTED
08h 00h DT L WROMAEBKVF LOGICAL UNIT COMMUNICATION FAILURE
08h 01h DT L WROMAEBKVF LOGICAL UNIT COMMUNICATION TIME-OUT
08h 02h DT L WROMAEBKVF LOGICAL UNIT COMMUNICATION PARITY ERROR
08h 03h DT ROM BK LOGICAL UNIT COMMUNICATION CRC ERROR (ULTRA-DMA/32)
08h 04h DT LPWRO K UNREACHABLE COPY TARGET
09h 00h DT WRO B TRACK FOLLOWING ERROR
09h 01h WRO K TRACKING SERVO FAILURE
09h 02h WRO K FOCUS SERVO FAILURE
09h 03h WRO SPINDLE SERVO FAILURE
09h 04h DT WRO B HEAD SELECT FAULT
0Ah 00h DT LPWROMAEBKVF ERROR LOG OVERFLOW
0Bh 00h DT LPWROMAEBKVF WARNING
0Bh 01h DT LPWROMAEBKVF WARNING - SPECIFIED TEMPERATURE EXCEEDED
0Bh 02h DT LPWROMAEBKVF WARNING - ENCLOSURE DEGRADED
0Bh 03h DT LPWROMAEBKVF WARNING - BACKGROUND SELF-TEST FAILED
0Bh 04h DT LPWROMAEBKVF WARNING - BACKGROUND PRE-SCAN DETECTED MEDIUM ERROR
0Bh 05h DT LPWROMAEBKVF WARNING - BACKGROUND MEDIUM SCAN DETECTED MEDIUM ERROR
0Ch 00h T R WRITE ERROR
0Ch 01h K WRITE ERROR - RECOVERED WITH AUTO REALLOCATION
0Ch 02h D W O BK WRITE ERROR - AUTO REALLOCATION FAILED
0Ch 03h D W O BK WRITE ERROR - RECOMMEND REASSIGNMENT
0Ch 04h DT W O B COMPRESSION CHECK MISCOMPARE ERROR
0Ch 05h DT W O B DATA EXPANSION OCCURRED DURING COMPRESSION
0Ch 06h DT W O B BLOCK NOT COMPRESSIBLE
0Ch 07h R WRITE ERROR - RECOVERY NEEDED
0Ch 08h R WRITE ERROR - RECOVERY FAILED
0Ch 09h R WRITE ERROR - LOSS OF STREAMING
0Ch 0Ah R WRITE ERROR - PADDING BLOCKS ADDED
0Ch 0Bh DT WROM B AUXILIARY MEMORY WRITE ERROR
0Ch 0Ch DT LPWROMAEBKVF WRITE ERROR - UNEXPECTED UNSOLICITED DATA
0Ch 0Dh DT LPWROMAEBKVF WRITE ERROR - NOT ENOUGH UNSOLICITED DATA
0Ch 0Fh R DEFECTS IN ERROR WINDOW
0Dh 00h DT LPWRO A K ERROR DETECTED BY THIRD PARTY TEMPORARY INITIATOR
0Dh 01h DT LPWRO A K THIRD PARTY DEVICE FAILURE
0Dh 02h DT LPWRO A K COPY TARGET DEVICE NOT REACHABLE
0Dh 03h DT LPWRO A K INCORRECT COPY TARGET DEVICE TYPE
0Dh 04h DT LPWRO A K COPY TARGET DEVICE DATA UNDERRUN
0Dh 05h DT LPWRO A K COPY TARGET DEVICE DATA OVERRUN
0Eh 00h DT PWROMAEBK F INVALID INFORMATION UNIT
0Eh 01h DT PWROMAEBK F INFORMATION UNIT TOO SHORT
0Eh 02h DT PWROMAEBK F INFORMATION UNIT TOO LONG
0Eh 03h DT P R MAEBK F INVALID FIELD IN COMMAND INFORMATION UNIT

0Fh 00h
10h 00h D W O BK ID CRC OR ECC ERROR
10h 01h DT W O LOGICAL BLOCK GUARD CHECK FAILED
10h 02h DT W O LOGICAL BLOCK APPLICATION TAG CHECK FAILED
10h 03h DT W O LOGICAL BLOCK REFERENCE TAG CHECK FAILED
11h 00h DT WRO BK UNRECOVERED READ ERROR
11h 01h DT WRO BK READ RETRIES EXHAUSTED
11h 02h DT WRO BK ERROR TOO LONG TO CORRECT
11h 03h DT W O BK MULTIPLE READ ERRORS
11h 04h D W O BK UNRECOVERED READ ERROR - AUTO REALLOCATE FAILED
11h 05h WRO B L-EC UNCORRECTABLE ERROR
11h 06h WRO B CIRC UNRECOVERED ERROR
11h 07h W O B DATA RE-SYNCHRONIZATION ERROR
11h 08h T INCOMPLETE BLOCK READ
11h 09h T NO GAP FOUND
11h 0Ah DT O BK MISCORRECTED ERROR
11h 0Bh D W O BK UNRECOVERED READ ERROR - RECOMMEND REASSIGNMENT
11h 0Ch D W O BK UNRECOVERED READ ERROR - RECOMMEND REWRITE THE DATA
11h 0Dh DT WRO B DE-COMPRESSION CRC ERROR
11h 0Eh DT WRO B CANNOT DECOMPRESS USING DECLARED ALGORITHM
11h 0Fh R ERROR READING UPC/EAN NUMBER
11h 10h R ERROR READING ISRC NUMBER
11h 11h R READ ERROR - LOSS OF STREAMING
11h 12h DT WROM B AUXILIARY MEMORY READ ERROR
11h 13h DT LPWROMAEBKVF READ ERROR - FAILED RETRANSMISSION REQUEST
11h 14h D READ ERROR - LBA MARKED BAD BY APPLICATION CLIENT
12h 00h D W O BK ADDRESS MARK NOT FOUND FOR ID FIELD
13h 00h D W O BK ADDRESS MARK NOT FOUND FOR DATA FIELD
14h 00h DT L WRO BK RECORDED ENTITY NOT FOUND
14h 01h DT WRO BK RECORD NOT FOUND
14h 02h T FILEMARK OR SETMARK NOT FOUND
14h 03h T END-OF-DATA NOT FOUND
14h 04h T BLOCK SEQUENCE ERROR
14h 05h DT W O BK RECORD NOT FOUND - RECOMMEND REASSIGNMENT
14h 06h DT W O BK RECORD NOT FOUND - DATA AUTO-REALLOCATED
14h 07h T LOCATE OPERATION FAILURE
15h 00h DT L WROM BK RANDOM POSITIONING ERROR
15h 01h DT L WROM BK MECHANICAL POSITIONING ERROR
15h 02h DT WRO BK POSITIONING ERROR DETECTED BY READ OF MEDIUM
16h 00h D W O BK DATA SYNCHRONIZATION MARK ERROR
16h 01h D W O BK DATA SYNC ERROR - DATA REWRITTEN
16h 02h D W O BK DATA SYNC ERROR - RECOMMEND REWRITE
16h 03h D W O BK DATA SYNC ERROR - DATA AUTO-REALLOCATED
16h 04h D W O BK DATA SYNC ERROR - RECOMMEND REASSIGNMENT
17h 00h DT WRO BK RECOVERED DATA WITH NO ERROR CORRECTION APPLIED
17h 01h DT WRO BK RECOVERED DATA WITH RETRIES
17h 02h DT WRO BK RECOVERED DATA WITH POSITIVE HEAD OFFSET
17h 03h DT WRO BK RECOVERED DATA WITH NEGATIVE HEAD OFFSET
17h 04h WRO B RECOVERED DATA WITH RETRIES AND/OR CIRC APPLIED
17h 05h D WRO BK RECOVERED DATA USING PREVIOUS SECTOR ID
17h 06h D W O BK RECOVERED DATA WITHOUT ECC - DATA AUTO-REALLOCATED
17h 07h D WRO BK RECOVERED DATA WITHOUT ECC - RECOMMEND REASSIGNMENT
17h 08h D WRO BK RECOVERED DATA WITHOUT ECC - RECOMMEND REWRITE
17h 09h D WRO BK RECOVERED DATA WITHOUT ECC - DATA REWRITTEN
18h 00h DT WRO BK RECOVERED DATA WITH ERROR CORRECTION APPLIED
18h 01h D WRO BK RECOVERED DATA WITH ERROR CORR. & RETRIES APPLIED
18h 02h D WRO BK RECOVERED DATA - DATA AUTO-REALLOCATED
18h 03h R RECOVERED DATA WITH CIRC
18h 04h R RECOVERED DATA WITH L-EC
18h 05h D WRO BK RECOVERED DATA - RECOMMEND REASSIGNMENT
18h 06h D WRO BK RECOVERED DATA - RECOMMEND REWRITE
18h 07h D W O BK RECOVERED DATA WITH ECC - DATA REWRITTEN

18h 08h R RECOVERED DATA WITH LINKING
19h 00h D O K DEFECT LIST ERROR
19h 01h D O K DEFECT LIST NOT AVAILABLE
19h 02h D O K DEFECT LIST ERROR IN PRIMARY LIST
19h 03h D O K DEFECT LIST ERROR IN GROWN LIST
1Ah 00h DT LPWROMAEBKVF PARAMETER LIST LENGTH ERROR
1Bh 00h DT LPWROMAEBKVF SYNCHRONOUS DATA TRANSFER ERROR
1Ch 00h D O BK DEFECT LIST NOT FOUND
1Ch 01h D O BK PRIMARY DEFECT LIST NOT FOUND
1Ch 02h D O BK GROWN DEFECT LIST NOT FOUND
1Dh 00h DT WRO BK MISCOMPARE DURING VERIFY OPERATION
1Eh 00h D W O BK RECOVERED ID WITH ECC CORRECTION
1Fh 00h D O K PARTIAL DEFECT LIST TRANSFER
20h 00h DT LPWROMAEBKVF INVALID COMMAND OPERATION CODE
20h 01h DT PWROMAEBK ACCESS DENIED - INITIATOR PENDING-ENROLLED
20h 02h DT PWROMAEBK ACCESS DENIED - NO ACCESS RIGHTS
20h 03h DT PWROMAEBK ACCESS DENIED - INVALID MGMT ID KEY
20h 04h T ILLEGAL COMMAND WHILE IN WRITE CAPABLE STATE
20h 05h T Obsolete
20h 06h T ILLEGAL COMMAND WHILE IN EXPLICIT ADDRESS MODE
20h 07h T ILLEGAL COMMAND WHILE IN IMPLICIT ADDRESS MODE
20h 08h DT PWROMAEBK ACCESS DENIED - ENROLLMENT CONFLICT
20h 09h DT PWROMAEBK ACCESS DENIED - INVALID LU IDENTIFIER
20h 0Ah DT PWROMAEBK ACCESS DENIED - INVALID PROXY TOKEN
20h 0Bh DT PWROMAEBK ACCESS DENIED - ACL LUN CONFLICT
21h 00h DT WROM BK LOGICAL BLOCK ADDRESS OUT OF RANGE
21h 01h DT WROM BK INVALID ELEMENT ADDRESS
21h 02h R INVALID ADDRESS FOR WRITE
21h 03h R INVALID WRITE CROSSING LAYER JUMP
22h 00h D ILLEGAL FUNCTION (USE 20 00, 24 00, OR 26 00)
23h 00h
24h 00h DT LPWROMAEBKVF INVALID FIELD IN CDB
24h 01h DT LPWROMAEBKVF CDB DECRYPTION ERROR
24h 02h T Obsolete
24h 03h T Obsolete
24h 04h F SECURITY AUDIT VALUE FROZEN
24h 05h F SECURITY WORKING KEY FROZEN
24h 06h F NONCE NOT UNIQUE
24h 07h F NONCE TIMESTAMP OUT OF RANGE
25h 00h DT LPWROMAEBKVF LOGICAL UNIT NOT SUPPORTED
26h 00h DT LPWROMAEBKVF INVALID FIELD IN PARAMETER LIST
26h 01h DT LPWROMAEBKVF PARAMETER NOT SUPPORTED
26h 02h DT LPWROMAEBKVF PARAMETER VALUE INVALID
26h 03h DT LPWROMAE K THRESHOLD PARAMETERS NOT SUPPORTED
26h 04h DT LPWROMAEBKVF INVALID RELEASE OF PERSISTENT RESERVATION
26h 05h DT LPWROMA BK DATA DECRYPTION ERROR
26h 06h DT LPWRO K TOO MANY TARGET DESCRIPTORS
26h 07h DT LPWRO K UNSUPPORTED TARGET DESCRIPTOR TYPE CODE
26h 08h DT LPWRO K TOO MANY SEGMENT DESCRIPTORS
26h 09h DT LPWRO K UNSUPPORTED SEGMENT DESCRIPTOR TYPE CODE
26h 0Ah DT LPWRO K UNEXPECTED INEXACT SEGMENT
26h 0Bh DT LPWRO K INLINE DATA LENGTH EXCEEDED
26h 0Ch DT LPWRO K INVALID OPERATION FOR COPY SOURCE OR DESTINATION
26h 0Dh DT LPWRO K COPY SEGMENT GRANULARITY VIOLATION
26h 0Eh DT PWROMAEBK INVALID PARAMETER WHILE PORT IS ENABLED
26h 0Fh F INVALID DATA-OUT BUFFER INTEGRITY CHECK VALUE
26h 10h T DATA DECRYPTION KEY FAIL LIMIT REACHED
26h 11h T INCOMPLETE KEY-ASSOCIATED DATA SET
26h 12h T VENDOR SPECIFIC KEY REFERENCE NOT FOUND
27h 00h DT WRO BK WRITE PROTECTED
27h 01h DT WRO BK HARDWARE WRITE PROTECTED
27h 02h DT WRO BK LOGICAL UNIT SOFTWARE WRITE PROTECTED

27h 03h T R ASSOCIATED WRITE PROTECT
27h 04h T R PERSISTENT WRITE PROTECT
27h 05h T R PERMANENT WRITE PROTECT
27h 06h R CONDITIONAL WRITE PROTECT
28h 00h DT LPWROMAEBKVF NOT READY TO READY CHANGE, MEDIUM MAY HAVE CHANGED
28h 01h DT WROM B IMPORT OR EXPORT ELEMENT ACCESSED
28h 02h R FORMAT-LAYER MAY HAVE CHANGED
29h 00h DT LPWROMAEBKVF POWER ON, RESET, OR BUS DEVICE RESET OCCURRED
29h 01h DT LPWROMAEBKVF POWER ON OCCURRED
29h 02h DT LPWROMAEBKVF SCSI BUS RESET OCCURRED
29h 03h DT LPWROMAEBKVF BUS DEVICE RESET FUNCTION OCCURRED
29h 04h DT LPWROMAEBKVF DEVICE INTERNAL RESET
29h 05h DT LPWROMAEBKVF TRANSCEIVER MODE CHANGED TO SINGLE-ENDED
29h 06h DT LPWROMAEBKVF TRANSCEIVER MODE CHANGED TO LVD
29h 07h DT LPWROMAEBKVF I_T NEXUS LOSS OCCURRED
2Ah 00h DT L WROMAEBKVF PARAMETERS CHANGED
2Ah 01h DT L WROMAEBKVF MODE PARAMETERS CHANGED
2Ah 02h DT L WROMAE K LOG PARAMETERS CHANGED
2Ah 03h DT LPWROMAE K RESERVATIONS PREEMPTED
2Ah 04h DT LPWROMAE RESERVATIONS RELEASED
2Ah 05h DT LPWROMAE REGISTRATIONS PREEMPTED
2Ah 06h DT LPWROMAEBKVF ASYMMETRIC ACCESS STATE CHANGED
2Ah 07h DT LPWROMAEBKVF IMPLICIT ASYMMETRIC ACCESS STATE TRANSITION FAILED
2Ah 08h DT WROMAEBKVF PRIORITY CHANGED
2Ah 09h D CAPACITY DATA HAS CHANGED
2Ah 10h DT M E V TIMESTAMP CHANGED
2Ah 11h T DATA ENCRYPTION PARAMETERS CHANGED BY ANOTHER I_T NEXUS
2Ah 12h T DATA ENCRYPTION PARAMETERS CHANGED BY VENDOR SPECIFIC EVENT
2Ah 13h T DATA ENCRYPTION KEY INSTANCE COUNTER HAS CHANGED
2Bh 00h DT LPWRO K COPY CANNOT EXECUTE SINCE HOST CANNOT DISCONNECT
2Ch 00h DT LPWROMAEBKVF COMMAND SEQUENCE ERROR
2Ch 01h TOO MANY WINDOWS SPECIFIED
2Ch 02h INVALID COMBINATION OF WINDOWS SPECIFIED
2Ch 03h R CURRENT PROGRAM AREA IS NOT EMPTY
2Ch 04h R CURRENT PROGRAM AREA IS EMPTY
2Ch 05h B ILLEGAL POWER CONDITION REQUEST
2Ch 06h R PERSISTENT PREVENT CONFLICT
2Ch 07h DT LPWROMAEBKVF PREVIOUS BUSY STATUS
2Ch 08h DT LPWROMAEBKVF PREVIOUS TASK SET FULL STATUS
2Ch 09h DT LPWROM EBKVF PREVIOUS RESERVATION CONFLICT STATUS
2Ch 0Ah F PARTITION OR COLLECTION CONTAINS USER OBJECTS
2Ch 0Bh T NOT RESERVED
2Dh 00h T OVERWRITE ERROR ON UPDATE IN PLACE
2Eh 00h R INSUFFICIENT TIME FOR OPERATION
2Fh 00h DT LPWROMAEBKVF COMMANDS CLEARED BY ANOTHER INITIATOR
2Fh 01h D COMMANDS CLEARED BY POWER LOSS NOTIFICATION
2Fh 02h DT LPWROMAEBKVF COMMANDS CLEARED BY DEVICE SERVER
30h 00h DT WROM BK INCOMPATIBLE MEDIUM INSTALLED
30h 01h DT WRO BK CANNOT READ MEDIUM - UNKNOWN FORMAT
30h 02h DT WRO BK CANNOT READ MEDIUM - INCOMPATIBLE FORMAT
30h 03h DT R K CLEANING CARTRIDGE INSTALLED
30h 04h DT WRO BK CANNOT WRITE MEDIUM - UNKNOWN FORMAT
30h 05h DT WRO BK CANNOT WRITE MEDIUM - INCOMPATIBLE FORMAT
30h 06h DT WRO B CANNOT FORMAT MEDIUM - INCOMPATIBLE MEDIUM
30h 07h DT L WROMAEBKVF CLEANING FAILURE
30h 08h R CANNOT WRITE - APPLICATION CODE MISMATCH
30h 09h R CURRENT SESSION NOT FIXATED FOR APPEND
30h 0Ah DT WROMAEBK CLEANING REQUEST REJECTED
30h 0Ch T WORM MEDIUM - OVERWRITE ATTEMPTED
30h 0Dh T WORM MEDIUM - INTEGRITY CHECK
30h 10h R MEDIUM NOT FORMATTED
31h 00h DT WRO BK MEDIUM FORMAT CORRUPTED

31h 01h D L RO B FORMAT COMMAND FAILED
31h 02h R ZONED FORMATTING FAILED DUE TO SPARE LINKING
32h 00h D W O BK NO DEFECT SPARE LOCATION AVAILABLE
32h 01h D W O BK DEFECT LIST UPDATE FAILURE
33h 00h T TAPE LENGTH ERROR
34h 00h DT LPWROMAEBKVF ENCLOSURE FAILURE
35h 00h DT LPWROMAEBKVF ENCLOSURE SERVICES FAILURE
35h 01h DT LPWROMAEBKVF UNSUPPORTED ENCLOSURE FUNCTION
35h 02h DT LPWROMAEBKVF ENCLOSURE SERVICES UNAVAILABLE
35h 03h DT LPWROMAEBKVF ENCLOSURE SERVICES TRANSFER FAILURE
35h 04h DT LPWROMAEBKVF ENCLOSURE SERVICES TRANSFER REFUSED
35h 05h DT L WROMAEBKVF ENCLOSURE SERVICES CHECKSUM ERROR
36h 00h L RIBBON, INK, OR TONER FAILURE
37h 00h DT L WROMAEBKVF ROUNDED PARAMETER
38h 00h B EVENT STATUS NOTIFICATION
38h 02h B ESN - POWER MANAGEMENT CLASS EVENT
38h 04h B ESN - MEDIA CLASS EVENT
38h 06h B ESN - DEVICE BUSY CLASS EVENT
39h 00h DT L WROMAE K SAVING PARAMETERS NOT SUPPORTED
3Ah 00h DT L WROM BK MEDIUM NOT PRESENT
3Ah 01h DT WROM BK MEDIUM NOT PRESENT - TRAY CLOSED
3Ah 02h DT WROM BK MEDIUM NOT PRESENT - TRAY OPEN
3Ah 03h DT WROM B MEDIUM NOT PRESENT - LOADABLE
3Ah 04h DT WROM B MEDIUM NOT PRESENT - MEDIUM AUXILIARY MEMORY ACCESSIBLE
3Bh 00h T L SEQUENTIAL POSITIONING ERROR
3Bh 01h T TAPE POSITION ERROR AT BEGINNING-OF-MEDIUM
3Bh 02h T TAPE POSITION ERROR AT END-OF-MEDIUM
3Bh 03h L TAPE OR ELECTRONIC VERTICAL FORMS UNIT NOT READY
3Bh 04h L SLEW FAILURE
3Bh 05h L PAPER JAM
3Bh 06h L FAILED TO SENSE TOP-OF-FORM
3Bh 07h L FAILED TO SENSE BOTTOM-OF-FORM
3Bh 08h T REPOSITION ERROR
3Bh 09h READ PAST END OF MEDIUM
3Bh 0Ah READ PAST BEGINNING OF MEDIUM
3Bh 0Bh POSITION PAST END OF MEDIUM
3Bh 0Ch T POSITION PAST BEGINNING OF MEDIUM
3Bh 0Dh DT WROM BK MEDIUM DESTINATION ELEMENT FULL
3Bh 0Eh DT WROM BK MEDIUM SOURCE ELEMENT EMPTY
3Bh 0Fh R END OF MEDIUM REACHED
3Bh 11h DT WROM BK MEDIUM MAGAZINE NOT ACCESSIBLE
3Bh 12h DT WROM BK MEDIUM MAGAZINE REMOVED
3Bh 13h DT WROM BK MEDIUM MAGAZINE INSERTED
3Bh 14h DT WROM BK MEDIUM MAGAZINE LOCKED
3Bh 15h DT WROM BK MEDIUM MAGAZINE UNLOCKED
3Bh 16h R MECHANICAL POSITIONING OR CHANGER ERROR
3Bh 17h F READ PAST END OF USER OBJECT
3Ch 00h
3Dh 00h DT LPWROMAE K INVALID BITS IN IDENTIFY MESSAGE
3Eh 00h DT LPWROMAEBKVF LOGICAL UNIT HAS NOT SELF-CONFIGURED YET
3Eh 01h DT LPWROMAEBKVF LOGICAL UNIT FAILURE
3Eh 02h DT LPWROMAEBKVF TIMEOUT ON LOGICAL UNIT
3Eh 03h DT LPWROMAEBKVF LOGICAL UNIT FAILED SELF-TEST
3Eh 04h DT LPWROMAEBKVF LOGICAL UNIT UNABLE TO UPDATE SELF-TEST LOG
3Fh 00h DT LPWROMAEBKVF TARGET OPERATING CONDITIONS HAVE CHANGED
3Fh 01h DT LPWROMAEBKVF MICROCODE HAS BEEN CHANGED
3Fh 02h DT LPWROM BK CHANGED OPERATING DEFINITION
3Fh 03h DT LPWROMAEBKVF INQUIRY DATA HAS CHANGED
3Fh 04h DT WROMAEBK COMPONENT DEVICE ATTACHED
3Fh 05h DT WROMAEBK DEVICE IDENTIFIER CHANGED
3Fh 06h DT WROMAEB REDUNDANCY GROUP CREATED OR MODIFIED
3Fh 07h DT WROMAEB REDUNDANCY GROUP DELETED

3Fh 08h DT WROMAEB SPARE CREATED OR MODIFIED
3Fh 09h DT WROMAEB SPARE DELETED
3Fh 0Ah DT WROMAEBK VOLUME SET CREATED OR MODIFIED
3Fh 0Bh DT WROMAEBK VOLUME SET DELETED
3Fh 0Ch DT WROMAEBK VOLUME SET DEASSIGNED
3Fh 0Dh DT WROMAEBK VOLUME SET REASSIGNED
3Fh 0Eh DT LPWROMAE REPORTED LUNS DATA HAS CHANGED
3Fh 0Fh DT LPWROMAEBKVF ECHO BUFFER OVERWRITTEN
3Fh 10h DT WROM B MEDIUM LOADABLE
3Fh 11h DT WROM B MEDIUM AUXILIARY MEMORY ACCESSIBLE
3Fh 12h DT LPWR MAEBK F iSCSI IP ADDRESS ADDED
3Fh 13h DT LPWR MAEBK F iSCSI IP ADDRESS REMOVED
3Fh 14h DT LPWR MAEBK F iSCSI IP ADDRESS CHANGED
40h 00h D RAM FAILURE (SHOULD USE 40 NN)
40h NNh DT LPWROMAEBKVF DIAGNOSTIC FAILURE ON COMPONENT NN (80h-FFh)
41h 00h D DATA PATH FAILURE (SHOULD USE 40 NN)
42h 00h D POWER-ON OR SELF-TEST FAILURE (SHOULD USE 40 NN)
43h 00h DT LPWROMAEBKVF MESSAGE ERROR
44h 00h DT LPWROMAEBKVF INTERNAL TARGET FAILURE
44h 71h DT B ATA DEVICE FAILED SET FEATURES
45h 00h DT LPWROMAEBKVF SELECT OR RESELECT FAILURE
46h 00h DT LPWROM BK UNSUCCESSFUL SOFT RESET
47h 00h DT LPWROMAEBKVF SCSI PARITY ERROR
47h 01h DT LPWROMAEBKVF DATA PHASE CRC ERROR DETECTED
47h 02h DT LPWROMAEBKVF SCSI PARITY ERROR DETECTED DURING ST DATA PHASE
47h 03h DT LPWROMAEBKVF INFORMATION UNIT iuCRC ERROR DETECTED
47h 04h DT LPWROMAEBKVF ASYNCHRONOUS INFORMATION PROTECTION ERROR DETECTED
47h 05h DT LPWROMAEBKVF PROTOCOL SERVICE CRC ERROR
47h 06h DT MAEBKVF PHY TEST FUNCTION IN PROGRESS
47h 7Fh DT PWROMAEBK SOME COMMANDS CLEARED BY ISCSI PROTOCOL EVENT
48h 00h DT LPWROMAEBKVF INITIATOR DETECTED ERROR MESSAGE RECEIVED
49h 00h DT LPWROMAEBKVF INVALID MESSAGE ERROR
4Ah 00h DT LPWROMAEBKVF COMMAND PHASE ERROR
4Bh 00h DT LPWROMAEBKVF DATA PHASE ERROR
4Bh 01h DT PWROMAEBK INVALID TARGET PORT TRANSFER TAG RECEIVED
4Bh 02h DT PWROMAEBK TOO MUCH WRITE DATA
4Bh 03h DT PWROMAEBK ACK/NAK TIMEOUT
4Bh 04h DT PWROMAEBK NAK RECEIVED
4Bh 05h DT PWROMAEBK DATA OFFSET ERROR
4Bh 06h DT PWROMAEBK INITIATOR RESPONSE TIMEOUT
4Ch 00h DT LPWROMAEBKVF LOGICAL UNIT FAILED SELF-CONFIGURATION
4Dh NNh DT LPWROMAEBKVF TAGGED OVERLAPPED COMMANDS (NN = TASK TAG)
4Eh 00h DT LPWROMAEBKVF OVERLAPPED COMMANDS ATTEMPTED
4Fh 00h
50h 00h T WRITE APPEND ERROR
50h 01h T WRITE APPEND POSITION ERROR
50h 02h T POSITION ERROR RELATED TO TIMING
51h 00h T RO ERASE FAILURE
51h 01h R ERASE FAILURE - INCOMPLETE ERASE OPERATION DETECTED
52h 00h T CARTRIDGE FAULT
53h 00h DT L WROM BK MEDIA LOAD OR EJECT FAILED
53h 01h T UNLOAD TAPE FAILURE
53h 02h DT WROM BK MEDIUM REMOVAL PREVENTED
53h 03h M MEDIUM REMOVAL PREVENTED BY DATA TRANSFER ELEMENT
53h 04h T MEDIUM THREAD OR UNTHREAD FAILURE
54h 00h P SCSI TO HOST SYSTEM INTERFACE FAILURE
55h 00h P SYSTEM RESOURCE FAILURE
55h 01h D O BK SYSTEM BUFFER FULL
55h 02h DT LPWROMAE K INSUFFICIENT RESERVATION RESOURCES
55h 03h DT LPWROMAE K INSUFFICIENT RESOURCES
55h 04h DT LPWROMAE K INSUFFICIENT REGISTRATION RESOURCES
55h 05h DT PWROMAEBK INSUFFICIENT ACCESS CONTROL RESOURCES

55h 06h DT WROM B AUXILIARY MEMORY OUT OF SPACE
55h 07h F QUOTA ERROR
55h 08h T MAXIMUM NUMBER OF SUPPLEMENTAL DECRYPTION KEYS EXCEEDED
56h 00h
57h 00h R UNABLE TO RECOVER TABLE-OF-CONTENTS
58h 00h O GENERATION DOES NOT EXIST
59h 00h O UPDATED BLOCK READ
5Ah 00h DT LPWROM BK OPERATOR REQUEST OR STATE CHANGE INPUT
5Ah 01h DT WROM BK OPERATOR MEDIUM REMOVAL REQUEST
5Ah 02h DT WRO A BK OPERATOR SELECTED WRITE PROTECT
5Ah 03h DT WRO A BK OPERATOR SELECTED WRITE PERMIT
5Bh 00h DT LPWROM K LOG EXCEPTION
5Bh 01h DT LPWROM K THRESHOLD CONDITION MET
5Bh 02h DT LPWROM K LOG COUNTER AT MAXIMUM
5Bh 03h DT LPWROM K LOG LIST CODES EXHAUSTED
5Ch 00h D O RPL STATUS CHANGE
5Ch 01h D O SPINDLES SYNCHRONIZED
5Ch 02h D O SPINDLES NOT SYNCHRONIZED
5Dh 00h DTLPWROMAEBKVF FAILURE PREDICTION THRESHOLD EXCEEDED
5Dh 01h R B MEDIA FAILURE PREDICTION THRESHOLD EXCEEDED
5Dh 02h R LOGICAL UNIT FAILURE PREDICTION THRESHOLD EXCEEDED
5Dh 03h R SPARE AREA EXHAUSTION PREDICTION THRESHOLD EXCEEDED
5Dh 10h D B HARDWARE IMPENDING FAILURE GENERAL HARD DRIVE FAILURE
5Dh 11h D B HARDWARE IMPENDING FAILURE DRIVE ERROR RATE TOO HIGH
5Dh 12h D B HARDWARE IMPENDING FAILURE DATA ERROR RATE TOO HIGH
5Dh 13h D B HARDWARE IMPENDING FAILURE SEEK ERROR RATE TOO HIGH
5Dh 14h D B HARDWARE IMPENDING FAILURE TOO MANY BLOCK REASSIGNS
5Dh 15h D B HARDWARE IMPENDING FAILURE ACCESS TIMES TOO HIGH
5Dh 16h D B HARDWARE IMPENDING FAILURE START UNIT TIMES TOO HIGH
5Dh 17h D B HARDWARE IMPENDING FAILURE CHANNEL PARAMETRICS
5Dh 18h D B HARDWARE IMPENDING FAILURE CONTROLLER DETECTED
5Dh 19h D B HARDWARE IMPENDING FAILURE THROUGHPUT PERFORMANCE
5Dh 1Ah D B HARDWARE IMPENDING FAILURE SEEK TIME PERFORMANCE
5Dh 1Bh D B HARDWARE IMPENDING FAILURE SPIN-UP RETRY COUNT
5Dh 1Ch D B HARDWARE IMPENDING FAILURE DRIVE CALIBRATION RETRY COUNT
5Dh 20h D B CONTROLLER IMPENDING FAILURE GENERAL HARD DRIVE FAILURE
5Dh 21h D B CONTROLLER IMPENDING FAILURE DRIVE ERROR RATE TOO HIGH
5Dh 22h D B CONTROLLER IMPENDING FAILURE DATA ERROR RATE TOO HIGH
5Dh 23h D B CONTROLLER IMPENDING FAILURE SEEK ERROR RATE TOO HIGH
5Dh 24h D B CONTROLLER IMPENDING FAILURE TOO MANY BLOCK REASSIGNS
5Dh 25h D B CONTROLLER IMPENDING FAILURE ACCESS TIMES TOO HIGH
5Dh 26h D B CONTROLLER IMPENDING FAILURE START UNIT TIMES TOO HIGH
5Dh 27h D B CONTROLLER IMPENDING FAILURE CHANNEL PARAMETRICS
5Dh 28h D B CONTROLLER IMPENDING FAILURE CONTROLLER DETECTED
5Dh 29h D B CONTROLLER IMPENDING FAILURE THROUGHPUT PERFORMANCE
5Dh 2Ah D B CONTROLLER IMPENDING FAILURE SEEK TIME PERFORMANCE
5Dh 2Bh D B CONTROLLER IMPENDING FAILURE SPIN-UP RETRY COUNT
5Dh 2Ch D B CONTROLLER IMPENDING FAILURE DRIVE CALIBRATION RETRY COUNT
5Dh 30h D B DATA CHANNEL IMPENDING FAILURE GENERAL HARD DRIVE FAILURE
5Dh 31h D B DATA CHANNEL IMPENDING FAILURE DRIVE ERROR RATE TOO HIGH
5Dh 32h D B DATA CHANNEL IMPENDING FAILURE DATA ERROR RATE TOO HIGH
5Dh 33h D B DATA CHANNEL IMPENDING FAILURE SEEK ERROR RATE TOO HIGH
5Dh 34h D B DATA CHANNEL IMPENDING FAILURE TOO MANY BLOCK REASSIGNS
5Dh 35h D B DATA CHANNEL IMPENDING FAILURE ACCESS TIMES TOO HIGH
5Dh 36h D B DATA CHANNEL IMPENDING FAILURE START UNIT TIMES TOO HIGH
5Dh 37h D B DATA CHANNEL IMPENDING FAILURE CHANNEL PARAMETRICS
5Dh 38h D B DATA CHANNEL IMPENDING FAILURE CONTROLLER DETECTED
5Dh 39h D B DATA CHANNEL IMPENDING FAILURE THROUGHPUT PERFORMANCE
5Dh 3Ah D B DATA CHANNEL IMPENDING FAILURE SEEK TIME PERFORMANCE
5Dh 3Bh D B DATA CHANNEL IMPENDING FAILURE SPIN-UP RETRY COUNT
5Dh 3Ch D B DATA CHANNEL IMPENDING FAILURE DRIVE CALIBRATION RETRY COUNT
5Dh 40h D B SERVO IMPENDING FAILURE GENERAL HARD DRIVE FAILURE

5Dh 41h D B SERVO IMPENDING FAILURE DRIVE ERROR RATE TOO HIGH
5Dh 42h D B SERVO IMPENDING FAILURE DATA ERROR RATE TOO HIGH
5Dh 43h D B SERVO IMPENDING FAILURE SEEK ERROR RATE TOO HIGH
5Dh 44h D B SERVO IMPENDING FAILURE TOO MANY BLOCK REASSIGNS
5Dh 45h D B SERVO IMPENDING FAILURE ACCESS TIMES TOO HIGH
5Dh 46h D B SERVO IMPENDING FAILURE START UNIT TIMES TOO HIGH
5Dh 47h D B SERVO IMPENDING FAILURE CHANNEL PARAMETRICS
5Dh 48h D B SERVO IMPENDING FAILURE CONTROLLER DETECTED
5Dh 49h D B SERVO IMPENDING FAILURE THROUGHPUT PERFORMANCE
5Dh 4Ah D B SERVO IMPENDING FAILURE SEEK TIME PERFORMANCE
5Dh 4Bh D B SERVO IMPENDING FAILURE SPIN-UP RETRY COUNT
5Dh 4Ch D B SERVO IMPENDING FAILURE DRIVE CALIBRATION RETRY COUNT
5Dh 50h D B SPINDLE IMPENDING FAILURE GENERAL HARD DRIVE FAILURE
5Dh 51h D B SPINDLE IMPENDING FAILURE DRIVE ERROR RATE TOO HIGH
5Dh 52h D B SPINDLE IMPENDING FAILURE DATA ERROR RATE TOO HIGH
5Dh 53h D B SPINDLE IMPENDING FAILURE SEEK ERROR RATE TOO HIGH
5Dh 54h D B SPINDLE IMPENDING FAILURE TOO MANY BLOCK REASSIGNS
5Dh 55h D B SPINDLE IMPENDING FAILURE ACCESS TIMES TOO HIGH
5Dh 56h D B SPINDLE IMPENDING FAILURE START UNIT TIMES TOO HIGH
5Dh 57h D B SPINDLE IMPENDING FAILURE CHANNEL PARAMETRICS
5Dh 58h D B SPINDLE IMPENDING FAILURE CONTROLLER DETECTED
5Dh 59h D B SPINDLE IMPENDING FAILURE THROUGHPUT PERFORMANCE
5Dh 5Ah D B SPINDLE IMPENDING FAILURE SEEK TIME PERFORMANCE
5Dh 5Bh D B SPINDLE IMPENDING FAILURE SPIN-UP RETRY COUNT
5Dh 5Ch D B SPINDLE IMPENDING FAILURE DRIVE CALIBRATION RETRY COUNT
5Dh 60h D B FIRMWARE IMPENDING FAILURE GENERAL HARD DRIVE FAILURE
5Dh 61h D B FIRMWARE IMPENDING FAILURE DRIVE ERROR RATE TOO HIGH
5Dh 62h D B FIRMWARE IMPENDING FAILURE DATA ERROR RATE TOO HIGH
5Dh 63h D B FIRMWARE IMPENDING FAILURE SEEK ERROR RATE TOO HIGH
5Dh 64h D B FIRMWARE IMPENDING FAILURE TOO MANY BLOCK REASSIGNS
5Dh 65h D B FIRMWARE IMPENDING FAILURE ACCESS TIMES TOO HIGH
5Dh 66h D B FIRMWARE IMPENDING FAILURE START UNIT TIMES TOO HIGH
5Dh 67h D B FIRMWARE IMPENDING FAILURE CHANNEL PARAMETRICS
5Dh 68h D B FIRMWARE IMPENDING FAILURE CONTROLLER DETECTED
5Dh 69h D B FIRMWARE IMPENDING FAILURE THROUGHPUT PERFORMANCE
5Dh 6Ah D B FIRMWARE IMPENDING FAILURE SEEK TIME PERFORMANCE
5Dh 6Bh D B FIRMWARE IMPENDING FAILURE SPIN-UP RETRY COUNT
5Dh 6Ch D B FIRMWARE IMPENDING FAILURE DRIVE CALIBRATION RETRY COUNT
5Dh FFh DT LPWROMAEBKVF FAILURE PREDICTION THRESHOLD EXCEEDED (FALSE)
5Eh 00h DT LPWRO A K LOW POWER CONDITION ON
5Eh 01h DT LPWRO A K IDLE CONDITION ACTIVATED BY TIMER
5Eh 02h DT LPWRO A K STANDBY CONDITION ACTIVATED BY TIMER
5Eh 03h DT LPWRO A K IDLE CONDITION ACTIVATED BY COMMAND
5Eh 04h DT LPWRO A K STANDBY CONDITION ACTIVATED BY COMMAND
5Eh 41h B POWER STATE CHANGE TO ACTIVE
5Eh 42h B POWER STATE CHANGE TO IDLE
5Eh 43h B POWER STATE CHANGE TO STANDBY
5Eh 45h B POWER STATE CHANGE TO SLEEP
5Eh 47h BK POWER STATE CHANGE TO DEVICE CONTROL
5Fh 00h
60h 00h LAMP FAILURE
61h 00h VIDEO ACQUISITION ERROR
61h 01h UNABLE TO ACQUIRE VIDEO
61h 02h OUT OF FOCUS
62h 00h SCAN HEAD POSITIONING ERROR
63h 00h R END OF USER AREA ENCOUNTERED ON THIS TRACK
63h 01h R PACKET DOES NOT FIT IN AVAILABLE SPACE
64h 00h R ILLEGAL MODE FOR THIS TRACK
64h 01h R INVALID PACKET SIZE
65h 00h DT LPWROMAEBKVF VOLTAGE FAULT
66h 00h AUTOMATIC DOCUMENT FEEDER COVER UP
66h 01h AUTOMATIC DOCUMENT FEEDER LIFT UP

66h 02h DOCUMENT JAM IN AUTOMATIC DOCUMENT FEEDER
66h 03h DOCUMENT MISS FEED AUTOMATIC IN DOCUMENT FEEDER
67h 00h A CONFIGURATION FAILURE
67h 01h A CONFIGURATION OF INCAPABLE LOGICAL UNITS FAILED
67h 02h A ADD LOGICAL UNIT FAILED
67h 03h A MODIFICATION OF LOGICAL UNIT FAILED
67h 04h A EXCHANGE OF LOGICAL UNIT FAILED
67h 05h A REMOVE OF LOGICAL UNIT FAILED
67h 06h A ATTACHMENT OF LOGICAL UNIT FAILED
67h 07h A CREATION OF LOGICAL UNIT FAILED
67h 08h A ASSIGN FAILURE OCCURRED
67h 09h A MULTIPLY ASSIGNED LOGICAL UNIT
67h 0Ah DT LPWROMAEBKVF SET TARGET PORT GROUPS COMMAND FAILED
67h 0Bh DT B ATA DEVICE FEATURE NOT ENABLED
68h 00h A LOGICAL UNIT NOT CONFIGURED
69h 00h A DATA LOSS ON LOGICAL UNIT
69h 01h A MULTIPLE LOGICAL UNIT FAILURES
69h 02h A PARITY/DATA MISMATCH
6Ah 00h A INFORMATIONAL, REFER TO LOG
6Bh 00h A STATE CHANGE HAS OCCURRED
6Bh 01h A REDUNDANCY LEVEL GOT BETTER
6Bh 02h A REDUNDANCY LEVEL GOT WORSE
6Ch 00h A REBUILD FAILURE OCCURRED
6Dh 00h A RECALCULATE FAILURE OCCURRED
6Eh 00h A COMMAND TO LOGICAL UNIT FAILED
6Fh 00h R COPY PROTECTION KEY EXCHANGE FAILURE - AUTHENTICATION FAILURE
6Fh 01h R COPY PROTECTION KEY EXCHANGE FAILURE - KEY NOT PRESENT
6Fh 02h R COPY PROTECTION KEY EXCHANGE FAILURE - KEY NOT ESTABLISHED
6Fh 03h R READ OF SCRAMBLED SECTOR WITHOUT AUTHENTICATION
6Fh 04h R MEDIA REGION CODE IS MISMATCHED TO LOGICAL UNIT REGION
6Fh 05h R DRIVE REGION MUST BE PERMANENT/REGION RESET COUNT ERROR
6Fh 06h R INSUFFICIENT BLOCK COUNT FOR BINDING NONCE RECORDING
6Fh 07h R CONFLICT IN BINDING NONCE RECORDING
70h NNh T DECOMPRESSION EXCEPTION SHORT ALGORITHM ID OF NN
71h 00h T DECOMPRESSION EXCEPTION LONG ALGORITHM ID
72h 00h R SESSION FIXATION ERROR
72h 01h R SESSION FIXATION ERROR WRITING LEAD-IN
72h 02h R SESSION FIXATION ERROR WRITING LEAD-OUT
72h 03h R SESSION FIXATION ERROR - INCOMPLETE TRACK IN SESSION
72h 04h R EMPTY OR PARTIALLY WRITTEN RESERVED TRACK
72h 05h R NO MORE TRACK RESERVATIONS ALLOWED
72h 06h R RMZ EXTENSION IS NOT ALLOWED
72h 07h R NO MORE TEST ZONE EXTENSIONS ARE ALLOWED
73h 00h R CD CONTROL ERROR
73h 01h R POWER CALIBRATION AREA ALMOST FULL
73h 02h R POWER CALIBRATION AREA IS FULL
73h 03h R POWER CALIBRATION AREA ERROR
73h 04h R PROGRAM MEMORY AREA UPDATE FAILURE
73h 05h R PROGRAM MEMORY AREA IS FULL
73h 06h R RMA/PMA IS ALMOST FULL
73h 10h R CURRENT POWER CALIBRATION AREA ALMOST FULL
73h 11h R CURRENT POWER CALIBRATION AREA IS FULL
73h 17h R RDZ IS FULL
74h 00h T SECURITY ERROR
74h 01h T UNABLE TO DECRYPT DATA
74h 02h T UNENCRYPTED DATA ENCOUNTERED WHILE DECRYPTING
74h 03h T INCORRECT DATA ENCRYPTION KEY
74h 04h T CRYPTOGRAPHIC INTEGRITY VALIDATION FAILED
74h 05h T ERROR DECRYPTING DATA
74h 06h T UNKNOWN SIGNATURE VERIFICATION KEY
74h 07h T ENCRYPTION PARAMETERS NOT USEABLE
74h 08h DT R M E VF DIGITAL SIGNATURE VALIDATION FAILURE

74h 09h T ENCRYPTION MODE MISMATCH ON READ
74h 0Ah T ENCRYPTED BLOCK NOT RAW READ ENABLED
74h 0Bh T INCORRECT ENCRYPTION PARAMETERS

Appendix C – Using BAM from a Program

Introduction

Beginning in the STB Suite version 8.1 the Developers Toolbox (DTB) api includes functions to use the Bus Analyzer Module (BAM) from within an application program.

This article will describe the DTB functions used to control BAM and show an example of a simple program to capture a trace and save it to a file.

The BAM-specific DTB functions

The DTB api calls to control BAM are as follows:

```
int VCSCSIBAMconfigure(long BufSize, long PhaseSize, int Flags, int Phases);

int VCSCSIBAMclearBuffer();

int VCSCSIBAMdrive(int ha, int target, int lun, int capture);

int VCSCSIBAMstartCapture();

int VCSCSIBAMstopCapture();

enum _BAM_FILE_TYPES {eBAMFileRaw, eBAMFileExcel };

typedef enum _BAM_FILE_TYPES eBAM_FILE_TYPES;

 int VCSCSIBAMsaveCapture(BYTE *fname,eBAM_FILE_TYPES eSaveType);
```

In short these functions are used to define the device which you wish to capture bus traffic to/from, which BAM phases you wish to capture, the size of your capture buffer, and flags such as STOP_ON_BUFFER_FULL to control trace capture behavior.

Once a capture session as been defined it is a simple matter to call one function to start the capture, another function to stop the capture, and finally a function to save the capture data to either raw BAM type data or to an Excel-type (comma delimited) text file.

A programming example

The following code snip shows how to control a complete BAM capture session via DTB:

```
VCSCSIBAMconfigure(32,512,1,0); // set up a 32M capture buffer, capture 512 bytes of data
                                // set flags to stop capture on buffer full, "phases" value of
                                // 0 means use the default phases settings , which is:
//    Phases = SENSE_PHASE |
//                 OK_PHASE |
//                CDB_PHASE |
//           DATA_IN_PHASE |
//          DATA_OUT_PHASE |
//                ATA_PHASE |
//        ATA_STATUS_PHASE |
//                SRB_PHASE |
//        SRB_STATUS_PHASE |
//             RESET_PHASE;
//

VCSCSIBAMclearBuffer();
VCSCSIBAMdrive(0,0,0,1); // set up to capture on boot drive
VCSCSIBAMstartCapture(); // start it
Sleep(6000); // wait while some system stuff makes I/O
VCSCSIBAMstopCapture(); // stop it
VCSCSIBAMsaveCapture(&MyFile); // save it
```

Summary

Capturing BAM trace data for any device on your test system is simply a matter of a few basic DTB function calls.