ATA Software Programming Interface (ATASPI) Specification

DOS and Windows® Operating Systems
NOTICES

This is a working draft of a proposed specification to be submitted as an American National Standard. As such, this is not a completed standard, has not yet received any form of approval. This specification is subject to change and use of the information contained in this document is done so at your own risk. Until such time that this proposed specification is formally submitted to the X3T10 ANSI committee, Future Domain retains all exclusive rights to edit and control this specification.

Permission is granted to users of this document to reproduce this document for standardization activities only. Any commercial for profit use is strictly prohibited.

Technical Editor
Kevin Calvert
Future Domain Corporation
2801 McGaw Ave.
Irvine, California 92714
USA

Telephone (714) 253-0522
FAX (714) 253-0913
Internet ID KEVIN@FDC.MHS.COMPUERVE.COM

The most recent version of this specification in Microsoft Word For Windows 2.0 format is available for downloading by logging onto the Future Domain BBS using the following settings:

Phone: (714) 253-0432
Communication Settings:
2400 -14000,N,8,1
V.32/V.32bis/V.42/V.42bis
Hours: 24 Hours
File Name ATASP072.EXE

Patent/Copyright Statement
The developers of this specification have requested those holder’s of patents or copyrights that may be required for the implementation of this specification, disclose such patents or copyrights to the publisher. No patent or copyright search has been undertaken at this time in order to identify any patents or copyrights that may apply.

Use Disclaimer
THIS SPECIFICATION IS MADE AVAILABLE WITHOUT CHARGE FOR USE IN DEVELOPMENT OF DEVICE DRIVERS AND APPLICATION SOFTWARE. FUTURE DOMAIN MAKES NO REPRESENTATION OR WARRANTY REGARDING THIS SPECIFICATION OR ANY ITEM BASED ON THIS SPECIFICATION, AND FUTURE DOMAIN DISCLAIMS ALL EXPRESS AND IMPLIED WARRANTIES, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND FREEDOM FROM INFRINGEMENT. FUTURE DOMAIN MAKES NO WARRANTY OF ANY KIND THAT ANY ITEM DEVELOPED BASED ON THIS SPECIFICATION WILL NOT INFRINGE ANY COPYRIGHT, PATENT, TRADE SECRET OR OTHER INTELLECTUAL PROPERTY RIGHT OF ANY PERSON OR ENTITY IN ANY COUNTRY.
# Table of Contents

1. Introduction ........................................................................................................... 1
   1.1. Document References......................................................................................... 2

2. ATASPI for DOS Specification ................................................................................ 3
   2.1. Opening ATASPI............................................................................................ 3
   2.2. Obtaining the ATASPI Entry Point................................................................. 3
   2.3. Closing ATASPI............................................................................................. 3
   2.4. Calling ATASPI from DOS ............................................................................ 4
   2.5. Calling ATASPI for DOS from Windows....................................................... 5

2.6. Commands Supported by ATASPI Manager ...................................................... 6
   2.6.1. ATA Controller Inquiry ............................................................................ 6
   2.6.2. Get ATA Device Type .............................................................................. 7
   2.6.3. Execute ATA I/O ...................................................................................... 8
   2.6.4. Abort ATA Request .................................................................................. 11
   2.6.5. Reset ATA Device .................................................................................... 11
   2.6.6. Set ATA Controller Parameter .................................................................. 12
   2.6.7. Get ATA Disk Drive Information ............................................................... 13

3. ATASPI For Windows Specification ...................................................................... 15
   3.1. Accessing ATASPI For Windows ................................................................. 15
   3.2. Windows ATASPI Functions ........................................................................ 15
       3.2.1. GetATASPISupportInfo ..................................................................... 15
       3.2.2. SendATASPICommand .................................................................... 17
   3.3. Windows ATASPI Commands ...................................................................... 17
       3.3.1. ATA Controller Inquiry Command ..................................................... 17
       3.3.2. Get ATA Device Type Command ..................................................... 19
       3.3.3. Execute ATA I/O Command ............................................................... 20
       3.3.4. Abort ATA Request Command ............................................................ 24
       3.3.5. Reset ATA Device Command ............................................................. 26

3.4. ATASPI.H ....................................................................................................... 28

4. Document History ................................................................................................ 32
List of Figures

Figure 1 - ATASPI Manager Environment

List of Tables

Table 1 - ATASPI Status Return Values .............................................. 6
Table 2 - ATA Controller Inquiry Command ........................................ 7
Table 3 - Get ATA Device Type Command .......................................... 7
Table 4 - Peripheral Device Type Values for ATAPI Devices ................. 8
Table 5 - Execute ATA I/O Command ................................................. 8
Table 6 - Execute ATA I/O Request Flags .......................................... 9
Table 7 - ACB Definition for Task File Structure ............................. 11
Table 8 - Abort ATA Request Command .......................................... 11
Table 9 - Reset ATA Device Command ............................................ 12
Table 10 - Reset ATA Device Request Flags ................................... 12
Table 11 - Set ATA Controller Parameter Command ........................... 13
Table 12 - Get ATA Disk Drive Information Command ....................... 13
Table 13 - Get ATA Disk Drive Information Drive Flags .................... 13
Table 14 - INT 13h Info bits .......................................................... 14
Table 15 - GetATSPI|SupportInfo Status Byte return values ............... 16
Table 16 - Valid SendATSPICommand Return Values ......................... 17
Table 17 - ATA Controller Inquiry ARB ......................................... 18
Table 18 - ATA Controller Inquiry ARB Return Values ..................... 18
Table 19 - Get ATA Device Type .................................................. 19
Table 20 - ATA Device Type Values ............................................. 20
Table 21 - Get ATA Device Type ARB Return Values ......................... 20
Table 22 - Execute ATA I/O ARB .................................................. 22
Table 23 - ARB_Flags .................................................................. 22
Table 24 - Execute ATA I/O ARB Return Values ............................... 24
Table 25 - Abort ATA Request Command ....................................... 25
Table 26 - Abort ATA Request Command Return Values .................. 25
Table 27 - Reset ATA Device ARB .............................................. 26
Table 28 - Reset ATA Device ARB Return Values ............................. 27
1. Introduction

ATA Software Programming Interface (ATASPI) specification defines an API used by developers to control IDE, eIDE and ATAPI compliant peripherals in an PC environment. ATASPI is a software oriented interface designed to simplify writing device drivers for ATA/ATAPI peripherals attached to ATA controllers. ATASPI allows all industry standard ATA controllers to be accessed in the same manner, allowing a single device driver to be written for all of the controllers.

Most often, device driver code is written to fill the gap between two interfaces. One interface is generally provided by the operating system, the other by the target hardware. In this case, the target hardware is an all industry standard ATA controller in combination with some ATA/ATAPI peripheral.

In the past, IDE hard disks were the only peripheral type attached on the ATA bus. These disks were being controlled by one host who provided the software support (e.g. system BIOS, option ROM BIOS or Windows Fast Disk ...). Due to this simple configuration, there was no need for complex ATA chips; BIOS and OS drivers designed to directly address the hardware to avoiding unnecessary programming overhead.

If all device drivers followed the same path, a new driver had to be written each time a new controller comes out with a new ATA chip. Such a new driver would take into account the differences between controllers. In well-structured code, this was laborious at best. In poorly structured code, this usually resulted in an almost completely new driver for each new controller.

Another problem often encountered by device driver developers was a lack of knowledge both about how all industry standard hardware worked, and about the lowest levels of ATA/ATAPI protocol. A programmer had to expend considerable effort programming the controller hardware instead of concentrating on programming his ATA/ATAPI device.

Finally, another problem encountered was when two device drivers were both loaded, one might try to take action while the other was already manipulating the hardware. This might cause one or both drivers to fail. The solution to all the problems stated above is an interface that:

- Can handle the low level protocol issues of an ATA controller
- Can be accessed by more than one device driver
- Can provide a consistent interface to device drivers despite the fact that the underlying hardware was completely different from one controller to another.

This solution is an ATASPI Manager that is meant to significantly simply the ATA/ATAPI device driver developer's job by handling all the low level ATA/ATAPI details and providing a consistent interface across different controllers. This lets the programmer concentrate on the important job of implementing a new driver for his new device. Since all accesses to the hardware are handled by the ATASPI Manager, multiple device drivers can use the same controller without conflicting with one another. This allows devices of different types (IDE, eIDE, ATAPI CD-ROM, ATAPI Tape ...) to share the same ATA controller. Figure 1 shows how an ATASPI Manger fits in a typical ATA/ATAPI environment:
Figure 1 - ATASPI Manager Environment

1.1. Document References

- AT Attachment Interface with Extensions Revision 2d or later ("ATA-2"), ANSI X3 Committee (June 24, 1994)
- ATA Packet Interface for CD-ROM's, SFF-8020 Revision 1.2 or later ("ATAPI CD-ROM"), Small Form Factor Committee (February 24, 1994)
- ATAPI for Streaming Tape Devices Revision C or later ("ATAPI TAPE"), (May 14, 1994)
2. ATASPI for DOS Specification

2.1. Opening ATASPI

Device drivers wishing to access ATASPI must open the driver by performing a DOS INT 21h function call to OPEN A FILE. The following indicates the register condition upon entry and exit of the call to INT 21h:

On Entry:
- AX = 3D00h
- DS:DX = Pointer to '$ATAMGR$', 0

On Return:
- AX = File Handle if carry flag is not set
- Error Code if carry flag is set

2.2. Obtaining the ATASPI Entry Point

Device drivers can obtain the entry point to ATASPI by performing the DOS INT 21h function call IOCTL READ. The following indicates the register condition upon entry and exit of the call to INT 21h:

On Entry:
- AX = 4402h
- DS:DX = Pointer to a 4 byte data buffer
- CX = 4
- BX = File Handle

On Return:
- AX = Nothing

The following data is returned in the buffer pointed to by DS:DX containing the ATASPI entry point.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size in Bytes</th>
<th>Direction</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>02h</td>
<td>In</td>
<td>ATASPI entry point Offset¹</td>
</tr>
<tr>
<td>02h</td>
<td>02h</td>
<td>In</td>
<td>ATASPI entry point Segment²</td>
</tr>
</tbody>
</table>

Device drivers could then use this ATASPI entry point to send a request to the ATASPI Manager.

2.3. Closing ATASPI

Device drivers wishing to close ATASPI must do it by performing a DOS INT 21h function call CLOSE A FILE. The following indicates the register condition upon entry and exit of the call to INT 21h:

On Entry:
- AH = 3Eh
- BX = File Handle

On Return:
- AX = Error code if carry flag is set
- Nothing if carry flag is not set

¹ Offset portion of a 32-bit pointer in Intel X86 format (16-bit offset, 16-bit segment)
² Segment portion of a 32-bit pointer in Intel X86 format (16-bit offset, 16-bit segment)
2.4. Calling ATASPI from DOS

The following code segment is an example on calling ATASPI:

```c
#include <stdio.h>
#include <dos.h>
#define DOS_OPEN_FILE 0x3D00
#define DOS_IOCTL_READ 0x4402
#define DOS_CLOSE_FILE 0xEE

typedef char (far * function)();

main()
{
    union REGS
    {
        struct SREGS
        {
            xregister;
        } sregister;
        char __far *ATASPINamePointer;
        char __far *ATASPIEntryPoint;
        char ATASPIName[] = "$ATAMGR$";
        function ATASPIEntryPoint;
        int FileHandle, i;
        char ATACControllerInquiry[64];
    } xregister;

    ATASPINamePointer = (char __far *)(&ATASPIName[0]);
    ATASPIEntryPoint = (char __far *)(&ATASPIEntryPoint);

    // Open the ATASPI Manager
    xregister.x.ax = DOS_OPEN_FILE;
    sregister.ds = FP_SEG(ATASPINamePointer);
    xregister.x.dx = FP_OFF(ATASPINamePointer);
    intdosx(&xregister, &xregister, &sregister);
    if (xregister.x.cflag)
    {
        printf("Error: ATASPI Manager not found\n");
        return;
    }

    // Get the ATASPI Entry Point
    FileHandle = xregister.x.ax;
    xregister.x.bx = FileHandle;
    xregister.x.ax = DOS_IOCTL_READ;
    sregister.ds = FP_SEG(ATASPIEntryPoint);
    xregister.x.dx = FP_OFF(ATASPIEntryPoint);
    xregister.x.cx = 4;
    intdosx(&xregister, &xregister, &sregister);

    // Close the ATASPI Manager
    xregister.h.ah = DOS_CLOSE_FILE;
    xregister.x.bx = FileHandle;
    intdosx(&xregister, &xregister, &sregister);

    // Initialize the ATASPI Controller Inquiry Command
```
for (i=0; i<64; i++)
    ATAControllerInquiry[i] = 0;

// Issue an ATASPI ATA Controller Inquiry Command

(*ATASPIEntryPoint)(ATAControllerInquiry);

if (ATAControllerInquiry[1]==1) {
    printf("ATA Controller Inquiry successful\n");
} else {
    printf("ATA Controller Inquiry returned with error code %2x\n",ATAControllerInquiry[1]);
}

return;

2.5. Calling ATASPI for DOS from Windows

Microsoft Windows is a Graphical User Interface which runs under DOS as an application. However, writing a device driver or application capable of making ATASPI calls in a Windows environment is not as simple as in the strictly DOS case. The problem arises because ATASPI uses Real Mode Interface and Windows uses the DOS Protected Mode Interface (DPMI). Whereas ATASPI expects a physical Segment and Offset for the ATA Request and the entry point of ATASPI, Windows uses a "Selector" and Offset to address data and code. In order to program correctly in this environment a consortium of companies, Microsoft and Intel among them, has written a specification called the "DOS Protected Mode Interface Specification". The details of the specification are too complex to go into detail here and it is recommended a copy can be obtained from the DPMI committee for programming purposes. However, as a brief overview two steps need to be followed to comply with DPMI:

1. Allocate all ATA Requests and buffers down in Real Mode Memory. This can be accomplished using Window's "GlobalDOSAlloc" routine or using DPMI Interrupt 31h - Function 100h. This makes it possible for the ATASPI module and manager to locate the ATA Request and data buffer using segments and offsets.

2. Call the Real Mode Procedure with Far Return Frame Function (Interrupt 31h - Function 0301h). This makes it possible to call the ATASPI Manager which is a Real Mode Procedure.

Also, see ATASPI For Windows Specification section on page 14.
2.6. Commands Supported by ATASPI Manager

The ATASPI Manager supports the following ATA Request Blocks (ARB):

- ATA Controller Inquiry
- Get ATA Device Type
- Execute ATA I/O
- Abort ATA Request
- Reset ATA Device
- Set ATA Controller Parameter
- Get ATA Disk Drive Information

Upon return, the ATASPI Manager sets the Status byte to one of the following possible values. The actual values returned depend on the command called. Consult each command for valid return values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>Request in progress</td>
</tr>
<tr>
<td>01h</td>
<td>Request completed without error</td>
</tr>
<tr>
<td>02h</td>
<td>Request aborted by host</td>
</tr>
<tr>
<td>04h</td>
<td>Request completed with error</td>
</tr>
<tr>
<td>80h</td>
<td>Invalid request</td>
</tr>
<tr>
<td>81h</td>
<td>Invalid ATA controller number</td>
</tr>
<tr>
<td>82h</td>
<td>ATA Device not installed</td>
</tr>
<tr>
<td>83h</td>
<td>ATA Controller/Device busy</td>
</tr>
</tbody>
</table>

Table 1 - ATASPI Status Return Values

Note: If ATASPI ever returns 83h as the status, it is the responsibility of the driver / applications to re-send the request at a later time.

Note: All reserved fields defined for each ARB must be initialized to zero.

Note: In the sections that follow, various tables use the following headings and are defined as follows:

- **Offset**: Field starting location in data buffer.
- **Size in Bytes**: The length of this field in bytes.
- **Direction**: Indicates when data is valid. 
  - *In* indicates data valid after return,
  - *Out* indicates data to be supplied when ATASPI is called.

2.6.1. ATA Controller Inquiry

*ATA Controller Inquiry* has two purposes. One is to get the total number of ATA controllers found by the manager and the other is to retrieve Controller Unique Parameters.

To get the total number of ATA controllers, *ATA Controller Number* must be set to 0FFh. Specific information on a controller (as specified in *ATA Controller Number*) is returned in Controller Unique Parameters.
Table 2 - ATA Controller Inquiry Command

The Status byte returned is set to one of the following values:

01h Completed without error
81h Invalid ATA Controller number

The ATA Manager ID field contains a 16-byte ASCII string describing the ATA Manager.

The ATA Controller ID field contains a 16-byte ASCII string describing the ATA controller.

The Controller Unique Parameters field is vendor unique and is defined by controller vendors.

2.5.2. Get ATA Device Type

Get ATA Device Type returns the device type of the specified target. If more detailed information is necessary, issue the Inquiry Command through Execute ATA I/O.

The Device ID field can be set to:
00h Device 0 (Master device)
01h Device 1 (Slave device)

The Status byte returned is set to one of the following values:
01h Device Present and Peripheral Device Type field is valid
82h Device Not Present
For ATAPI devices, the Peripheral Device Type field contains byte 0 of the Inquiry Command.

<table>
<thead>
<tr>
<th>Value</th>
<th>Device Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>Direct-access device (e.g. magnetic disk)</td>
</tr>
<tr>
<td>01h</td>
<td>Tape device (QIC-121 SCSI Architectural Model)</td>
</tr>
<tr>
<td>02h-04h</td>
<td>Reserved</td>
</tr>
<tr>
<td>05h</td>
<td>CD-ROM device</td>
</tr>
<tr>
<td>06h</td>
<td>Reserved</td>
</tr>
<tr>
<td>07h</td>
<td>Optical memory device (e.g. some optical disks)</td>
</tr>
<tr>
<td>08h-0Eh</td>
<td>Reserved</td>
</tr>
<tr>
<td>0Fh</td>
<td>Tape device (Cost Sensitive Architectural Model)</td>
</tr>
<tr>
<td>00h-1Eh</td>
<td>Reserved</td>
</tr>
<tr>
<td>1Fh</td>
<td>Unknown or no device type</td>
</tr>
</tbody>
</table>

Table 4 - Peripheral Device Type Values for ATAPI Devices

For non-ATAPI devices such as eIDE and IDE disks, the Peripheral Device Type field contains 80h.

2.6.3. Execute ATA I/O

The Execute ATA I/O command is used to issue ATA and ATAPI commands to IDE devices. The caller of this command may be informed of the completion of the command through either a Polling or a Posting method.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size in Bytes</th>
<th>Direction</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>01h</td>
<td>Out</td>
<td>Command Code = 2</td>
</tr>
<tr>
<td>01h</td>
<td>01h</td>
<td>In</td>
<td>Status</td>
</tr>
<tr>
<td>02h</td>
<td>01h</td>
<td>Out</td>
<td>ATA Controller Number (zero-based)</td>
</tr>
<tr>
<td>03h</td>
<td>01h</td>
<td>Out</td>
<td>Request Flag</td>
</tr>
<tr>
<td>04h</td>
<td>04h</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>06h</td>
<td>01h</td>
<td>Out</td>
<td>Device ID</td>
</tr>
<tr>
<td>08h</td>
<td>01h</td>
<td>Out</td>
<td>Reserved</td>
</tr>
<tr>
<td>0Ah</td>
<td>04h</td>
<td>Out</td>
<td>Data Transfer Length</td>
</tr>
<tr>
<td>0Eh</td>
<td>01h</td>
<td>Out</td>
<td>Sense Allocation Length (N)</td>
</tr>
<tr>
<td>0Fh</td>
<td>02h</td>
<td>Out</td>
<td>Data Buffer Pointer (Offset)</td>
</tr>
<tr>
<td>11h</td>
<td>02h</td>
<td>Out</td>
<td>Data Buffer Pointer (Segment)</td>
</tr>
<tr>
<td>13h</td>
<td>04h</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>17h</td>
<td>01h</td>
<td>Out</td>
<td>ACB Length (M)</td>
</tr>
<tr>
<td>18h</td>
<td>01h</td>
<td>Out</td>
<td>ATA Controller Status</td>
</tr>
<tr>
<td>19h</td>
<td>01h</td>
<td>In</td>
<td>Device Status</td>
</tr>
<tr>
<td>1Ah</td>
<td>02h</td>
<td>Out</td>
<td>Post Routine Address (Offset)</td>
</tr>
<tr>
<td>1Ch</td>
<td>02h</td>
<td>Out</td>
<td>Post Routine Address (Segment)</td>
</tr>
<tr>
<td>1Eh</td>
<td>02h</td>
<td>Out</td>
<td>Data Transfer Block Size (in bytes)</td>
</tr>
<tr>
<td>20h</td>
<td>20h</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>40h</td>
<td>M</td>
<td>Out</td>
<td>ATA Command Block (ACB)</td>
</tr>
<tr>
<td>40h+M</td>
<td>N</td>
<td>In</td>
<td>Sense Allocation Area</td>
</tr>
</tbody>
</table>

Table 5 - Execute ATA I/O Command

The Status byte returned is set to one of the following values:

- 00h: Request in progress
- 01h: Completed without error
- 02h: Request aborted by host
- 04h: Completed with error
- 81h: Invalid ATA Controller Number
ATA Software Programming Interface (ATASPI) Specification
for DOS and Windows Operating Systems

82h  ATA Device Not Present
83h  ATA Controller/Device busy

The Request Flags byte is defined as follows:

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

Table 6 - Execute ATA I/O Request Flags

The Post bit indicates whether we need to call a Post routine upon completion of the request:

0  Disable Posting
1  Enable Posting

Posting is used to notify a calling routine that the command request has been completed. A posting routine is also referred to as a callback routine.

The Request Type bit specifies whether ARB contains an ATAPI Packet command or an AT Task File Structure:

0  ATAPI Packet command (e.g. Inquiry, Mode Select ...)
1  AT Task File Structure

The Direction bits specify the data transfer directions:

00  Direction determined by device
01  Data In
10  Data Out
11  No Data Transfer

If the command issued involved data transfer, setting the Direction bits to Data In or Data Out is required. No Data Transfer option can only be used with commands that have no data transfer such as Seek, Execute Device Diagnostic ... [See Data Transfer Length field for additional data transfer capabilities].

The DUA Bit (DSC Unavailable Action) for an ATAPI packet command determines what action the ATASPI manager is to take if the DSC Bit is not set in the task file status register. The ATASPI manager expects the DSC bit to be set before sending an ATAPI command block. The DUA Bit allows the caller to specify which action the ATASPI Manager to take if DSC bit not set:

0  The ATASPI manager must queue this ATAPI request and service it latter when DSC bit is set.
1  The ATASPI manager must return to the caller with status set to a BUSY condition. It is the responsibility of the caller to issue the request at a latter time.

The ByteXfer bit indicates the data transfer method to use and is defined as follows:

0  use word transfer mode
1  use byte transfer mode
The Device ID field can be set to:

00 Device 0 (Master device)  
01 Device 1 (Slave device)

The Data Transfer Length field indicates the total number of data bytes the application wishes to send out or receive. For an ATA Task File Request, this length must be equal to the number of data bytes actually transferred on the bus. For an ATAPI Packet request, this length could be different from the number of data bytes transferred on the bus. If they are not the same, the ATASPI Manager will pad or truncate the appropriate number of bytes to complete the transaction and notify the caller with the ATA Controller Status field set to Data Underrun / Overrun.

The ACB Length field indicates the total number of valid bytes in the ACB field. For Task File request, ACB Length must be set to 7. For ATAPI Packet requests the value is dependent on the packet size and must match the number of bytes contained in the packet.

The ATA Controller Status field returns one of the following status:

00h No error  
11h Device Not Present  
12h Data Overrun / Underrun

The Device Status field returns 00h (No Device status) if there is no error, otherwise it would contain the Error Register upon return. (Refer to the ATA-2 or ATAPI Packet Interface specs for detail description of the Error Register).

The Post Routine Address field is used only if the Post bit is set in the Request Flag.

The Data Transfer Block Size field is used to specify the number of data bytes to transfer per hardware interrupt. For Task File request, Data Transfer Block Size (DTBS) is used to determine the maximum number of data bytes to transfer per hardware interrupt. The default value used by the ATASPI manager (when DTBS = 0) is 512 bytes. If necessary (for Read/Write Multiple ...), this could be set to a different value. For ATAPI Packet commands, Data Transfer Block Size (DTBS) is used to tell the device how much data the host prefers to transfer per interrupt. The default value used by ATASPI (when DTBS = 0) is 930h bytes. This is only a preferred value the ATASPI Manager issues to the device but the device controls the actual number of bytes per hardware interrupt to transfer.
If the Request Flag indicates an ATAPI Packet command, the ATA/ATAPI Command Block (ACB) field contains the command bytes to be sent out to the Device. If the Request Flag indicates an AT Task File Structure, the ACB is defined as follows:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Features</td>
<td>Command specific features</td>
</tr>
<tr>
<td>1</td>
<td>Sector Count</td>
<td>Number of Sectors</td>
</tr>
<tr>
<td>2</td>
<td>Sector Number</td>
<td>In CHS mode, starting sector number for the command</td>
</tr>
<tr>
<td></td>
<td>* LBA bits 0-7</td>
<td>* In LBA mode, contains bits 0-7</td>
</tr>
<tr>
<td>3</td>
<td>Cylinder LSB</td>
<td>In CHS mode, low order 8 bits of starting cylinder address</td>
</tr>
<tr>
<td></td>
<td>* LBA bits 8-15</td>
<td>* In LBA mode, contains bits 8-15</td>
</tr>
<tr>
<td>4</td>
<td>Cylinder MSB</td>
<td>In CHS mode, high order bits of starting cylinder address</td>
</tr>
<tr>
<td></td>
<td>* LBA bits 16-23</td>
<td>* In LBA mode, contains bits 16-23</td>
</tr>
<tr>
<td>5</td>
<td>Device / Head</td>
<td>In CHS mode, refer to ATA spec for description</td>
</tr>
<tr>
<td></td>
<td>* LBA bits 24-27</td>
<td>* In LBA mode, contains bits 24-27</td>
</tr>
<tr>
<td>6</td>
<td>Command</td>
<td>Command to be sent to Device; refer to ATA spec for list of commands.</td>
</tr>
</tbody>
</table>

Table 7 - ACB Definition for Task File Structure

The Sense Allocation Area is filled with Sense Data only if a Check Condition occurred and Sense Allocation Length is non-zero.

2.6.4. Abort ATA Request

The Abort ATA Request command is used to terminate a previously issued ARB. The success of this command is never assured. This command always returns successful status.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size in Bytes</th>
<th>Direction</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>01h</td>
<td>Out</td>
<td>Command Code = 3</td>
</tr>
<tr>
<td>01h</td>
<td>01h</td>
<td>In</td>
<td>Status</td>
</tr>
<tr>
<td>02h</td>
<td>01h</td>
<td>Out</td>
<td>ATA Controller Number (zero-based)</td>
</tr>
<tr>
<td>03h</td>
<td>01h</td>
<td>Out</td>
<td>Request Flag = 0</td>
</tr>
<tr>
<td>04h</td>
<td>04h</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>05h</td>
<td>02h</td>
<td>Out</td>
<td>Offset of ARB to be aborted</td>
</tr>
<tr>
<td>0Ah</td>
<td>02h</td>
<td>Out</td>
<td>Segment of ARB to be aborted</td>
</tr>
</tbody>
</table>

Table 8 - Abort ATA Request Command

The Status byte returned is set to the following value:

01h Completed without error

2.6.5. Reset ATA Device

The Reset ATA Device command is used to reset a specific device. The consequence of this command is dependent on the type of device to be reset. If the specified device does not support a packetized protocol, e.g., an IDE hard disk, the SRST bit in the Device Control Register (refer to the ATA-2 spec for more information) will be asserted. This will cause all devices in the same bus to be reset. If the device is an ATAPI (packetized protocol) device, an ATAPI Soft Reset (ATA Code 08h) will be issued and will cause only the specified device to be reset.
If the specified device is an ATAPI device, it will receive an ATAPI Soft Reset (ATA code = 08h).

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size in Bytes</th>
<th>Direction</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>01h</td>
<td>Out</td>
<td>Command Code = 4</td>
</tr>
<tr>
<td>01h</td>
<td>01h</td>
<td>In</td>
<td>Status</td>
</tr>
<tr>
<td>02h</td>
<td>01h</td>
<td>Out</td>
<td>ATA Controller Number (zero-based)</td>
</tr>
<tr>
<td>03h</td>
<td>01h</td>
<td>Out</td>
<td>Request Flag</td>
</tr>
<tr>
<td></td>
<td>04h</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>05h</td>
<td>01h</td>
<td>Out</td>
<td>Device ID</td>
</tr>
<tr>
<td>06h</td>
<td>01h</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>07h</td>
<td>01h</td>
<td>In</td>
<td>ATA Controller Status</td>
</tr>
<tr>
<td>08h</td>
<td>01h</td>
<td>In</td>
<td>Device Status</td>
</tr>
<tr>
<td>1Ah</td>
<td>02h</td>
<td>Out</td>
<td>Post Routine Address (Offset)</td>
</tr>
<tr>
<td>1Ch</td>
<td>02h</td>
<td>Out</td>
<td>Post Routine Address (Segment)</td>
</tr>
<tr>
<td>1Dh</td>
<td>01h</td>
<td></td>
<td>reserved for ATAPI workspace</td>
</tr>
</tbody>
</table>

Table 9 - Reset ATA Device Command

The Device ID field can be set to:

0  Device 0 (Master device)
1  Device 1 (Slave device)

The Request Flags byte is defined as follows:

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

    | Revd | Revd | Revd | Revd | Revd | Revd | Revd |

Table 10 - Reset ATA Device Request Flags

The Post bit indicates whether we need to call a Post routine upon completion of the request:

0  Disable Posting
1  Enable Posting

The Status byte returned is set to one of the following values:

01h  Request completed without error
81h  Invalid ATA Controller Number
82h  Device Not Present

The ATA Controller Status field returns one of the following status:

00h  No error
11h  Device Not Present

The Device Status field returns 00h (No Device status) if there is no error, otherwise it contains the Error Register upon return. (Refer to the ATA-2 or ATA Packet Interface specs for detail description of the Error Register).

The Post Routine Address field is used only if the Post bit is set in the Request Flag.
2.6.6. Set ATA Controller Parameter

ATA controllers with different features, capabilities could be set using this command.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size in Bytes</th>
<th>Direction</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>01h</td>
<td>Out</td>
<td>Command Code = 5</td>
</tr>
<tr>
<td>01h</td>
<td>01h</td>
<td>In</td>
<td>Status</td>
</tr>
<tr>
<td>02h</td>
<td>01h</td>
<td>Out</td>
<td>ATA Controller Number (zero-based)</td>
</tr>
<tr>
<td>03h</td>
<td>01h</td>
<td>Out</td>
<td>Request Flag = 0</td>
</tr>
<tr>
<td>04h</td>
<td>01h</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>06h</td>
<td>10h</td>
<td>Out</td>
<td>Controller Unique Parameters (Vendor Unique)</td>
</tr>
</tbody>
</table>

Table 11 - Set ATA Controller Parameter Command

Controller Unique Parameters field is vendor unique and to be defined by controller vendors.

2.6.7. Get ATA Disk Drive Information

The Get ATA Disk Information command is used to obtain the INT 13h disk drive status and information.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size in Bytes</th>
<th>Direction</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>01h</td>
<td>Out</td>
<td>Command Code = 6</td>
</tr>
<tr>
<td>01h</td>
<td>01h</td>
<td>In</td>
<td>Status</td>
</tr>
<tr>
<td>02h</td>
<td>01h</td>
<td>Out</td>
<td>ATA Controller Number (zero-based)</td>
</tr>
<tr>
<td>03h</td>
<td>01h</td>
<td>Out</td>
<td>Request Flag = 0</td>
</tr>
<tr>
<td>04h</td>
<td>01h</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>06h</td>
<td>01h</td>
<td>Out</td>
<td>Device ID</td>
</tr>
<tr>
<td>09h</td>
<td>01h</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>0Ah</td>
<td>01h</td>
<td>In</td>
<td>Drive Flags</td>
</tr>
<tr>
<td>0Bh</td>
<td>01h</td>
<td>In</td>
<td>INT 13h Drive</td>
</tr>
<tr>
<td>0Ch</td>
<td>01h</td>
<td>In</td>
<td>Preferred Head Translation</td>
</tr>
<tr>
<td>0Dh</td>
<td>01h</td>
<td>In</td>
<td>Preferred Sector Translation</td>
</tr>
<tr>
<td>0Eh</td>
<td>01h</td>
<td>-</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Table 12 - Get ATA Disk Drive Information Command

The Status byte returned is set to the following value:
01h Completed without error

The Device ID field can be set to:
00h Device 0 (Master device)
01h Device 1 (Slave device)

The Drive Flags byte is defined as follows:

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revd</td>
<td>Revd</td>
<td>Revd</td>
<td>Revd</td>
<td>Revd</td>
<td>Revd</td>
<td>INT 13h Info</td>
<td></td>
</tr>
</tbody>
</table>

Table 13 - Get ATA Disk Drive Information Drive Flags
The INT 13h Info bits return information pertaining to the INT 13h Drive field. The following table defines the values returned.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>The given drive (Controller Number / Device ID) is not accessible via INT 13h. If you wish to read/write to this drive, you'll need to send ATASIPI read/write requests to the drive. The INT 13h Drive field is invalid.</td>
</tr>
<tr>
<td>01</td>
<td>The given drive (Controller Number / Device ID) is accessible via INT 13h. The INT 13h Drive field contains the drive's INT 13h drive number. This drive is under control of DOS.</td>
</tr>
<tr>
<td>10</td>
<td>The given drive (Controller Number / Device ID) is accessible via INT 13h. The INT 13h Drive field contains the drive's INT 13h drive number. This drive is NOT under control of DOS and can be used, for example, by a ATA Disk Driver.</td>
</tr>
<tr>
<td>11</td>
<td>Invalid</td>
</tr>
</tbody>
</table>

Table 14 - INT 13h Info bits

The INT 13h Drive field returns the INT 13h drive number for the given controller number, device ID. Valid INT 13h drive numbers range from 80-Ffh.

The Preferred Head Translation field indicates the given controller / disk drive's preferred head translation method.

The Preferred Sector Translation field indicates the given controller / disk drive's preferred sector translation method.
3. ATASPI For Windows Specification

The ATASPI for Windows Manager is a Windows v3.x Dynamic Link Library (DLL) named ATASPI.DLL. The DLL is loaded into memory by Windows when one of its exported functions is called by a Windows application. The Windows ATASPI Manager is multi-tasking and can support both polling and posting for completion of ATASPI commands.

It is beyond the scope of this specification to define the protocol between ATASPI for DOS and ATASPI For Windows. Different applications have different needs and performance may or may not be an issue. Support for DOS, Windows or both are the decision of each vendor. Hardware vendors will typically provide an ATASPI manager for their chip set or possibly a generic solution. Implementation and the complete feature set supported will depend on each vendor supplying an ATASPI manager.

3.1. Accessing ATASPI For Windows

Access to the Windows ATASPI Manager is via two library function calls. These functions are imported to the Windows application by adding the following lines in the .DEF file of the application:

```
IMPORTS
   ATASPI.GetATASPISupportInfo
   ATASPI.SendATASPICommand
```

The following briefly describes each of these functions:

GetATASPISupportInfo
This function is used to determine the status of the Windows ATASPI Manager. It is also used to find out how many controllers are supported by ATASPI.

SendATASPICommand
This function is used to issue ATASPI commands to the ATASPI Manager.

3.2. Windows ATASPI Functions

3.2.1. GetATASPISupportInfo

WORD GetATASPISupportInfo(VOID)

The GetATASPISupportInfo function returns information about the status of the Windows ATASPI Manager. This function must be called first before attempting to issue any ATASPI commands. If this function does not return a successful completion code (AS_COMP), it is not recommended to issue any ATASPI commands.

The GetATASPISupportInfo function does not pass any parameters. Upon completion, the function returns a WORD value which indicates the status of the ATASPI manager. The format of the returned WORD value is as follows:

HIBYTE: Status Byte
LOBYTE: Total Number of Controllers Supported if Status Byte indicates success or, 0 if Status Byte indicates error
Returns

The following are the valid Status Byte (HIBYTE) values returned:

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Value (in Hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS_COMP</td>
<td>01</td>
<td>Windows ATASPI Manager is successfully initialized and is ready to accept ATASPI commands.</td>
</tr>
<tr>
<td>AS_NO_WIN_MANAGER</td>
<td>E1</td>
<td>One of the modules required by Windows ATASPI Manager is not loaded. ATASPI commands will fail.</td>
</tr>
<tr>
<td>AS_ILLEGAL_MODE</td>
<td>E2</td>
<td>Windows is not running in 386 Enhanced mode.</td>
</tr>
<tr>
<td>AS_FAILED_INIT</td>
<td>E4</td>
<td>Windows ATASPI Manager failed initialization.</td>
</tr>
</tbody>
</table>

Table 15 - GetATSPISupportInfo Status Byte return values

Example Code

```c
WORD WINATASPIStatus;
BYTE NumControllers;
HWND hwnd;

WINATASPIStatus = GetATSPISupportInfo();
switch ( HIBYTE(WINATASPIStatus) )
{
  case AS_COMP:
    // ATASPI for Windows is properly initialized
    NumControllers = LOBYTE(WINATASPIStatus);
    break;
  case AS_ILLEGAL_MODE:
    MessageBox( hwnd,"ATASPI for Windows does not support this mode!!", szAppName,MB_ICONSTOP );
    return 0;
  case AS_NO_WIN_MANAGER:
    MessageBox( hwnd,"No Windows ATASPI Manager found!!", szAppName,MB_ICONSTOP );
    return 0;
  default:
    MessageBox( hwnd,"ATASPI for Windows is not initialized!!", szAppName,MB_ICONSTOP );
    return 0;
}
```
3.2.2. SendATASPICommand

WORD SendATASPICommand(lpARB ARB)

The SendATASPICommand function is used to issue ATASPI commands to the Windows ATASPI Manager. The function is passed a far pointer (lpARB) to an ATASPI Request Block (ARB). The following are the ATASPI commands supported and their symbolic names as defined in the included ATASPI.H at the end of this chapter:

- ATA Controller Inquiry (AC_CONTROLLER_INQUIRY)
- Get ATA Device Type (AC_GET_DEVICE_TYPE)
- Execute ATA I/O (AC_EXEC_ATA_CMD)
- Abort ATA Request (AC_ABORT_ARB)
- Reset ATA Device (AC_RESET_DEV)

Detailed description of each of the ATASPI commands are described in the next sections.

Return Values

Upon completion of the SendATASPICommand, it returns a WORD value that indicates the outcome of the ATASPI command. The following table contains all valid return values. Actual returned values differ for each ATASPI command. Refer to the specific ATASPI command section for valid returned values.

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Value (Hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS_PENDING</td>
<td>0x00</td>
<td>ARB being processed</td>
</tr>
<tr>
<td>AS_COMP</td>
<td>0x01</td>
<td>ARB completed without error</td>
</tr>
<tr>
<td>AS_ABORTED</td>
<td>0x02</td>
<td>ARB aborted</td>
</tr>
<tr>
<td>AS_ABORT_FAIL</td>
<td>0x03</td>
<td>Unable to abort ARB</td>
</tr>
<tr>
<td>AS_ERR</td>
<td>0x04</td>
<td>ARB completed with error</td>
</tr>
<tr>
<td>AS_INVALID_CMD</td>
<td>0x80</td>
<td>Invalid ATASPI command</td>
</tr>
<tr>
<td>AS_INVALID_CNUM</td>
<td>0x81</td>
<td>Invalid ATA controller number</td>
</tr>
<tr>
<td>AS_NO_DEVICE</td>
<td>0x82</td>
<td>ATA device not installed</td>
</tr>
<tr>
<td>AS_ATA_BUSY</td>
<td>0x83</td>
<td>ATA controller/device busy</td>
</tr>
<tr>
<td>AS_INVALID_ARB</td>
<td>0xE0</td>
<td>Invalid parameter set in ARB</td>
</tr>
<tr>
<td>AS_NO_WIN_MANAGER</td>
<td>0xE1</td>
<td>ATASPI manager doesn't support Windows</td>
</tr>
<tr>
<td>AS_ILLEGAL_MODE</td>
<td>0xE2</td>
<td>Unsupported Windows mode</td>
</tr>
<tr>
<td>AS_FAILED_INIT</td>
<td>0xE4</td>
<td>ATASPI for windows failed init</td>
</tr>
<tr>
<td>AS_ATASPI_IS_BUSY</td>
<td>0xE5</td>
<td>No resources available to execute cmd</td>
</tr>
<tr>
<td>AS_BUFFER_TOO_BIG</td>
<td>0xE6</td>
<td>Buffer size too big to handle</td>
</tr>
</tbody>
</table>

Table 16 - Valid SendATASPICommand Return Values

3.3. Windows ATASPI Commands

3.3.1. ATA Controller Inquiry Command

This command is used to retrieve information about a specific controller. It can also be used to get the total number of controllers supported by the Windows ATASPI Manager.
To issue this command, the pointer passed to SendATASPICommand must point to an ATA Controller Inquiry ARB as defined below:

```c
typedef struct {
    BYTE ARB_Cmd;
    BYTE ARB_Status;
    BYTE ARB_CntlrID;
    BYTE ARB_Flags;
    DWORD ARB_Reserved;
    BYTE CNTLR_Count;
    BYTE CNTLR_Reserved;
    BYTE CNTLR_ManagerID[16];
    BYTE CNTLR_Identifier[16];
    BYTE CNTLR_Unique[16];
} ARB_Controller_Inquiry;
```

The following table describes each of the fields in the ATA Controller Inquiry ARB:

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Offset</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARB_Cmd</td>
<td>00h</td>
<td>Out</td>
<td>This has the ATASPI Command Code. This must be equal to 00h.</td>
</tr>
<tr>
<td>ARB_Status</td>
<td>01h</td>
<td>In</td>
<td>This indicates the outcome of the command. Refer to the table below for valid values.</td>
</tr>
<tr>
<td>ARB_CntlrID</td>
<td>02h</td>
<td>Out</td>
<td>This refers to the zero-based controller ID to interrogate. If this value is OFFh, the field CNTLR_Count will have the total number of controllers supported.</td>
</tr>
<tr>
<td>ARB_Reserved</td>
<td>03h</td>
<td>-</td>
<td>Reserved. Must be zero.</td>
</tr>
<tr>
<td>CNTLR_Count</td>
<td>06h</td>
<td>In</td>
<td>This is the total number of controllers supported. This value is valid only if ARB_CntlrID on return is OFFh.</td>
</tr>
<tr>
<td>CNTLR_Reserved</td>
<td>06h</td>
<td>-</td>
<td>Reserved. Must be zero.</td>
</tr>
<tr>
<td>CNTLR_ManagerID[.]</td>
<td>0Ah</td>
<td>In</td>
<td>This is a 16-byte buffer containing an ASCII string describing the ATASPI Manager.</td>
</tr>
<tr>
<td>CNTLR_Identifier[.]</td>
<td>1Ah</td>
<td>In</td>
<td>This is a 16-byte buffer containing an ASCII string describing the controller specified by ARB_CntlrID.</td>
</tr>
<tr>
<td>CNTLR_Unique[.]</td>
<td>2Ah</td>
<td>In</td>
<td>This is a 16-byte buffer containing vendor unique information as defined by the controller vendor.</td>
</tr>
</tbody>
</table>

Table 17 - ATA Controller Inquiry ARB

Returns

Upon completion of the command, a WORD value containing the outcome of the command is returned. The following are the valid values returned:

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Value (in Hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS_COMP</td>
<td>01</td>
<td>Command completed without error</td>
</tr>
<tr>
<td>AS_INVALID_CNUM</td>
<td>81</td>
<td>Controller number passed is invalid</td>
</tr>
<tr>
<td>AS_INVALID_ARB</td>
<td>E0</td>
<td>ARB contains invalid parameter(s)</td>
</tr>
</tbody>
</table>

Table 18 - ATA Controller Inquiry ARB Return Values
Example Code

```c
ARB_Controller_Inquiry ATAControllerInquiryARB;
WORD ARBStatus;
:
:
// Make sure all reserved fields are initialized to zero
ATAControllerInquiryARB.ARB_Cmd = 0x00;
ATAControllerInquiryARB.ARB_CntlrID = 0;
ATAControllerInquiryARB.ARB_Flags = 0;
ARBStatus = SendATASPICommand((lпARB) &ATAControllerInquiryARB);
:
:
```

3.3.2. Get ATA Device Type Command

This command is used to obtain the device type of a specific target device. This command does not return detailed information about the device. It is recommended to issue an Execute ATA I/O Command for such purpose.

To issue this command, the pointer passed to SendATASPICommand must point to a Get ATA Device Type ARB defined as follows:

```c
typedef struct {
    BYTE ARB_Cmd;
    BYTE ARB_Status;
    BYTE ARB_CntlrID;
    BYTE ARB_Flags;
    DWORD ARB_Reserved;
    BYTE ARB_DeviceID;
    BYTE ARB_Reserved1;
    BYTE ARB_DeviceType;
} ARB_GDEVBlock;
```

The following table describes each of the fields in the Get ATA Device Type ARB:

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Offset</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARB_Cmd</td>
<td>00h</td>
<td>Out</td>
<td>This has the ATASPI Command Code. This must be equal to 01h.</td>
</tr>
<tr>
<td>ARB_Status</td>
<td>01h</td>
<td>In</td>
<td>This indicates the outcome of the command. Refer to the table below for valid values.</td>
</tr>
<tr>
<td>ARB_CntlrID</td>
<td>02h</td>
<td>Out</td>
<td>This refers to the zero-based controller ID that has the device to interrogate.</td>
</tr>
<tr>
<td>ARB_Reserved</td>
<td>03h</td>
<td>In</td>
<td>Reserved. Must be zero.</td>
</tr>
<tr>
<td>ARB_DeviceID</td>
<td>08h</td>
<td>Out</td>
<td>Device ID. Value set to 0 if Master device. Value set to 1 if Slave device.</td>
</tr>
<tr>
<td>ARB_Reserved</td>
<td>09h</td>
<td>In</td>
<td>Reserved. Must be zero.</td>
</tr>
<tr>
<td>ARB_DeviceType</td>
<td>0Ah</td>
<td>In</td>
<td>If ARB_Status is equal to AS_COMP, this field indicates the Peripheral Device Type.</td>
</tr>
</tbody>
</table>

Table 19 - Get ATA Device Type
The following table include all currently defined ATA device types and their device values used in ARB_DeviceType field.

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Value(Dec)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT_DirectAccess</td>
<td>00</td>
<td>Direct access device (e.g. magnetic disk)</td>
</tr>
<tr>
<td>DT_TapeQIC</td>
<td>01</td>
<td>Tape device (QIC-121 Model)</td>
</tr>
<tr>
<td>DT_CDROM</td>
<td>05</td>
<td>CD-ROM device</td>
</tr>
<tr>
<td>DT_Optical</td>
<td>07</td>
<td>Optical memory device</td>
</tr>
<tr>
<td>DT_TapeCS</td>
<td>0C</td>
<td>Tape device (Cost Sensitive Model)</td>
</tr>
<tr>
<td>DT_Unknown</td>
<td>1F</td>
<td>Unknown device</td>
</tr>
<tr>
<td>DT ATAType</td>
<td>80</td>
<td>Non-ATAPI device (IDE hard disk)</td>
</tr>
</tbody>
</table>

Table 20 - ATA Device Type Values

Returns

Upon completion of the command, a WORD value containing the outcome of the command is returned. The following are the valid values returned:

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Value (in Hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS_COMP</td>
<td>01</td>
<td>Command completed without error. The field ARB_DeviceType has a valid value.</td>
</tr>
<tr>
<td>AS_INVALID_CNUM</td>
<td>81</td>
<td>Controller number passed is invalid</td>
</tr>
<tr>
<td>AS_NO_DEVICE</td>
<td>82</td>
<td>Device is not present</td>
</tr>
<tr>
<td>AS_INVALID_ARB</td>
<td>E0</td>
<td>ARB contains invalid parameter(s)</td>
</tr>
</tbody>
</table>

Table 21 - Get ATA Device Type ARB Return Values

Example Code

```
ARB_GDEVBlock GetDeviceTypeARB;
WORD ARBStatus;
;
// Make sure all reserved fields are initialized to zero
GetDeviceTypeARB.ARB_Cmd = 0x01;
GetDeviceTypeARB.ARB_Cmd1 = 0;
GetDeviceTypeARB.ARB_Cmd2 = 0;
GetDeviceTypeARB.ARB_Cmd3 = 0;
ARBStatus = SendATASPICommand ((lpARB) &GetDeviceTypeARB);
;
```

3.3.3. Execute ATA I/O Command

*The Execute ATA I/O command is used to issue ATA and ATAPI commands to IDE devices. Multiple commands may be outstanding at any time. This multi-tasking capability is handled by the Windows ATAPI Manager. Completion of this command may be relayed to the caller either through a Polling method or a Posting method.*
To issue this command, the pointer passed to SendATASPICommand must point to one of the Execute ATA I/O ARBs defined below:

```c
// Structure for ATA Task File CDB
typedef struct {
    BYTE ARB_Cmd;
    BYTE ARB_Status;
    BYTE ARB_CntlrID;
    BYTE ARB_Flags;
    DWORD ARB_Reserved;
    BYTE ARB_DeviceID;
    BYTE ARB_Reserved1;
    DWORD ARB_BufLen;
    BYTE ARB_SenseLen;
    BYTE far * ARB_BufPointer;
    DWORD ARB_Reserved2;
    BYTE ARB_ACBLen;
    BYTE ARB_CntlrStat;
    BYTE ARB_DeVStat;
    FARPROC ARB_PostProc;
    WORD ARB_Data_BlkSize;
    BYTE ARB_Reserved3[32];
    BYTE ARB_ACBByte[7];
    BYTE ARB_SenseArea[SENSE_LEN];
} ARB_ExecATATFCmd;

// Structure for ATAPI 12 Byte ATAPI CDB
typedef struct {
    BYTE ARB_Cmd;
    BYTE ARB_Status;
    BYTE ARB_CntlrID;
    BYTE ARB_Flags;
    DWORD ARB_Reserved;
    BYTE ARB_DeviceID;
    BYTE ARB_Reserved1;
    DWORD ARB_BufLen;
    BYTE ARB_SenseLen;
    BYTE far * ARB_BufPointer;
    DWORD ARB_Reserved2;
    BYTE ARB_ACBLen;
    BYTE ARB_CntlrStat;
    BYTE ARB_DeVStat;
    FARPROC ARB_PostProc;
    WORD ARB_Data_BlkSize;
    BYTE ARB_Reserved3[32];
    BYTE ARB_ACBByte[12];
    BYTE ARB_SenseArea[SENSE_LEN];
} ARB_ExecATAPKT12Cmd;

// Note: For 16 byte ATAPI CDB commands, create an ARB with // ARB_ACBByte[16].
The following table describes each of the fields in the Execute ATA I/O ARB:

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Offset</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARB_Cmd</td>
<td>00h</td>
<td>Out</td>
<td>This has the ATASPI Command Code. This must be equal to 02h.</td>
</tr>
<tr>
<td>ARB_Status</td>
<td>01h</td>
<td>In</td>
<td>This indicates the outcome of the command. Refer to the table below for</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>valid values. See Flag definition that follows.</td>
</tr>
<tr>
<td>ARB_CntrlID</td>
<td>02h</td>
<td>Out</td>
<td>This refers to the zero-based controller ID of the IDE device.</td>
</tr>
<tr>
<td>ARB_Flags</td>
<td>03h</td>
<td>Out</td>
<td>This field tells the ATASPI Manager certain characteristics of this</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>request. Reserved. Must be zero.</td>
</tr>
<tr>
<td>ARB_Resid1</td>
<td>04h</td>
<td>-</td>
<td>Device ID. Value set to 0 if Master device. Value set to 1 if Slave</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>device.</td>
</tr>
<tr>
<td>ARB_BufLen</td>
<td>0Ah</td>
<td>Out</td>
<td>This refers to the data length in number of bytes. This field must be set</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>to zero if no data transfer is expected.</td>
</tr>
<tr>
<td>ARB_SenseLen</td>
<td>0Eh</td>
<td>Out</td>
<td>This refers to the number (N) of sense bytes to transfer when a Check</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Condition occurred.</td>
</tr>
<tr>
<td>ARB_BufPointer</td>
<td>0Fh</td>
<td>Out</td>
<td>This is the pointer to the data buffer. The buffer area must be in locked</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>memory.</td>
</tr>
<tr>
<td>ARB_Resid2</td>
<td>13h</td>
<td>-</td>
<td>Reserved. Must be zero.</td>
</tr>
<tr>
<td>ARB_ACBLen</td>
<td>17h</td>
<td>Out</td>
<td>This indicates the number (M) of valid bytes in the ARB_ACBByte field.</td>
</tr>
<tr>
<td>ARB_CntlrStat</td>
<td>18h</td>
<td>In</td>
<td>This field indicates the status of the controller upon completion of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>command.</td>
</tr>
<tr>
<td>ARB_DevStat</td>
<td>19h</td>
<td>In</td>
<td>This field is set to zero if no error occurred. Otherwise, this field will</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>have the value of the Error Register on the ATA bus. Refer to the ATAPI-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>or ATAPI Packet Interface specs for a description of the Error register.</td>
</tr>
<tr>
<td>ARB_PostProc</td>
<td>1Ah</td>
<td>Out</td>
<td>If the Post bit is set on the ARB_Flags byte, this field contains the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>address to call upon completion of the command.</td>
</tr>
<tr>
<td>ARB_Data_BlkSize</td>
<td>1Eh</td>
<td>Out</td>
<td>The ARB_Data_BlkSize field is used to specify the number of data bytes to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>transfer per hardware interrupt. See detailed explanation that follows.</td>
</tr>
<tr>
<td>ARB_Resid3</td>
<td>20h</td>
<td>-</td>
<td>Reserved. Must be zero.</td>
</tr>
<tr>
<td>ARB_ACBByte[M]</td>
<td>40h</td>
<td>Out</td>
<td>If Request Type is an task file request, this area contains the 7 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Task File command. Otherwise, this area contains the command bytes of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ATAPI device.</td>
</tr>
<tr>
<td>ARB_SenseArea[N]</td>
<td>10h+M</td>
<td>In</td>
<td>This area will contain the sense data if a Check Condition occurred.</td>
</tr>
</tbody>
</table>

Table 22 - Execute ATA I/O ARB

The ARB_Flags byte is defined as follows:

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resvd</td>
<td>ByteXfer</td>
<td>DUA Bit</td>
<td>Direction Bits</td>
<td>Request Type</td>
<td>Resvd</td>
<td>Post</td>
<td></td>
</tr>
</tbody>
</table>

Table 23 - ARB_Flags

The Post bit indicates whether we need to call a Post routine upon completion of the request.

0  Disable Posting
1  Enable Posting

The Request Type bit specifies whether ARB contains an ATAPI Packet command or an AT Task File Structure:

0  ATAPI Packet command (e.g. Inquiry, Mode Select ...)
1  AT Task File Structure
The **Direction** bits specify the data transfer directions:

- 00 Direction determined by device
- 01 Data In
- 10 Data Out
- 11 No Data Transfer

If the command sent out involves data transfer, setting the Direction bits to *Data In* or *Data Out* is **required**. No Data Transfer option can only be used with commands that have no data transfer such as Seek, Execute Device Diagnostic... [See Data Transfer Length field for additional data transfer capabilities].

The **DUA Bit** (DSC Unavailable Action) for an ATAPI packet command determines what action the ATASPI manager is to take if the DSC Bit is not set in the task file status register. The ATASPI manager is expecting the DSC bit to be set before sending an ATAPI command block, the caller can specify which action to take if DSC bit not set.

- 0 The ATASPI manager must queue this ATAPI request and service it later when DSC bit is set.
- 1 The ATASPI manager must return to the caller with status set to a BUSY condition. It is the responsibility of the caller to issue the request at a latter time.

The **ByteXfer** bit indicates that byte data transfer is requested. The default mode of operation for ATASPI Manager is word transfer.

- 0 use word transfer mode
- 1 use byte transfer mode

The **ARB_Data_BlkSize** field is used to specify the number of data bytes to transfer per hardware interrupt. For Task File request, Data Transfer Block Size (DTBS) is used to determine maximum number of data bytes to transfer per interrupt. The default value used by the ATASPI manager (when DTBS = 0) is 512 bytes.

For ATAPI Packet commands, Data Transfer Block Size (DTBS) is used to tell the device how much data the host prefers to transfer per interrupt. The default value used by ATASPI (when DTBS = 0) is 930h bytes.

If necessary (for Read/Write Multiple ...), this could be set to a different value. This is only a preferred value; for every interrupt received, the device specifies the exact amount of data to transfer.
Returns

Upon completion of the command, a WORD value containing the status of the command is returned. The following are the valid values returned:

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Value (in Hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS_PENDING</td>
<td>00</td>
<td>The command is in progress.</td>
</tr>
<tr>
<td>AS_CMP</td>
<td>01</td>
<td>Command completed without error.</td>
</tr>
<tr>
<td>AS_ABORTED</td>
<td>02</td>
<td>ATASPI command is aborted.</td>
</tr>
<tr>
<td>AS_ERR</td>
<td>04</td>
<td>Command completed with error.</td>
</tr>
<tr>
<td>AS_INVALID_CNUM</td>
<td>81</td>
<td>Controller number passed is invalid.</td>
</tr>
<tr>
<td>AS_NO_DEVICE</td>
<td>82</td>
<td>Device is not present.</td>
</tr>
<tr>
<td>AS_ATA_BUSY</td>
<td>83</td>
<td>ATA Controller or Device is busy.</td>
</tr>
<tr>
<td>AS_INVALID_ARB</td>
<td>E0</td>
<td>ARB contains invalid parameter(s).</td>
</tr>
<tr>
<td>AS_ATASPI_IS_BUSY</td>
<td>E5</td>
<td>ATASPI cannot handle the request at this time.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Re-send the request later.</td>
</tr>
<tr>
<td>AS_BUFFER_TOO_BIG</td>
<td>E6</td>
<td>Transfer size too big for ATASPI Manager.</td>
</tr>
</tbody>
</table>

Table 24 - Execute ATA I/O ARB Return Values

Example Code

```c
ARB_ExecATATFCmd ExecATAioARB;
WORD ARBStatus;
char IdentifyDriveData[512];
;
// Make sure all reserved fields are initialized to zero
ExecATAioARB.ARB_Cmd = 0x02;
ExecATAioARB.ARB_Cmd = 0;
ExecATAioARB.ARB_Cmd = 0x05;
ExecATAioARB.ARB_Cmd = 512;
ExecATAioARB.ARB_Cmd = SENSE_LEN;
ExecATAioARB.ARB_Cmd = IdentifyDriveData;
ExecATAioARB.ARB_Cmd = 7;
ExecATAioARB.ARB_Cmd = lpfnPostFunction;
ExecATAioARB.ARB_Cmd = 0;
ExecATAioARB.ARB_Cmd = 0;
ExecATAioARB.ARB_Cmd = 0;
ExecATAioARB.ARB_Cmd = 0;
ExecATAioARB.ARB_Cmd = 0;
ExecATAioARB.ARB_Cmd = 0xEC;
ARBStatus = SendATASPICmd((lpARB) & ExecATAioARB);
;
3.3.4. Abort ATA Request Command
```

This command is used to abort a previously issued ARB. The success of this command is never assured. This command will always return success.
To issue this command, the pointer passed to SendATASPICommand must point to an Abort
ATA Request ARB as defined below:

typedef struct {
    BYTE ARB_Cmd;
    BYTE ARB_Status;
    BYTE ARB_CntlrID;
    BYTE ARB_Flags;
    DWORD ARB_Reserved;
    lpARB ARB_ToAbort;
} ARB_Abort;

The following table describes each of the fields in the Abort ATA I/O ARB:

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Offset</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARB_Cmd</td>
<td>00h</td>
<td>Out</td>
<td>This has the ATASPI Command Code. This must be equal to 03h.</td>
</tr>
<tr>
<td>ARB_Status</td>
<td>01h</td>
<td>In</td>
<td>This indicates the outcome of the command. Refer to the table below for valid values.</td>
</tr>
<tr>
<td>ARB_CntlrID</td>
<td>02h</td>
<td>Out</td>
<td>This refers to the zero-based ID of the controller that owns the ARB to abort.</td>
</tr>
<tr>
<td>ARB_Reserved</td>
<td>03h</td>
<td>-</td>
<td>Reserved: Must be zero.</td>
</tr>
<tr>
<td>ARB_ToAbort</td>
<td>06h</td>
<td>Out</td>
<td>This field has the pointer to the ARB to be aborted.</td>
</tr>
</tbody>
</table>

Table 25 - Abort ATA Request Command

Returns

Upon completion of the command, a WORD value containing the outcome of the command
is returned. The following are the valid values returned:

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Value (in Hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS_COMP</td>
<td>01</td>
<td>Command completed without error. The field ARB_DeviceType has a valid value.</td>
</tr>
<tr>
<td>AS_INVALID_CNUM</td>
<td>81</td>
<td>Controller number passed is invalid</td>
</tr>
<tr>
<td>AS_INVALID_ARB</td>
<td>E0</td>
<td>The ARB contains invalid parameter(s).</td>
</tr>
</tbody>
</table>

Table 26 - Abort ATA Request Command Return Values

Example Code

ARB_ExecuteATASPICommand ARBtoAbort;
ARB_Abort AbortARB;
WORD ARB_Status;
;
;
// Make sure all reserved fields are initialized to zero
AbortARB.ARB_Cmd = 0x03;
AbortARB.ARB_CntlrID = 0;
AbortARB.ARB_Flags = 0;
AbortARB.ARB_ToAbort = (lpARB) & ARBtoAbort;
ARB_Status = SendATASPICommand ((lpARB) & AbortARB);
while (ResetDevARB.ARB_Status == AS_PENDING); // Poll for completion
;

Working Draft Version 0.72
3.3.5. Reset ATA Device Command

This command is used to reset a specific device. The consequence of this command is dependent on the type of device to be reset. If the specified device does not support a packetized protocol, e.g. an IDE hard disk, the SRST bit in the Device Control Register (refer to the ATA-2 spec for more information) will be asserted. This will cause all devices in the same bus to be reset. If the device is an ATAPI (packetized protocol) device, an ATAPI Soft Reset (ATA Code 08h) will be issued and will cause only the specified device to be reset.

To issue this command, the pointer passed to SendATASPCOMmand must point to a Reset ATA Device ARB as defined below:

typedef struct {
    BYTE ARB_Cmd;
    BYTE ARB_Status;
    BYTE ARB_cntlrID;
    BYTE ARB_Flags;
    DWORD ARB_Reservw;
    BYTE ARB_DeviceID;
    BYTE ARB_Reserv1[15];
    BYTE ARB_CntlrStat;
    BYTE ARB_DevStat;
    FARPROC ARB_PostProc;
    BYTE ARB_Reserv2[34];
} ARB_DeviceReset;

The following table describes each of the fields in the Reset ATA Device ARB:

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Offset</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARB_Cmd</td>
<td>00h</td>
<td>Out</td>
<td>This has the ATAPI Command Code. This must be equal to 04h.</td>
</tr>
<tr>
<td>ARB_Status</td>
<td>01h</td>
<td>In</td>
<td>This indicates the outcome of the command. Refer to the table below for valid values.</td>
</tr>
<tr>
<td>ARB_cntlrID</td>
<td>02h</td>
<td>Out</td>
<td>This refers to the zero-based controller ID.</td>
</tr>
<tr>
<td>ARB_Flags</td>
<td>03h</td>
<td>Out</td>
<td>This field tells the ATAPI Manager certain characteristics of this request. See the flag definition that follows.</td>
</tr>
<tr>
<td>ARB_Reserv</td>
<td>04h</td>
<td>Reserved</td>
<td>Reserved. Must be zero.</td>
</tr>
<tr>
<td>ARB_DeviceID</td>
<td>08h</td>
<td>Out</td>
<td>Device ID. Value set to 0 if Master device. Value set to 1 if Slave device.</td>
</tr>
<tr>
<td>ARB_Reserv1</td>
<td>09h</td>
<td>Reserved</td>
<td>Reserved. Must be zero.</td>
</tr>
<tr>
<td>ARB_CntlrStat</td>
<td>0Ah</td>
<td>In</td>
<td>This field indicates the status of the controller upon completion of the command. The following are the valid values: ACSTAT_OK - No error ACSTAT_NO_DEVICE - Device not present</td>
</tr>
<tr>
<td>ARB_DevStat</td>
<td>0Bh</td>
<td>In</td>
<td>This field is set to zero if no error occurred. Otherwise, this field will have the value of the Error Register on the ATA bus. Refer to the ATA-2 or ATA Packet Interface specs for a description of the Error register.</td>
</tr>
<tr>
<td>ARB_PostProc</td>
<td>1Ah</td>
<td>Out</td>
<td>If the Post bit is set on the ARB_Flags byte, this field contains the address to call upon completion of the command.</td>
</tr>
<tr>
<td>ARB_Reserv2</td>
<td>1Bh</td>
<td>Reserved</td>
<td>Reserved. Must be zero.</td>
</tr>
</tbody>
</table>

Table 27 - Reset ATA Device ARB
Returns

Upon completion of the command, a WORD value containing the outcome of the command is returned. The following are the valid values returned:

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Value (in Hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS_COMP</td>
<td>01</td>
<td>Command completed without error. The field ARB_DeviceType has a valid value.</td>
</tr>
<tr>
<td>AS_INVALID_CNUM</td>
<td>81</td>
<td>Controller number passed is invalid</td>
</tr>
<tr>
<td>AS_INVALID_ARB</td>
<td>E0</td>
<td>ARB contains invalid parameter(s)</td>
</tr>
<tr>
<td>AS_ATASPI_IS_BUSY</td>
<td>E5</td>
<td>ATASPI cannot handle the request at this time. Re-send the request later.</td>
</tr>
</tbody>
</table>

Table 28 - Reset ATA Device ARB Return Values

Example Code

ARB_DeviceReset ResetDevARB;
WORD ARBStatus;
 :
 :
// Make sure all reserved fields are initialized to zero
ResetDevARB.ARB_Cmd    = 0x04;
ResetDevARB.ARB_CntlID = 0;
ResetDevARB.ARB_Flags  = 0;
ResetDevARB.DeviceID   = 0;
ARBStatus = SendATASPICommand ((lpARB) &ResetDevARB);
while (ResetDevARB.ARB_Status == AS_PENDING);
  // Poll for completion
  :
  :
3.3.5. Reset ATA Device Command

This command is used to reset a specific device. The consequence of this command is dependent on the type of device to be reset. If the specified device does not support a packetized protocol, e.g. an IDE hard disk, the SRST bit in the Device Control Register (refer to the ATA-2 spec for more information) will be asserted. This will cause all devices in the same bus to be reset. If the device is an ATAPI (packetized protocol) device, an ATAPI Soft Reset (ATA Code 08h) will be issued and will cause only the specified device to be reset.

To issue this command, the pointer passed to SendATASPICommand must point to a Reset ATA Device ARB as defined below:

```c
typedef struct {
    BYTE ARB_Cmd;
    BYTE ARB_Status;
    BYTE ARB_CntlzID;
    BYTE ARB_Flags;
    DWORD ARB_Resvd;
    BYTE ARB_DeviceID;
    BYTE ARB_Resvd1[15];
    BYTE ARB_CntlzStat;
    BYTE ARB_DevStat;
    FARPROC ARB_PostProc;
    BYTE ARB_Resvd2[34];
} ARB_DeviceReset;
```

The following table describes each of the fields in the Reset ATA Device ARB:

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Offset</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARB_Cmd</td>
<td>00h</td>
<td>Out</td>
<td>This has the ATASPI Command Code. This must be equal to 04h.</td>
</tr>
<tr>
<td>ARB_Status</td>
<td>01h</td>
<td>In</td>
<td>This indicates the outcome of the command. Refer to the table below for valid values.</td>
</tr>
<tr>
<td>ARB_CntlzID</td>
<td>02h</td>
<td>Out</td>
<td>This refers to the zero-based controller ID.</td>
</tr>
<tr>
<td>ARB_Flags</td>
<td>03h</td>
<td>Out</td>
<td>This field tells the ATASPI Manager certain characteristics of this request. See the flag definition that follows.</td>
</tr>
<tr>
<td>ARB_Resvd</td>
<td>04h</td>
<td>Reserved</td>
<td>Must be zero.</td>
</tr>
<tr>
<td>ARB_DeviceID</td>
<td>08h</td>
<td>Out</td>
<td>Device ID. Value set to 0 if Master device. Value set to 1 if Slave device.</td>
</tr>
<tr>
<td>ARB_Resvd1</td>
<td>09h</td>
<td>Reserved</td>
<td>Must be zero.</td>
</tr>
<tr>
<td>ARB_CntlzStat</td>
<td>18h</td>
<td>In</td>
<td>This field indicates the status of the controller upon completion of the command. The following are the valid values: ACSTAT_OK - No error ACSTAT_NO_DEVICE - Device not present</td>
</tr>
<tr>
<td>ARB_DevStat</td>
<td>19h</td>
<td>In</td>
<td>This field is set to zero if no error occurred. Otherwise, this field will have the value of the Error Register on the ATA bus. Refer to the ATA-2 or ATA Packet Interface specs for a description of the Error register.</td>
</tr>
<tr>
<td>ARB_PostProc</td>
<td>1Ah</td>
<td>Out</td>
<td>If the Post bit is set on the ARB_Flags byte, this field contains the address to call upon completion of the command.</td>
</tr>
<tr>
<td>ARB_Resvd2</td>
<td>1Eh</td>
<td>Reserved</td>
<td>Must be zero.</td>
</tr>
</tbody>
</table>

Table 27 - Reset ATA Device ARB
Returns

Upon completion of the command, a WORD value containing the outcome of the command is returned. The following are the valid values returned:

<table>
<thead>
<tr>
<th>Symbolic Name</th>
<th>Value (in Hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS_Comp</td>
<td>01</td>
<td>Command completed without error. The field ARB_DeviceType has a valid value.</td>
</tr>
<tr>
<td>AS_INVALID_CNUM</td>
<td>81</td>
<td>Controller number passed is invalid</td>
</tr>
<tr>
<td>AS_INVALID_ARB</td>
<td>E0</td>
<td>ARB contains invalid parameter(s).</td>
</tr>
<tr>
<td>AS_ATASPI_IS_BUSY</td>
<td>E5</td>
<td>ATASPI cannot handle the request at this time. Re-send the request later.</td>
</tr>
</tbody>
</table>

Table 28 - Reset ATA Device ARB Return Values

Example Code

```c
ARB_DeviceReset ResetDevARB;
WORD ARBStatus;

:
:

// Make sure all reserved fields are initialized to zero
ResetDevARB.ARB_Cmd = 0x04;
ResetDevARB.ARB_CntlID = 0;
ResetDevARB.ARB_Flags = 0;
ResetDevARB.DeviceID = 0;
ARBStatus = SendATASPICommand ((lpARB) &ResetDevARB);
while (ResetDevARB.ARB_Status == AS_PENDING);

// Poll for completion
:
:
```
3.4. ATASPI.H

typedef struct {
    BYTE ARB_Cmd;  // ATASPI command code
    BYTE ARB_Status;  // ATASPI command status byte
    BYTE ARB_CmdID;  // ATASPI ATA controller number
    BYTE ARB_Flags;  // ATASPI request flags
    DWORD ARB_Reserve;  // Reserved, MUST = 0
} ARB_Struct, FAR * ARB_Struct_Ptr;

typedef ARB_Struct far * lPARB;

WORD FAR PASCAL SendATASPICommand (lPARB);
WORD FAR PASCAL GetATASPISupportInfo (VOID);

#define SENSE_LEN  14  // Default sense buffer length

#define AC_CONTROLLER_INQUIRY  0x00  // ATA Controller Inquiry
#define AC_SET_DEV_TYPE  0x01  // Get ATA Device Type
#define AC_EXECC command  0x02  // Execute ATA Command
#define AC_ABORT_ARB  0x03  // Abort an ARB
#define AC_RESET_DEV  0x04  // ATA Bus Device Reset

#define AF_DIR_APA  0x20  // Direction determined by APA command
#define AF_DIR_IN  0x28  // Transfer from APA device to controller
#define AF_DIR_OUT  0x29  // Transfer from controller to APA device
#define AF_ABORTING 0x2a  // Enable APA aborting
#define AF_TASK_REQ 0x2b  // Task file request
#define AF_BYTE_XFR  0x40  // Byte transfers requested
#define AF_DUA  0x42  // DSC Unavailable Action Bit

#define AS_PENDING  0x00  // ARB being processed
#define AS_COMPLETED 0x01  // ARB completed; no operation specified
#define AS_ABORTED 0x02  // ARB aborted
#define AS_ABORT_FAIL 0x11  // Unable to abort ARB
#define AS_ERR  0x04  // ARB completed; error
#define AS_INVALID_CMD 0x90  // Invalid ATASPI command
#define AS_INVALID_CNTR 0x91  // Invalid ATA controller number
#define AS_NO_DEVICE 0x80  // ATA device not installed
#define AS_ABORT_FAIL 0x83  // ATA controller/device busy
#define AS_INVALID_ARB 0x90  // Invalid parameter set in ARB
#define AS_NO_WWN_MGR 0x81  // ATASPI manager doesn't support Windows
#define AS_ILLEGAL_MODE 0x82  // Unsupported Windows mode
#define AS_FAILED_INIT 0x84  // ATASPI for windows failed init
#define AS_ATASPI_BUSY 0x85  // No resources available to execute cmd
#define AS_BUFFER_TOO_BIG 0x86  // Buffer size too big to handle.
ATA Software Programming Interface (ATASPI) Specification
for DOS and Windows Operating Systems

#define ACSTAT_OK 0x00 // ATA controller did not detect an error
#define ACSTAT_NO_DEVICE 0x11 // Device not present
#define ACSTAT_DQ_OU 0x12 // Data overrun, data underrun

#define DT_DIRECT_ACCESS 0x00 // Direct access device (e.g., magnetic disk)
#define DT_TAPEQIC 0x01 // Tape device (QIC-121 Model)
#define DT_CDROM 0x05 // CD-ROM device
#define DT_OPTICAL 0x07 // Optical memory device
#define DT_TAPECS 0x0C // Tape device (Cost Sensitive Model)
#define DT_UNKNOWN 0x1F // Unknown device type
#define DT_ATATYPE 0x80 // Non-ATAPI device (IDE hard disk)

typedef struct {
    BYTE ARB_Cmd; // ATASPI command code = AC_CONTROLLER_INQUIRY
    BYTE ARB_Status; // ATASPI command status byte
    BYTE ARB_CntlrID; // ATASPI ATA Controller number
    BYTE ARB_Flags; // ATASPI request flags
    DWORD ARB_Reserved; // Reserved, MUST = 0
    BYTE CNTLR_Count; // Number of ATA Controllers present
    BYTE CNTLR_Selected; // ATA ID of ATA Controller
    BYTE CNTLR_Manager[16]; // String describing the manager
    BYTE CNTLR.Identifier[16]; // String describing the ATA Controller
    BYTE CNTLR.Unique[16]; // ATA Controller Unique parameters
} ARB_Controller_Inquiry, FAR * ARB_Controller_Inquiry_Ptr;

typedef struct {
    BYTE ARB_Cmd; // ATASPI command code = AC_GET_DEV_TYPE
    BYTE ARB_Status; // ATASPI command status byte
    BYTE ARB_CntlrID; // ATASPI ATA Controller number
    BYTE ARB_Flags; // ATASPI request flags
    DWORD ARB_Reserved; // Reserved, MUST = 0
    BYTE ARB_DeviceID; // Device's ATA ID
    BYTE ARB_Reserved1; // Reserved, MUST = 0
    BYTE ARB_DeviceType; // Device's peripheral device type
} ARB_GDEVBlock, FAR * ARB_GDEVBlock_Ptr;

typedef struct {
    BYTE ARB_Cmd; // ATASPI command code = AC_EXEC_ATA_CMD
    BYTE ARB_Status; // ATASPI command status byte
    BYTE ARB_CntlrID; // ATASPI ATA Controller number
    BYTE ARB_Flags; // ATASPI request flags
    DWORD ARB_Reserved; // Reserved, MUST = 0
    BYTE ARB_DeviceID; // Target's ATA ID
    BYTE ARB_Reserved1; // Reserved, MUST = 0
    DWORD ARB_BufferLen; // Data Allocation Length
    BYTE ARB_BufferLen; // Data Buffer Pointer
    BYTE * ARB_Buffer; // Data Buffer Pointer
    DWORD ARB_Reserved2; // Reserved, MUST = 0
    BYTE ARB_ACBlen; // ATA ACB Length = 12
    BYTE ARB_CmdStat; // ATA Command Status
    BYTE ARB_DevStat; // Target Status
    FARPROC ARB_Protocols; // Protocols
    WORD ARB_Spi_BlkSize; // Data Block Size
    BYTE ARB_Reserved3[32]; // Reserved, MUST = 0
    BYTE ARB_ACBByte[7]; // ATA Task File Bytes
} ARB_Command, FAR * ARB_Command_Ptr;
BYTE ARB_SenseArea[SENSE_LEN]; // Request Sense buffer

} ARB_ExecATAPIKCmd, FAR * ARB_ExecATAPIKCmd_Ptr;

typedef struct { // Structure for 12-byte ATAPI CBs
    BYTE ARB_Cmd; // ATAPI command code = AC_EXEC_ATA_CMD
    BYTE ARB_Status; // ATAPI command status byte
    BYTE ARB_CntlID; // ATAPI ATA Controller number
    BYTE ARB_Flags; // ATAPI request flags
    DWORD ARB_Res0; // Reserved, MUST = 0
    BYTE ARB_DeviceID; // Target's ATA ID
    BYTE ARB_Res1; // Reserved, MUST = 0
    DWORD ARB_Res2; // Data Allocation Length
    BYTE ARB_SenseLen; // Sense Allocation Length
    BYTE far *ARB_Ptr; // Data Buffer Pointer
    DWORD ARB_ACBlen; // ATA CDB Length = 12
    BYTE ARB_CntlStat; // ATA Controller Status
    BYTE ARB_Ptr; // Post routine
    DWORD ARB_Res3; // Reserved, MUST = 0
    BYTE ARB_ACBBYte[12]; // ATAPI Command Block Bytes
    BYTE ARB_SenseArea[SENSE_LEN]; // Request Sense buffer
} ARB_ExecATAPIKCmd, FAR * ARB_ExecATAPIKCmd_Ptr;

} ARB_ExecATAPIKCmd, FAR * ARB_ExecATAPIKCmd_Ptr;

typedef struct { // Structure for 16-byte ATAPI CBs
    BYTE ARB_Cmd; // ATAPI command code = AC_EXEC_ATA_CMD
    BYTE ARB_Status; // ATAPI command status byte
    BYTE ARB_CntlID; // ATAPI ATA Controller number
    BYTE ARB_Flags; // ATAPI request flags
    DWORD ARB_Res0; // Reserved, MUST = 0
    BYTE ARB_DeviceID; // Target's ATA ID
    BYTE ARB_Res1; // Reserved, MUST = 0
    DWORD ARB_Res2; // Data Allocation Length
    BYTE ARB_SenseLen; // Sense Allocation Length
    BYTE far *ARB_Ptr; // Data Buffer Pointer
    DWORD ARB_ACBlen; // ATA CDB Length = 12
    BYTE ARB_CntlStat; // ATA Controller Status
    BYTE ARB_Ptr; // Post routine
    DWORD ARB_Res3; // Reserved, MUST = 0
    BYTE ARB_ACBBYte[16]; // ATAPI Command Block Bytes
    BYTE ARB_SenseArea[SENSE_LEN]; // Request Sense buffer
} ARB_ExecATAPIKCmd, FAR * ARB_ExecATAPIKCmd_Ptr;

} ARB_ExecATAPIKCmd, FAR * ARB_ExecATAPIKCmd_Ptr;

// *** ARB - ABORT AN ARB - AC_ABORT_ARE ***
typedef struct {
    BYTE ARB_Cmd; // ATAPI command code = AC_ABORT_ARE
    BYTE ARB_Status; // ATAPI command status byte
    BYTE ARB_CntlID; // ATAPI ATA Controller number
    BYTE ARB_Flags; // ATAPI request flags
    DWORD ARB_Res0; // Reserved, MUST = 0
    lPARB ARB_ToAbort; // Pointer to ARB to abort
} ARB_Abort, FAR * ARB_Abort_Ptr;

} ARB_Abort, FAR * ARB_Abort_Ptr;

// *** ARB - DEVICE RESET - AC_RESET_DEV ***
typedef struct {
    BYTE ARB_Cmd; // ATAPI command code = AC_RESET_DEV
    BYTE ARB_Status; // ATAPI command status byte
    BYTE ARB_CntlID; // ATAPI ATA Controller number
    BYTE ARB_Flags; // ATAPI request flags

Working Draft Version 0.72 30
DWord  ARB_Reserved;          // Reserved, MUST = 0
BYTE   ARB_DeviceID;         // Device's ATA ID
BYTE   ARB_Reserved1[15];    // Reserved, MUST = 0
BYTE   ARB_InitStat;         // ATA Controller Status
BYTE   ARB_TargetStat;       // Device Status
PAXPROC ARB_PostProc;         // Post routine
BYTE   ARB_Reserved2[34];    // Reserved, MUST = 0

! ARB_DeviceReset, FAR * ARB_DeviceReset_Ptr;
4. Document History

Version 0.50 (November 2, 1994) Initial draft

Version 0.60 (November 11, 1994) Correction of initial entry errors.

Version 0.70 (November 19, 1994) Added ATASPI For Windows Specification Section 4

Version 0.71 (November 23, 1994) Request Flags bit 5, DCS bit changed definition to DUA bit.

Version 0.72 (December 20, 1994) Request Flags bit 5, Remove reference to Special Block Size
Modify DUA bit description
General Editorial Cleanup