# SCSI TOOLBOX, LLC

## Using SAT to access SATA drives

# Contents

## 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

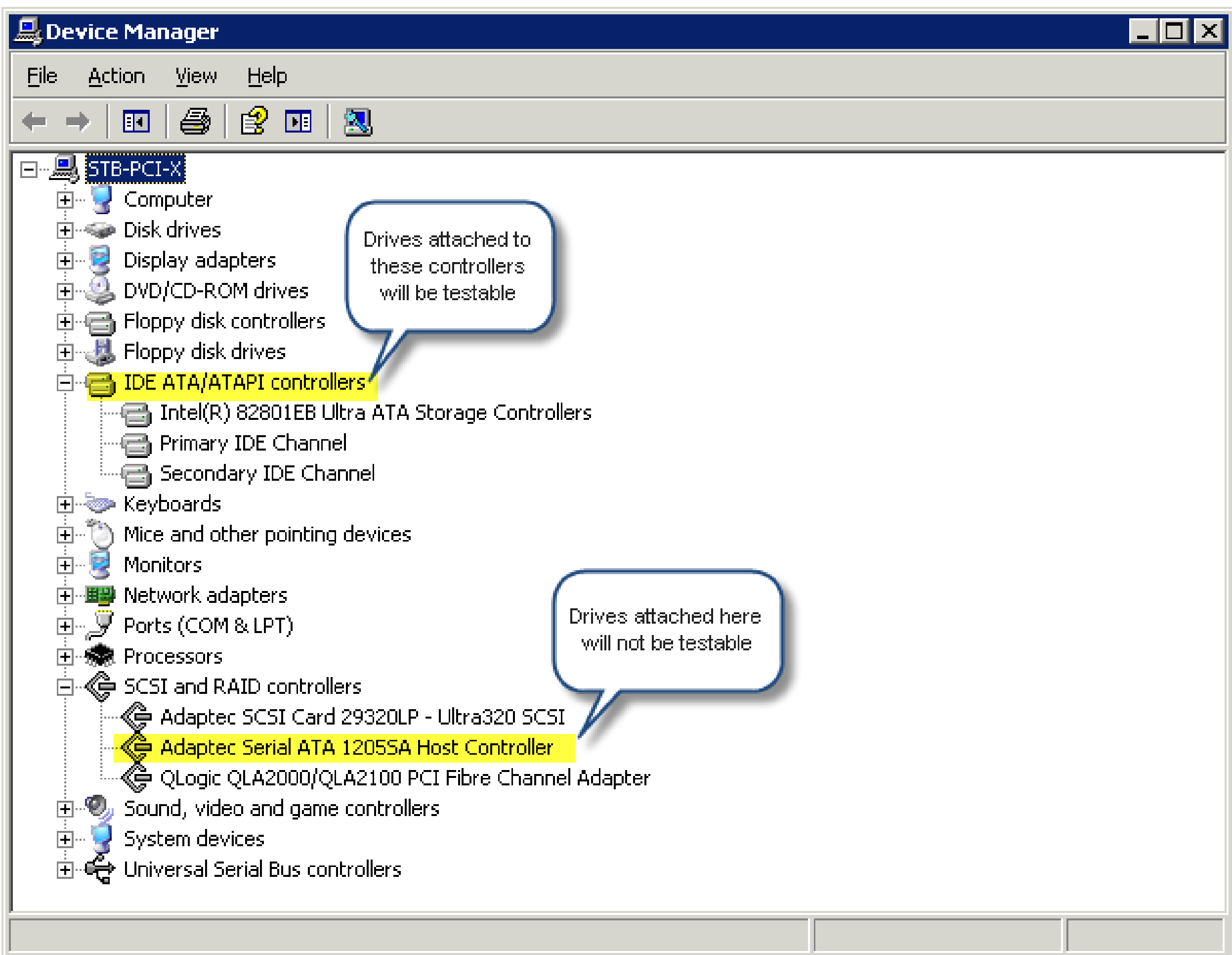
In short, SAT uses a 12 or 16 byte SCSI cdb which contains an embedded ATA task register command.

## 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.

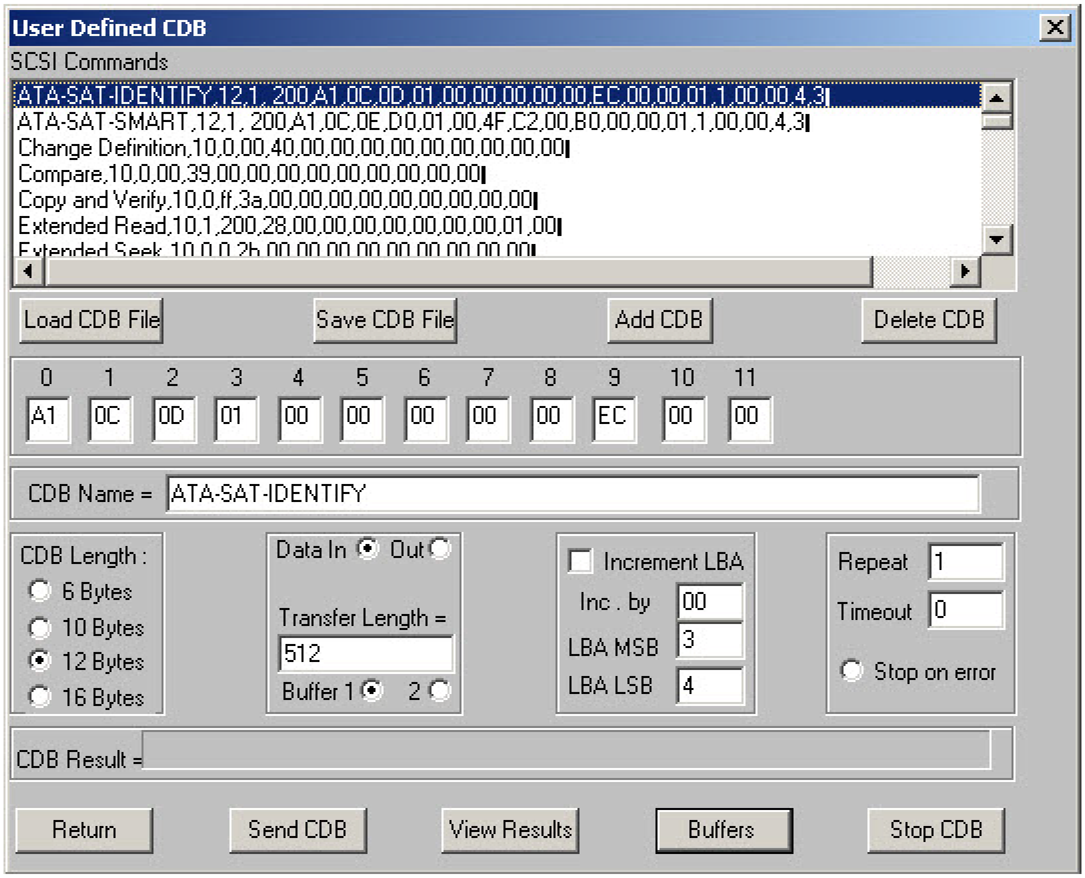
The 12-byte ATA Passthrough CDB we will use is defined as:

| 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

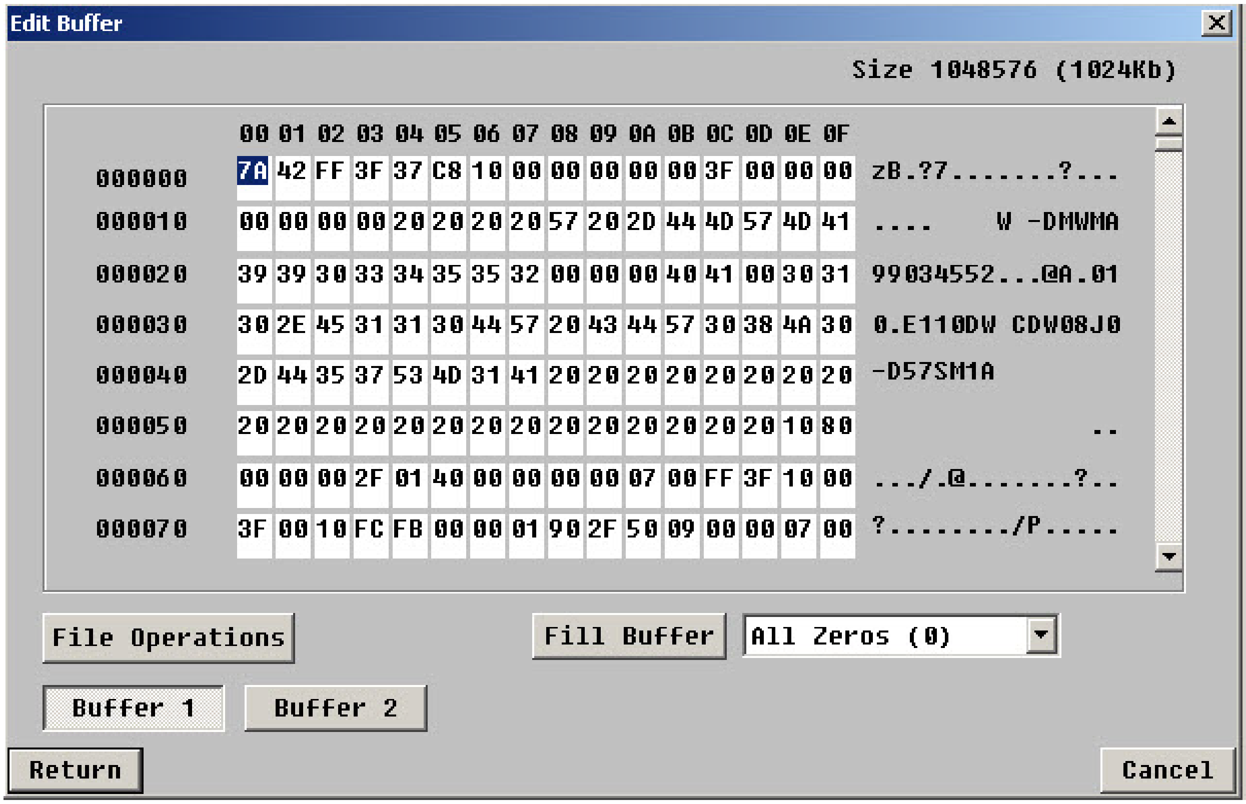
Here is a picture of the STB Suite User Defined CDB with an ATA Pass-through command defined which will issue an ATA IDENTIFY command to the attached drive




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)

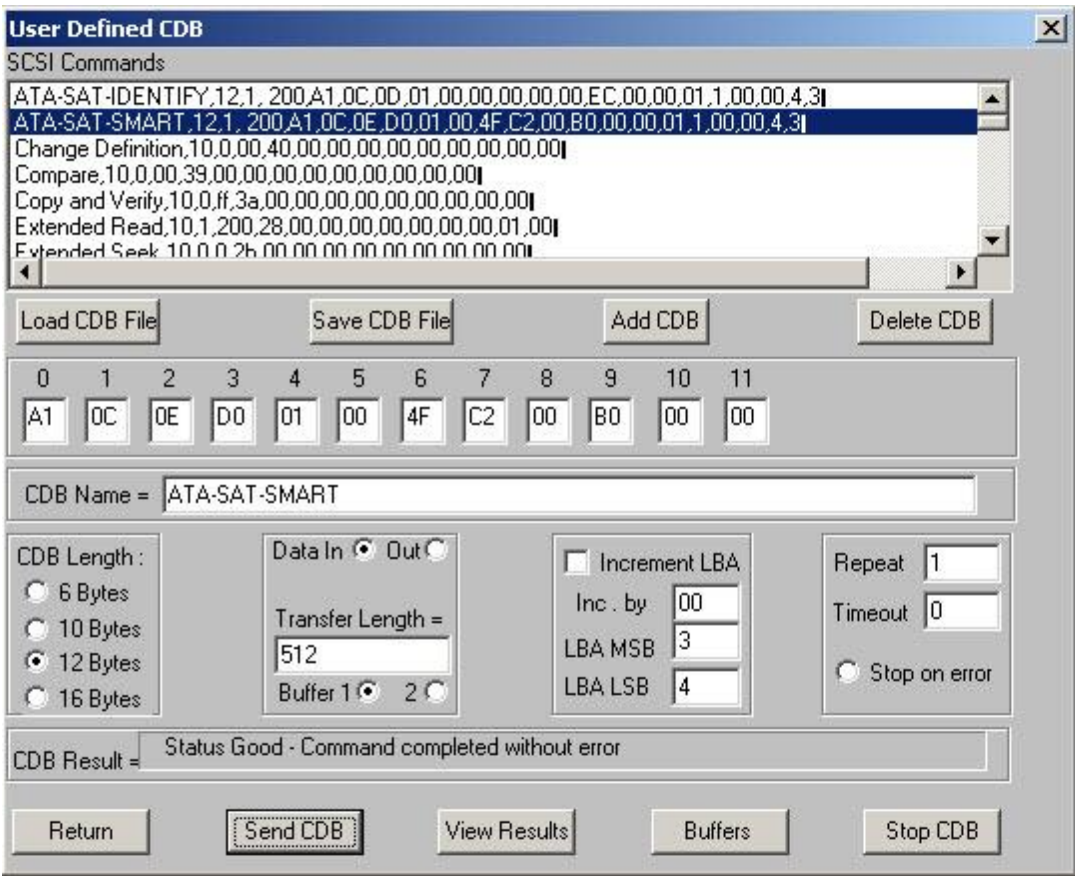
If the command completed successfully you can click the ***Buffer*** button so view the ATA IDENTIFY data returned from the drive.



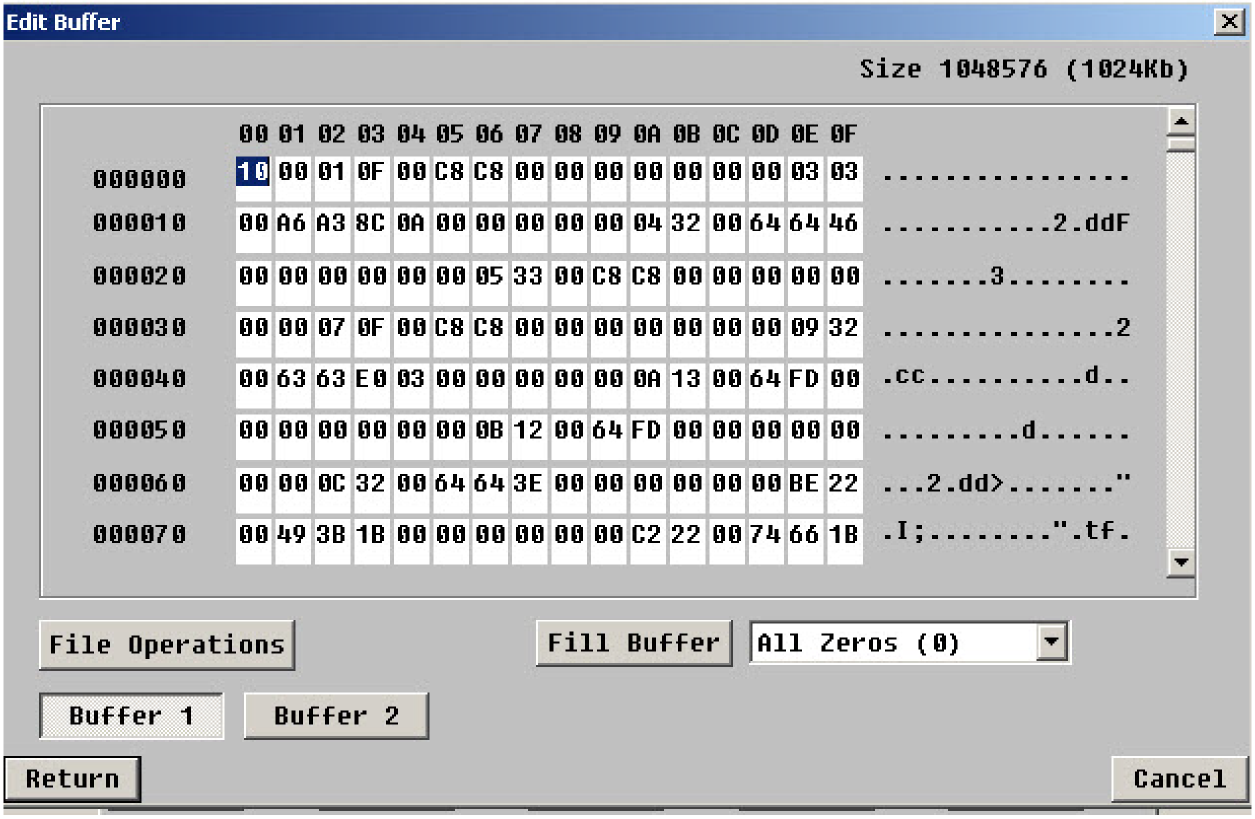

Note: ATA commands data byte-swapped.

## 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.