Developer Note

PowerBook 150

Developer Press

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

If you discover physical defects in the manual or in the media on which a software product is distributed, APDA will replace the media or manual at no charge to you provided you return the item to be replaced with proof of purchase to APDA.

ALL IMPLIED WARRANTIES ON THIS MANUAL, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Figures and Tables

About This Developer Note

This document describes the Macintosh JeDI computer, emphasizing those features that are new or different from other Macintosh PowerBook computers. It is written primarily for experienced Macintosh hardware and software developers who want to create products that are compatible with these new computers. If you are unfamiliar with Macintosh computers or would simply like more technical information, you may want to read the related technical manuals listed in "Supplementary Documents."

Contents of This Note

This developer note is divided into three chapters:

Chapter 1, "PowerBook 150 System Overview," gives a summary of the features of the JeDI computer.

Chapter 2, "IDE Hard Drive Interface," provides a description of the ATA IDE interface connector for the JeDI IDE internal hard disk drive.

Chapter 3, "Software for the ATA / IDE Hard Disk," describes the IDE device driver and IDE Manager software for controlling an IDE hard disk drive installed in the JeDI computer.

An index is also included.

Supplementary Documents

To supplement the information in this document, you might wish to obtain related documentation such as *Guide to the Macintosh Family Hardware*, second edition; *Designing Cards and Drivers for the Macintosh Family*, third edition; and *Inside Macintosh*. For detailed information about the Motorola 68030 microprocessor used in these computers, refer to the *MC68030 Enhanced 32-Bit Microprocessor User's Manual*. All of these documents are available through APDA.

APDA is Apple's worldwide source for hundreds of development tools, technical resources, training products, and information for anyone interested in developing applications on Apple platforms. Customers receive the *APDA Tools Catalog* featuring all current versions of Apple development tools and the most popular third-party development tools. APDA offers convenient payment and shipping options, including site licensing.

To order products or to request a complimentary copy of the *APDA Tools Catalog*, contact

APDA
Apple Computer, Inc.
P.O. Box 319
Buffalo, NY 14207-0319

Telephone	1-800-282-2732 (United States) 1-800-637-0029 (Canada) 716-871-6555 (International)
Fax	716-871-6511
AppleLink	APDA
America Online	APDAorder
CompuServe	76666,2405
Internet	APDA@applelink.apple.com

Conventions and Abbreviations

This developer note uses typographical conventions and abbreviations that are standard in Apple publications.

Typographical Conventions

Computer-language text—any text that is literally the same as it appears in computer input or output—appears in `Courier` font.

Hexadecimal numbers are preceded by a dollar sign ($). For example, the hexadecimal equivalent of decimal 16 is written as $10.

Standard Abbreviations

Standard units of measure used in this developer note include:

A	amperes	MB	megabytes
GB	gigabytes	MHz	megahertz
Hz	hertz	ms	milliseconds
K	1024	ns	nanoseconds
KB	kilobytes	V	volts
mA	milliamperes	W	watts

Standard abbreviations used in this developer note include:

$*n*	hexadecimal value *n*
AC	alternating current
ADB	Apple Desktop Bus
ASC	Apple Sound Chip
ASIC	application-specific integrated circuit
DAC	digital-to-analog converter
FSTN	film supertwist nematic (a type of LCD)
IC	integrated circuit
IDE	integrated device electronics
LCD	liquid crystal display
RAM	random-access memory
RAMDAC	random-access memory, digital-to-analog converter
ROM	read-only memory
RGB	red-green-blue (a video display system used by Apple computers)
SCSI	Small Computer System Interface
SVGA	super VGA (a video display system used with PC-type computers)
SWIM	Super Woz Integrated Machine
TFT	thin film transistor (a type of LCD)
VGA	video graphics adapter
VRAM	video RAM

CHAPTER 1

PowerBook 150 System Overview

The PowerBook 150 computer retains the physical appearance of the PowerBook 145B computer except that it houses a logic board built upon the PowerBook 200 series system architecture. This new logic board not only increases the processing performance of the PowerBook 150, but also allows for greater flexibility of RAM expansion. In addition to the increased performance and new RAM expansion capabilities, the PowerBook 150 also incorporates an internal IDE drive, rather than an internal SCSI drive. Support for external SCSI peripherals is still built into the computer.

This developer note describes the major features of the PowerBook 150 computer, emphasizing compatibility issues and expansion opportunities for developers.

IMPORTANT
Only the major differences between the PowerBook 150 and the PowerBook 145B are described in detail here. For a complete understanding of the PowerBook 150 computer, refer to the *Macintosh PowerBook 145B Developer Note*, the *Macintosh Developer Notes: Number 2*, and the *Macintosh PowerBook 140 and Macintosh PowerBook 170 Developer Note*, available from APDA. ▲

Features

The major features of the PowerBook 150 computer are:

- Microprocessor: Motorola 68030 running at 33 MHz. A 16 MHz power saving mode can be selected by the user. The PowerBook 150 does not include a math coprocessor.
- Read-only memory (ROM): 1 MB.
- Random-access memory (RAM): 4 MB of dynamic RAM (DRAM).
- RAM expansion: a RAM expansion slot accommodates Macintosh Duo system RAM expansion cards up to a total of 40 MB of RAM.
- Liquid crystal display: film super twist nematic (FSTN) 2-bit-per-pixel (4-level) grayscale, 640-by-480 pixels, with adjustable backlighting.
- Floppy disk: one internal 1.44 MB Apple SuperDrive with Super Woz Integrated Machine (SWIM) interface.
- Hard disk: one 120 MB internal 2.5-inch IDE hard disk drive.
- I/O (input/output): one HDI-30 connector for external SCSI devices and one 8-pin mini-DIN serial port.
- Sound: audio circuitry provides sound output through the built-in speaker. The PowerBook 150 does not support sound input.
- Keyboard: built-in keyboard with 3 mm of key travel.
- Trackball: built-in 300 mm trackball positioned below keyboard.
- Modem: internal 20-pin connector for an optional modem card. This slot accepts any fax/data modem card compatible with the PowerBook 145B. The PowerBook 150 does not support the PowerBook Express Modem.

- Battery: a rechargeable NiCad battery is included. A 2.4 volt battery provides backup power for the real-time clock and parameter RAM when the main battery is removed.
- Power supply: an external wall-mounted recharger / power adapter is included.
- Weight: 5.5 pounds.
- Size: 11.25 inches wide, 9.3 inches deep, and 2.25 inches high.

Compatibility Issues

This section highlights key areas you should investigate in order to ensure that your products work properly with the PowerBook 150 computer.

IDE Internal Drive Support

The PowerBook 150 computer incorporates an internal IDE (integrated device electronics) hard drive. This is a standard 2.5-inch IDE hard disk drive. The IDE drive is placed in the same mounting envelope in which the internal SCSI drive is located in the PowerBook 145B. A 44-pin connector is used to connect the drive to the main logic board. The IDE hardware interface to the main logic board is described in Chapter 2, “IDE Hard Drive Interface.”

Only one IDE drive can be connected to the PowerBook 150. The IDE drive must be the internal drive. You can connect additional external SCSI hard disk drives to the HDI-30 SCSI connector.

The IDE hard disk drive is supported by an IDE hard disk device driver and the ATA Manager. At the system level the IDE hard disk device driver and ATA Manager work in the same way that the SCSI Manager and associated device drivers work. The device driver provides drive partition and data management services for the operating system as well as support for determining device capacity and controlling device specific features. The ATA Manager provides an interface to the IDE drive for the device driver. For additional information about the IDE software, see Chapter 3, “Software for the ATA / IDE Hard Disk.”

SCSI Devices

The PowerBook 150 computer does not supply internal-termination power for devices on the SCSI bus as did for previous all-in-one PowerBook computers. Termination power has to be supplied by the external SCSI device. Internal-termination power in previous all-in-one PowerBook models was supplied by an internal SCSI hard disk drive with single-ended 1.3K ohm pull-up resistors to +5 volts.

The PowerBook 150 computer does not support SCSI disk mode.

Modem Slot

The PowerBook 150 modem slot accepts the any serial modem compatible with the PowerBook 145B. For detailed information about designing serial modems for PowerBook computers refer to *Designing Cards and Drivers for the Macintosh Family*, third edition.

The PowerBook 150 does not support the PowerBook Express Modem.

DRAM Expansion Slot

The RAM expansion slot is compatible with RAM expansion cards meeting Apple design specifications for the PowerBook Duo family. Refer to *Macintosh PowerBook Duo System Developer Note* for the mechanical and electrical design guideline. (A memory adapter kit, part number M3179LL/A, is required for connecting RAM expansion cards to the main logic board.)

Note
Because the PowerBook 150 contains 4 MB of built-in RAM, and the maximum RAM that can be addressed is 40 MB, the largest RAM expansion card that can be fully utilized is 36 MB. If you install a card with more that 36 MB, the additional RAM is unusable. ◆

Identifying the PowerBook 150 Computer

The `gestaltMachineType` value returned by the PowerBook 150 is 115 decimal. As discussed in *Inside Macintosh*, applications should not make decisions based on the machine type alone, but should use the appropriate Gestalt Manager routines to determine what features are available at run time.

Video Display

The PowerBook 150 video display is a 9.5-inch flat panel film super twist nematic (FSTN) dual-scan liquid crystal display (LCD). It provides 640-by-480 2-bit per pixel resolution, is capable of displaying 4 levels of gray, and has on-demand cold cathode fluorescent lamp (CCFL) backlighting.

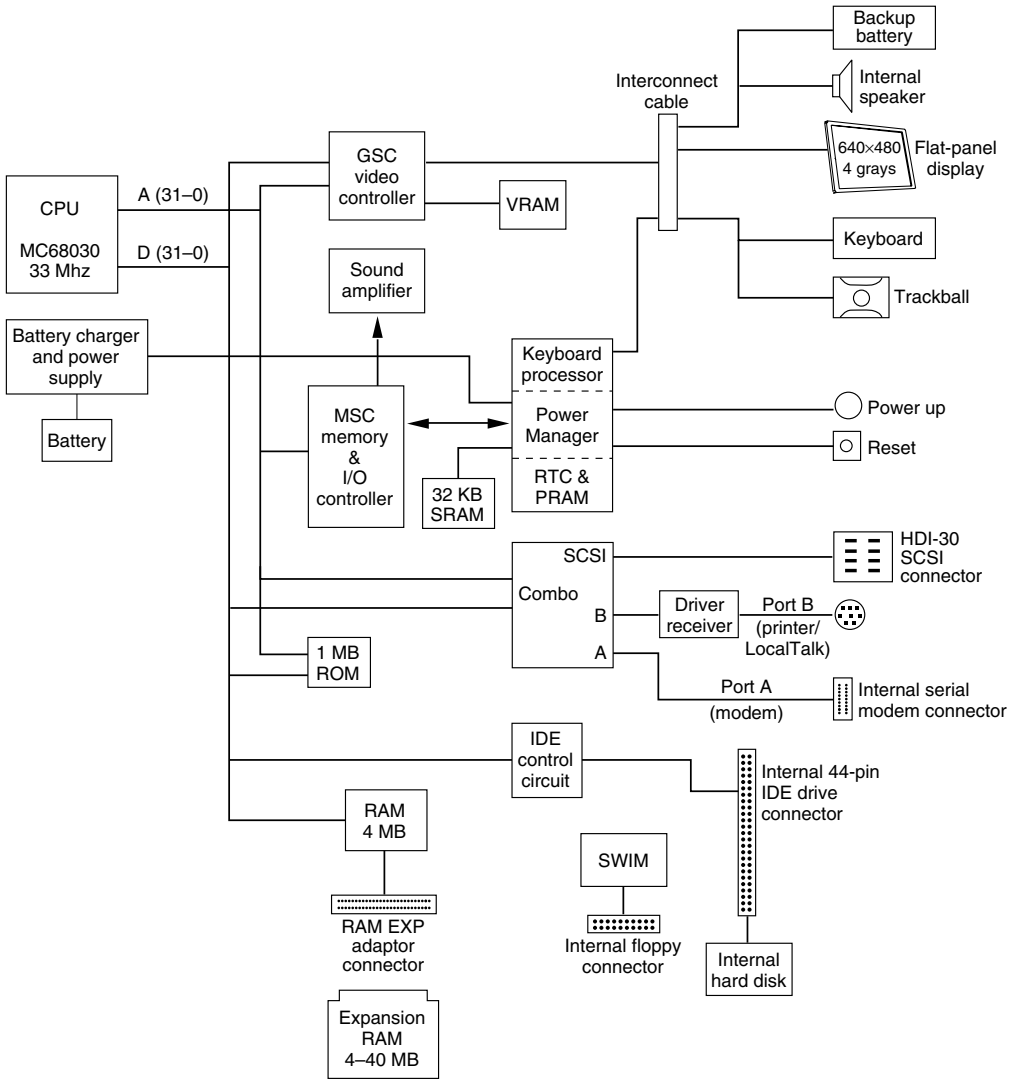
The video implementation on the PowerBook 150 is similar to the video on the PowerBook Duo 230. The video RAM is a 128K x 8-bit device that stores the data required to update and refresh the flat-panel video display. The VRAM for the PowerBook 150 is mapped to locations $6000 0000 through $6080 0000, as it is on the PowerBook Duo family. The display hardware supports 1-bit and 2-bit per-pixel grayscale. The value for the PowerBook 150 panel ID is 4.

PowerBook 150 System Architecture

The PowerBook 150 computer is designed to be the lowest cost all-in-one PowerBook computer solution available from Apple Computer, Inc. To achieve this goal, the PowerBook 150 system architecture incorporates computer technologies from the all-in-one design of the PowerBook 145B and the expandable digital hardware design of the PowerBook Duo 230.

Figure 1-1 shows a block diagram of the digital architecture of the PowerBook 150.

Figure 1-1 PowerBook 150 block diagram



Internal Hardware

The PowerBook 150 internal hardware is made up of a main logic board and supporting subsystems for optional DRAM expansion, an optional modem, system interconnect, and system power inverter.

Main Logic Board

The main logic board of the PowerBook 150 computer is based on the Duo 230 system. The main logic includes the power system, CPU, memory subsystem, video subsystem, data I/O subsystems, and supporting logic.

Both the top and bottom of the main logic board are extensively populated with supporting logic. Figure 1-2 shows the locations of the primary supporting logic and expansion connectors on the top of the PowerBook 150 logic board.

Figure 1-2 Component view (top) of the PowerBook 150 logic board

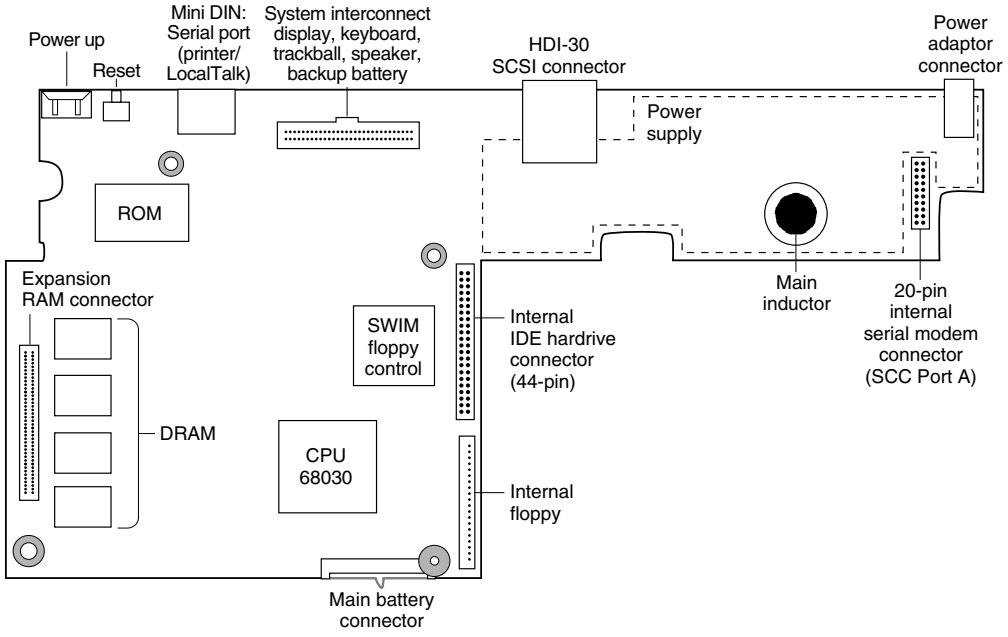
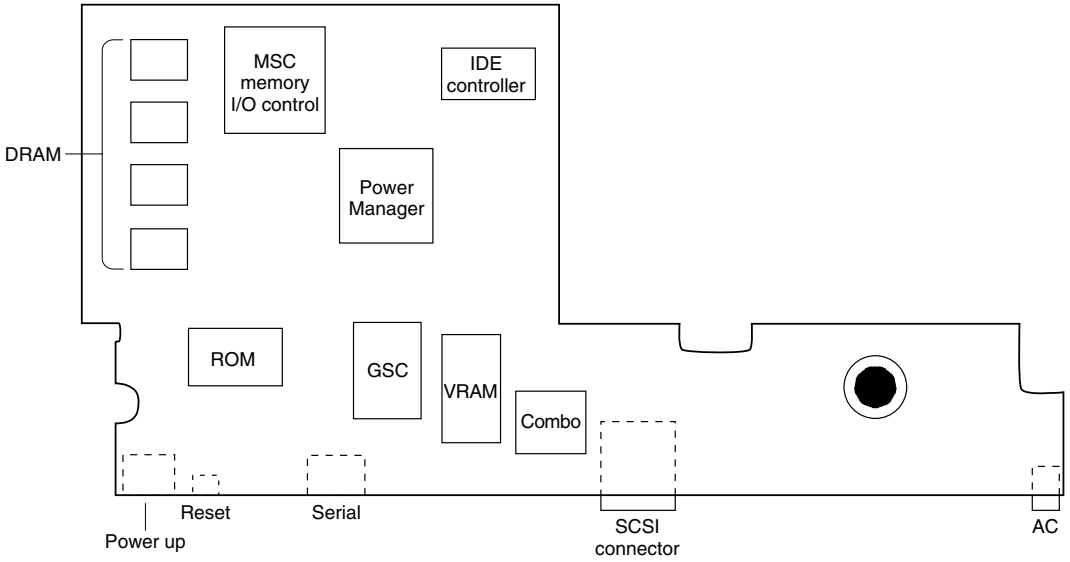


Figure 1-3 shows the locations of the supporting logic components on the bottom of the PowerBook 150 main logic board.

Figure 1-3 Component view (bottom) of the PowerBook 150 logic board



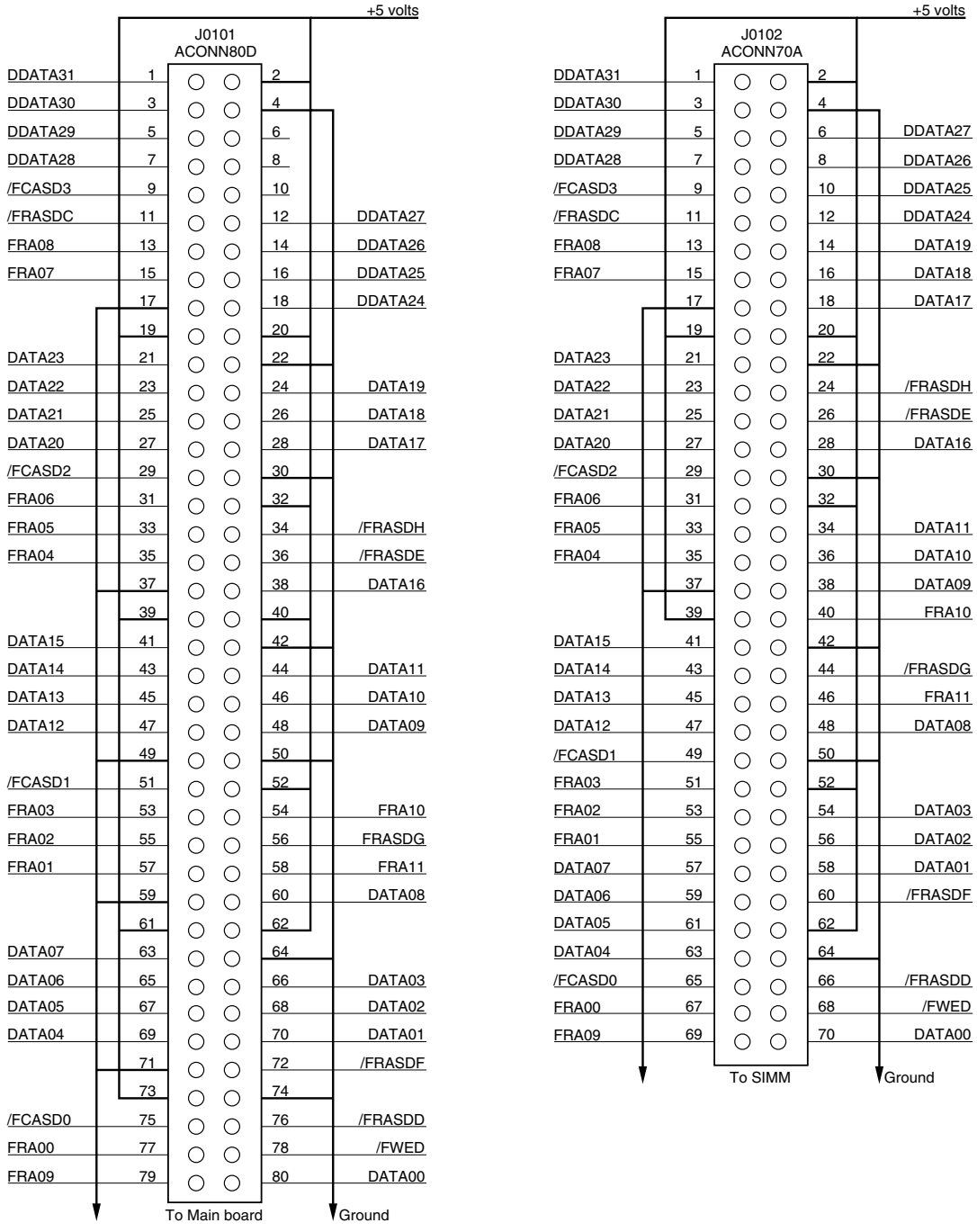
DRAM Expansion

An optional DRAM expansion card, which is the same as the DRAM expansion card for the Duo family of PowerBooks, can be plugged into the top of main logic board through a PowerBook 150 Memory Adapter Kit (see Figure 1-2 for the location of the DRAM-expansion connector on the main logic board). The part number for the Memory Adapter Kit is M3179LL / A. It is available through authorized Apple dealers. You can also obtain the part by ordering directly from the Acer purchasing department (AppleLink address is ACER.PUR.)

The DRAM memory adapter kit consists of a card with two interconnected connectors; an 80-pin AMP connector, part number C-177984-3, which plugs into the main logic board DRAM expansion connector, and a 70-pin JAE connector, part number SX10-70R-LSF-MH2, into which you connect a PowerBook Duo DRAM expansion card.

The pinout for the Memory Adapter Kit connectors is shown in Figure 1-4 and the pin-to-pin signal descriptions are listed in Table 1-1.

Figure 1-4 PowerBook 150 memory adapter kit connector pinout



In the following table, the 80-pin connector (to the main logic board) is referred to as J0101 and the 70-pin connector (to the DRAM SIMM) is referred to as the J0102.

Table 1-1 DRAM memory adapter kit connector signal assignments

J0101 Pin	J0102 Pin	Signal name	Description
1	1	DDATA[31]	Buffered data bit 31
2, 19, 20, 32, 39, 40, 61, 62, 73	2, 19, 20, 32, 39, 52, 62	+5V MAIN	+5 V main power
3	3	DDATA[30]	Buffered data bit 30
4, 17, 22, 30, 37, 42, 49, 50, 59, 64, 71, 74	4,17, 22, 30, 37, 42, 50, 64	GROUND	Ground
5	5	DDATA[29]	Buffered data bit 29
12	6	DDATA[27]	Buffered data bit 27
7	7	DDATA[28]	Buffered data bit 28
14	8	DDATA[26]	Buffered data bit 26
9	9	/FCASD[3]	Filtered column address strobe for DDATA[31-24]
16	10	DDATA[25]	Buffered data bit 25
11	11	/FRASDC	Filtered row address strobe C
18	12	DDATA[24]	Buffered data bit 24
13	13	FRA[8]	Filtered multiplexed address bit 8
24	14	DATA[19]	Data bit 19
15	15	FRA[7]	Filtered multiplexed address bit 7
26	16	DATA[18]	Data bit 18
28	18	DATA[17]	Data bit 17
21	21	DATA[23]	Data bit 23
23	23	DATA[22]	Data bit 22
34	24	/FRASDH	Filtered row addres strobe H
25	25	DATA[21]	Data bit 21
36	26	/FRASDE	Filtered row address strobe E
27	27	DATA[20]	Data bit 20
38	28	DATA[16]	Data bit 16

continued

Table 1-1 DRAM memory adapter kit connector signal assignments (continued)

J0101 Pin	J0102 Pin	Signal name	Description
29	29	/FCASD[2]	Filtered column address strobe for DDATA[23-16]
31	31	FRA[6]	Filtered multiplexed address bit 6
33	33	FRA[5]	Filtered multiplexed address bit 5
44	34	DATA[11]	Data bit 11
35	35	FRA[4]	Filtered multiplexed address bit 4
46	36	DATA[10]	Data bit 10
48	38	DATA[9]	Data bit 9
54	40	FRA[10]	Filtered multiplexed address bit 10
41	41	DATA[15]	Data bit 15
43	43	DATA[14]	Data bit 14
56	44	/FRASDG	Filtered row address strobe G
45	45	DATA[13]	Data bit 13
58	46	FRA[11]	Filtered multiplexed address bit 11
47	47	DATA[12]	Data bit 12
60	48	DATA[8]	Data bit 8
51	49	/FCASD[1]	Filtered column address strobe for DDATA[15-8]
53	51	FRA[3]	Filtered multiplexed address bit 3
55	53	FRA[2]	Filtered multiplexed address bit 2
66	54	DATA[3]	Data bit 3
57	55	FRA[1]	Filtered multiplexed address bit 1
68	56	DATA[2]	Data bit 2
63	57	DATA[7]	Data bit 7
70	58	DATA[1]	Data bit 1
65	59	DATA[6]	Data bit 6
72	60	/FRASDF	Filtered row address strobe F
67	61	DATA[5]	Data bit 5
69	63	DATA[4]	Data bit 4
75	65	/FCASD[0]	Filtered column address strobe DDATA[7-0]
76	66	/FRASDD	Filtered row address strobe D

continued

Table 1-1 DRAM memory adapter kit connector signal assignments (continued)

J0101 Pin	J0102 Pin	Signal name	Description
77	67	FRA[0]	Filtered multiplexed address bit 0
78	68	/FWED	Filtered write enable
79	69	FRA[9]	Filtered multiplexed address bit 9
80	70	DATA[0]	Data bit 0

System Interconnect

A connector on the top of the main logic board connects the main logic with the interconnect board. The interconnect board, which is located in the top case behind the keyboard, routes the trackball, keyboard, backlight inverter, and display cables to the main logic board. A 28 mm speaker and backup battery are also located on the interconnect board.

Backlight Inverter

The backlight inverter, which is also located in the top case behind the keyboard, converts DC power input from the main logic board to AC power and routes it to the display for backlighting. Contrast and brightness are controlled through the backlight inverter board.

Modem

The PowerBook 150 computer has a connector near the top right corner of main logic board for connecting an internal data fax modem. Internal modems compatible with the PowerBook 100, 140, 145B, and 170 will operate in the PowerBook 150 computer. Data and fax-data transfer rates up to 14.4K bits-per-second are supported.

CHAPTER 2

IDE Hard Drive Interface

IDE Hard Drive Interface

This chapter provides information about the IDE hard drive hardware interface to the PowerBook 150 computer. This interface, which is also used for IDE drives on IBM AT-compatible computers, is also referred to as the *ATA IDE interface specification*.

The implementation of the ATA interface on the PowerBook 150 computer is a subset of the ATA interface specification, ANSI Proposal X3T9.2/90-143, Revision 3.1. The controller hardware in the PowerBook 150 supports direct memory mapped programmed I/O transfers only and does not support DMA transfers.

The IDE drive interfaces with the PowerBook 150 main logic board through a 44-pin connector on the end of a ribbon-cable. The connector on the controller electronics of the IDE drive is a standard ATA 40-pin configuration. The PowerBook 150 IDE interface connector has 4 additional lines, which provide ground and +5 volts of power to the drive. The pinout of the PowerBook 150 IDE connector is shown in Figure 2-1.

Figure 2-1 44-pin IDE drive connector pinout

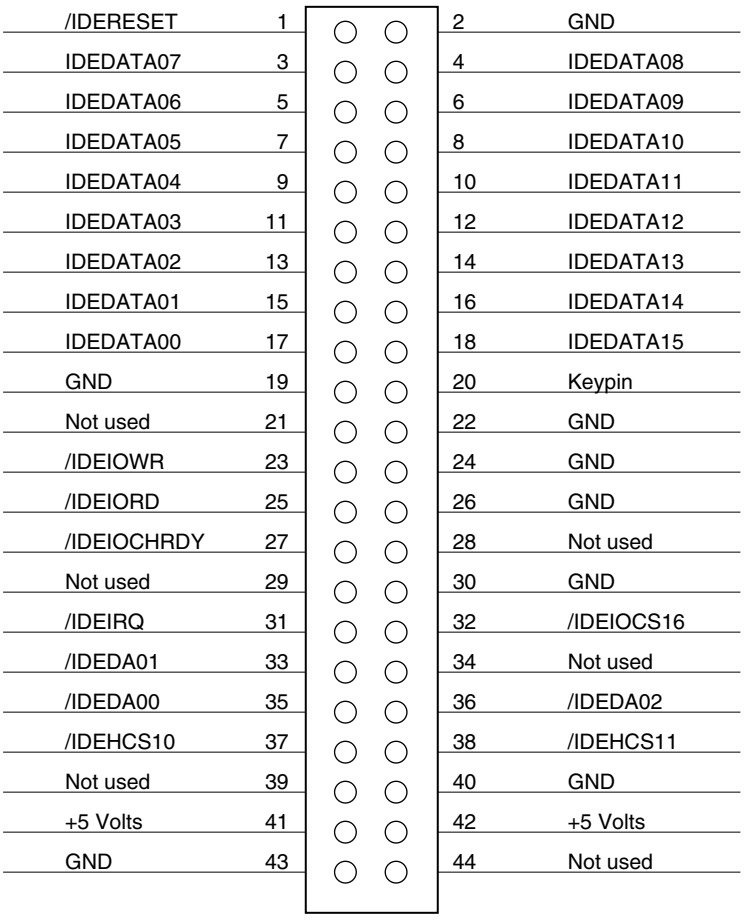


Table 2-1 provides the signal names and descriptions on the PowerBook 150 IDE connector.

Table 2-1 IDE connector signals

Pin number	Signal name	Signal description
1	/IDE_RESET	Hardware reset to the drive. Active low signal.
2	GND	Ground.
3	IDEDATA07	Data bit 7. IDEDATA00-07 signals are used to transfer data between all of the internal registers of the drive and the host.
4	IDEDATA08	Data bit 8. IDEDATA00–15 signals are used to transfer 16-bit data to and from the drive buffer.
5	IDEDATA06	Data bit 6. IDEDATA00-07 signals are used to transfer data between all of the internal registers of the drive and the host.
6	IDEDATA09	Data bit 9. IDEDATA00–15 signals are used to transfer 16-bit data to and from the drive buffer.
7	IDEDATA05	Data bit 5. IDEDATA00-07 signals are used to transfer data between all of the internal registers of the drive and the host.
8	IDEDATA10	Data bit 10. IDEDATA00–15 signals are used to transfer 16-bit data to and from the drive buffer.
9	IDEDATA04	Data bit 4. IDEDATA00-07 signals are used to transfer data between all of the internal registers of the drive and the host.
10	IDEDATA11	Data bit 11. IDEDATA00–15 signals are used to transfer 16-bit data to and from the drive buffer.
11	IDEDATA03	Data bit 3. IDEDATA00-07 signals are used to transfer data between all of the internal registers of the drive and the host.
12	IDEDATA12	Data bit 12. IDEDATA00–15 signals are used to transfer 16-bit data to and from the drive buffer.
13	IDEDATA02	Data bit 2. IDEDATA00-07 signals are used to transfer data between all of the internal registers of the drive and the host.
14	IDEDATA13	Data bit 13. IDEDATA00–15 signals are used to transfer 16-bit data to and from the drive buffer.
15	IDEDATA01	Data bit 1. IDEDATA00-07 signals are used to transfer data between all of the internal registers of the drive and the host.

continued

Table 2-1 IDE connector signals (continued)

Pin number	Signal name	Signal description
16	IDEDATA14	Data bit 14. IDEDATA00–15 signals are used to transfer 16-bit data to and from the drive buffer.
17	IDEDATA00	Data bit 0. IDEDATA00-07 signals are used to transfer data between all of the internal registers of the drive and the host.
18	IDEDATA15	Data bit 15. IDEDATA00–15 signals are used to transfer 16-bit data to and from the drive buffer.
19	GND	Ground.
20	Keypin	This pin is the keypin for the connector.
21	Not used	This is one of the DMA (direct memory access) handshake lines on the IDE drive. DMA is not supported.
22	GND	Ground.
23	/IDE_IOWR	Data write strobe. Active low signal.
24	GND	Ground.
25	/IDE_IORD	Data read strobe. Active low signal.
26	Not used	
27	IDE_IOCHRDY	I/O channel ready. This signal when driven low will insert wait states into the I/O read or write cycles.
28	GND	Ground.
29	Not used	This is one of the DMA (direct memory access) handshake lines on the IDE drive. DMA is not supported.
30	GND	Ground.
31	IDE_IRQ	Host interrupt. This active high signal is used to inform the PowerBook 150 that a data transfer is requested or that a command has terminated
32	/IDE_IOCS16	This signal is asserted low when the data port is accessed. The PowerBook 150 uses this signal to indicate a 16-bit data transfer.
33	IDE_DA01	This signal is used to specify which IDE drive register is being accessed by the PowerBook 150 computer. For more information, see the descriptions of the IDE_HCS10 and IDE_HCS11 signals in this table.
34	Not used	This is the PDIAG signal on the IDE drive. This signal is used in a two-drive configuration. It is not supported.
35	IDE_DA00	This signal is used to specify which IDE drive register is being accessed by the PowerBook 150 computer. For more information, see the descriptions of the IDE_HCS10 and IDE_HCS11 signals in this table.

continued

Table 2-1 IDE connector signals (continued)

Pin number	Signal name	Signal description
36	IDE_DA02	This signal is used to specify which IDE drive register is being accessed by the PowerBook 150 computer. For more information, see the descriptions of the IDE_HCS10 and IDE_HCS11 signals in this table.
37	IDE_HCS10	Register chip select signal. It is asserted high to access the additional control and status registers on the IDE drive.
38	IDE_HCS11	Register chip select signal. It is asserted high to access the main task file registers. The task file registers indicate the command, the sector address, and the sector count.
39	Not used	This is the DASP signal on the IDE drive. This signal is used in a two-drive configuration. It is not supported.
40	GND	Ground.
41	HDISK	+5 volts to ground.
42	HDISK	+5 volts power to the IDE drive from the PowerBook 150.
43	GND	Ground.
44	Not used	Not connected.

CHAPTER 3

Software for the ATA/IDE Hard Disk

This chapter describes the IDE system software for controlling IDE hard disk drives installed in Macintosh computers. To use the information in this chapter, you should already be familiar with writing programs for the Macintosh computer that call device drivers to manipulate devices directly. You should also be familiar with the ATA IDE specification, ANSI proposal X3T9.2/90-143, Revision 3.1.

Introduction to IDE Software

Support for IDE (integrated drive electronics) hard disk drives is incorporated in the JeDI ROM. Macintosh system software for accessing IDE hard drives is included in a new IDE hard disk drive device driver and the ATA Manager.

At the system level, the IDE device driver and ATA Manager work in the same way that the SCSI Manager and associated SCSI device drivers work. The IDE hard disk device driver provides drive partition, data management, and error-handling services for the operating system as well as support for determining device capacity and controlling device-specific features. The ATA Manager provides an interface to the IDE drive for the IDE device driver.

IDE hard disk drives appear on the desktop just like SCSI hard disk drives currently do. Other than those that perform low-level services, such as formatting and partitioning utilities, applications interact with the IDE hard disk drives in a device-independent manner through the File Manager or Printing Manager. The relationship of the ATA Manager to the Macintosh system architecture is shown in Figure 3-1.

The IDE software supports both synchronous and asynchronous data transfers. A completion routine must be provided by the caller for asynchronous operations.

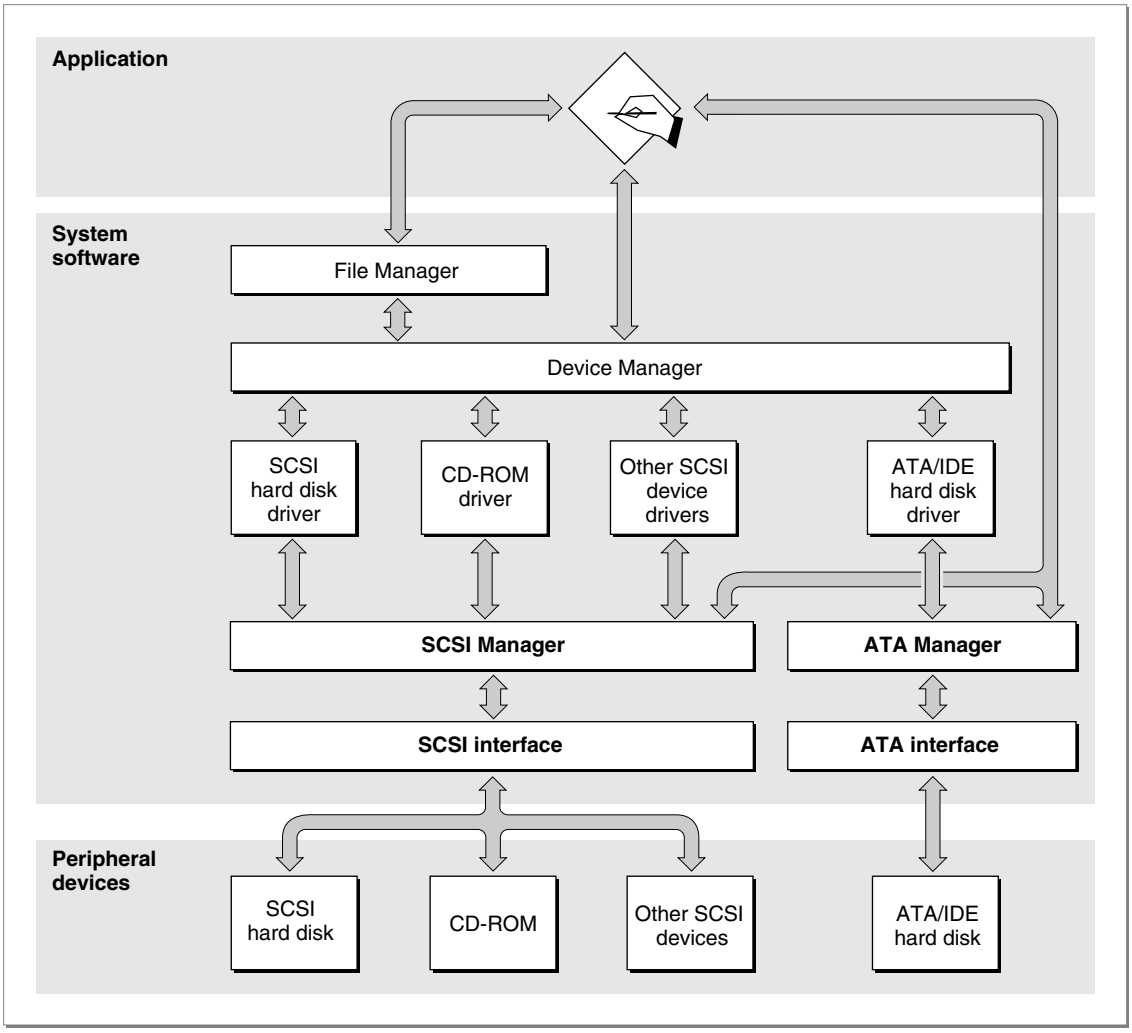
IDE Hard Disk Device Driver

The Macintosh IDE hard disk device driver provides operating system-dependent services through a set of driver function calls required to interface with the Macintosh operating system. In addition, it provides additional control and status calls that are specific to the IDE hard disk device driver implementation. The required driver calls, as specified in *Inside Macintosh*, are `open`, `close`, `prime`, `control`, and `status`.

In addition to the required function calls, the IDE hard disk device driver provides power management status information for the JeDI and additional device-specific features. IDE hard disk device driver control and status calls are defined in “IDE Hard Disk Device Driver Reference” beginning on page 3-4.

Note
The JeDI contains only one internal IDE hard drive, and it is a fixed hard disk drive. External IDE devices are not supported on the JeDI. ◆

Figure 3-1 Relationship of the ATA Manager to the Macintosh system architecture



At system startup time, the IDE device driver gets installed as one of the device drivers from the JeDI ROM. Note that this is different from the driver loading sequence for SCSI hard drive devices, which are RAM-based drivers that are loaded from the device media. Because of this difference, the IDE hard disk device driver cannot be patched out during system startup.

The IDE hard disk device driver has a driver reference number of -54 and a driver name of .ATDrvr. Like all Macintosh device drivers, calls to the IDE hard disk device driver can be made using either the reference number or the driver name.

The IDE hard disk device driver does not provide request queuing. All driver requests are either completed immediately or are passed to the ATA Manager for further processing. For further information about the control calls for the IDE hard disk device driver, see “IDE Hard Disk Device Driver Reference” beginning on page 3-4.

ATA Manager

The Macintosh ATA Manager schedules I/O requests from the IDE hard disk device driver, the operating system, and applications. It is also responsible for managing the hardware interface to the IDE controller electronics.

When making calls to the ATA Manager, you have to pass and retrieve parameter information through a parameter block. The size and contents of the parameter block depends on the call being made. However, every call to the ATA Manager has a common parameter block header structure. The structure of the `ataPBHdr` parameter block is common to all ATA parameter block data types. Several additional ATA parameter block data types have been defined for the various function calls to the ATA Manager. The additional parameter block data types, which are specific to the function call being made, are described in "ATA Manager Reference" beginning on page 3-14.

IDE Hard Disk Device Driver Reference

This section describes the Macintosh device driver services provided by the IDE hard disk device driver. The information in this section assumes that you are already familiar with using device driver services on the Macintosh computer. If you are not familiar with Macintosh device drivers, refer to Chapter 6, "Device Manager" in *Inside Macintosh, Volume II* for additional information.

High-Level Device Manager Routines

The IDE hard disk device driver supports the required set of high-level Device Manager routines, as defined in Chapter 6 of *Inside Macintosh, Volume II*. They are briefly defined here for convenience. Additional control functions supported in the IDE hard disk device driver are defined in "IDE Hard Disk Device Driver Control Calls" beginning on page 3-7.

Open

The `open` routine opens the IDE hard disk device driver during the boot sequence after the driver code is retrieved from the ROM. It returns a reference number to the driver, which is used in subsequent calls to the driver.

The following operations take place at boot time:

- memory allocation and driver globals and internal variables initialization
- power-on drive diagnostics
- device detection and verification
- device initialization

- device information uploading
- drive queue management and event posting

After booting, the driver responds with `noErr` to subsequent calls to the `open` routine, and does not repeat the operations performed at boot time.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`openErr`	-23	Could not open the driver
`DRVRCantAllocate`	-1793	Global memory allocation error
`ATABufFail`	-1796	Device buffer test failed

Close

The `close` routine deallocates the A5 memory storage, removes the drive queue entry point, removes the VBL task, and closes the IDE hard disk device driver.

RESULT CODES

`noErr`	`0`	Successful completion, no error occurred

Prime

The `prime` routine performs either a read or write command as specified by the caller. During this process, the following operations take place:

- byte-to-block translation
- address translation
- I/O parameter block updating
- high-level error recovery
- ATA Manager parameter block management (refer to “The ATA Parameter Block” on page 3-14 for more information about the parameter block structure for the ATA Manager)

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`ioErr`	-36	I/O error
`paramErr`	-50	Invalid parameter specified
`nsDrvErr`	-56	Specified drive does not exist

Status

The `status` routine returns status information about the IDE hard disk device driver. The type of information returned is specified in the `csCode` field and the information itself is pointed to by the `csParamPtr` field.

The IDE hard disk device driver implements the same status calls as supported by the SCSI hard disk device driver:

csCode	Definition
$08	Return drive status information
$43	Return driver Gestalt information
$70	Return power mode status information

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`statusErr`	-18	Unimplemented status call; could not complete requested operation
`nsDrvErr`	-56	Specified drive does not exist

Control

The `control` routine sends control information to the IDE hard disk device driver. The type of control call is specified in `csCode`.

The IDE hard disk device driver implements the same control calls supported by the SCSI hard disk device driver. The control calls are described in “IDE Hard Disk Device Driver Control Calls” beginning on page 3-7 and the list of control call codes is as follows:

csCode	Definition
$01	Kill I/O
$05	Verify media
$06	Format media
$07	Eject media
$21	Return drive icon
$22	Return media icon
$23	Return drive characteristics
$65	Need time code
$70	Power-mode status

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`controlErr`	-17	Unimplemented control call; could not complete requested operation
`nsDrvErr`	-56	Specified drive does not exist

IDE Hard Disk Device Driver Control Calls

The IDE hard disk device driver supports a standard set of control and status calls for accessing IDE hard disk drive devices.

Standard Control Calls

This section describes the standard control calls defined within the IDE hard disk device driver.

The Kill I/O Function

The function to kill I/O returns a `noErr` result if the logical drive number is valid; however, it is not supported and on an IDE hard disk drive. A `controlErr` status is returned.

INPUT PARAMETERS

`csCode`	A value of $01.
`ioVRefNum`	The logical drive number.
`csParam[]`	None defined.

OUTPUT PARAMETERS

`ioResult`	See the list of result codes.

RESULT CODES

`controlErr`	-17	Unimplemented control call; could not complete requested operation

The Verify Function

The verify function requests a read verification of the data on the IDE hard disk drive media.

INPUT PARAMETERS

`csCode`	A value of $05.
`ioVRefNum`	The logical drive number.
`csParam[]`	None defined.

OUTPUT PARMETERS

`ioResult`	See the list of result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	-56	Specified drive does not exist

The Format Function

The format function initializes the hard drive for use by the operating system. Since IDE hard drives are low-level formatted at the factory, this call does not perform any operation. The driver always returns `noErr` if the logical drive number is valid.

INPUT PARAMETERS

`csCode`	A value of $06.
`ioVRefNum`	The logical drive number.
`csParam[]`	None defined.

OUTPUT PARAMETERS

`ioResult`	See the list of result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	-56	Specified drive does not exist

The Eject Media Function

The eject media function prepares and initiates an eject operation from the specified drive. This call returns `noErr` if the logical drive number is valid, however it performs no other operation on the JeDI.

INPUT PARAMETERS

`csCode`	A value of $07.
`ioVRefNum`	The logical drive number.
`csParam[]`	None defined.

OUTPUT PARAMETERS

`ioResult`	See the list of result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	-56	Specified drive does not exist

The Return Drive Icon Function

The return drive icon function returns a pointer to the device icon and the device name string. The drive icon is the same as the media icon for IDE hard disk drives.

INPUT PARAMETERS

`csCode`	A value of $21.
`ioVRefNum`	The logical drive number.
`csParam[]`	None defined.

OUTPUT PARAMETERS

`csParam[0-1]`	The address of the drive icon and name string. The information is in ICN# format.
`ioResult`	See the list of result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	-56	Specified drive does not exist

The Return Media Icon Function

The return media icon function returns a pointer to the media icon and the name string. The media icon is the same as the drive icon for IDE hard disk drives.

INPUT PARAMETERS

`csCode`	A value of $22.
`ioVRefNum`	The logical drive number.
`csParam[]`	None defined.

OUTPUT PARAMETERS

`csParam[0-1]`	The address of the media icon and name string. The information is in ICN# format.
`ioResult`	See the list of result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	-56	Specified drive does not exist

The Return Drive Characteristics Function

The return drive characteristics function returns information about the characteristics of the specified drive as defined in *Inside Macintosh, Volume V.*

INPUT PARAMETERS

`csCode`	A value of $23.
`ioVRefNum`	The logical drive number.
`csParam[]`	None defined.

OUTPUT PARAMETERS

`csParam[0-1]`	Drive information: $0601 = primary, fixed, SCSI, internal; $0201 = primary, SCSI, internal
`ioResult`	See the list of result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	-56	Specified drive does not exist

The Need Time Code Function

The need time code function allows the driver time to perform periodic operations, such as checking for media insertion or ejection events related to removable cartridge drives. For additional information about how this call is used, see the description of the driver `dNeedTime` flag in Chapter 6 of *Inside Macintosh, Volume II*. This function performs no operation on the IDE hard disk drive in a JeDI computer.

INPUT PARAMETERS

`csCode`	A value of $65.
`csParam[]`	None defined.

OUTPUT PARAMETERS

`ioResult`	See the list of result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	-56	Specified drive does not exist

IDE Hard Disk Device Driver Status Call

This section describes the functions for retrieving status information from the IDE hard disk device driver.

The Drive Status Info Function

The IDE hard disk device driver provides a drive status call for retrieving status information from the drive. This function returns the same type of information that hard disk device drivers are required to return for the `DriveStatus` function, as described in Chapter 6 of *Inside Macintosh, Volume II*.

INPUT PARAMETERS

`csCode`	A value of $08.
`ioVRefNum`	The logical drive number.
`csParam[]`	None defined.

OUTPUT PARAMETERS

`csParam[]`	The `csParam` field contains status information about the internal IDE disk drive.
`ioResult`	See the list of result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	-56	Specified drive does not exist

The Return Driver Gestalt Function

This function provides applications information about the IDE hard disk device driver and the attached device. Several calls are supported under this function. A Gestalt selector is used to specify a particular call.

The `DriverGestaltParam` data type defines the IDE Gestalt parameter block:

```
typedef struct DriverGestaltParam
{
   ataPBHdr
   short              ioVRefNum;              /* device refNum */
   short              csCode;                 /* Gestalt code */
   OSType             driverGestaltSelector;  /* selector */
   driverGestaltInfo  driverGestaltResponse;  /* result */
} DriverGestaltParam;
```

The fields `driverGestaltSelector` and `driverGestaltResponse` are 32-bit fields.

INPUT PARAMETERS

`csCode`	A value of $43.
`ioVRefNum`	The logical drive number.
`driverGestaltSelector`	Gestalt function selector. This is a 32-bit ASCII field containing one of the following selectors:

`sync`	indicates synchronous or asynchronous driver;
`devt`	indicates type of device the driver is controlling;
`intf`	indicates the device interface;
`boot`	indicates PRAM value to designate this driver or device;
`vers`	indicates the version number of the driver.

OUTPUT PARAMETERS

`driverGestaltResponse`	The result returned; based on the driver gestalt selector. The possible return values are:
	`true` If the `sync` driver selector is specified, this Boolean value indicates that the driver is synchronous. A `false` value indicates asynchronous;
	`disk` If the `devt` driver selector is specified, this value indicates a hard disk device driver;
	`ide` If the `intf` driver selector is specified, this value indicates the interface is IDE;
	`0` If the `boot` driver selector is specified, this value indicates that this is the boot driver or boot device;
	nnnn If the `vers` selector is specified, the current version number of the driver is returned.
`ioResult`	See the list of result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`statusErr`	-18	Unknown selector was specified
`nsDrvErr`	-56	Specified drive does not exist

The Power-Mode Status Function

This function returns the current power mode state of the internal hard disk.

INPUT PARAMETERS

`csCode`	A value of $70.
`ioVRefNum`	The logical drive number.
`csParam[]`	None defined.

OUTPUT PARAMETERS

`csParam[]`	The most significant bit of this field contains one of the following values: 1 = Drive is in standby mode; 2 = Drive is in idle mode; 3 = Drive is in sleep mode.
`ioResult`	See the list of result codes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`controlErr`	-17	Unimplemented control call; could not complete requested operation
`nsDrvErr`	-56	Specified drive does not exist

ATA Manager Reference

This section defines the data structures and functions that are specific to the ATA Manager. The "The ATA Parameter Block" section shows the data structure of the ATA parameter block. The "Functions" section describes the functions for managing and performing data transfers through the ATA Manager.

The ATA Parameter Block

This section defines the fields that are common to all ATA Manager functions that use the ATA parameter block. The fields that are used for specific functions are defined in the description of the functions to which they apply. You use the ATA parameter block for all calls to the ATA Manager.

The parameter block includes a field, `MgrFCode`, in which you specify the function selector for the particular routine to be executed; you must specify a value for this field. Each ATA function may use different fields of the ATA parameter block for parameters specific to that function.

An arrow preceding the comment indicates whether the parameter is an input parameter, an output parameter, or both:

Arrow	Meaning
→	Input
←	Output
↔	Both

The `ataPBHdr` data type defines the ATA parameter block.

```
typedef    unsigned char   uchar;
typedef    unsigned short  ushort;
typedef    unsigned long   ulong;

typedef     struct   ataPBHdr
{
            Ptr      ataLink;       /* reserved */
            short    ataQType;      /* type byte */
            uchar    ataPBVers;     /* → parameter block
                                       version number */
```

```
            uchar    hdrReserved;   /* reserved */
            Ptr      hdrReserved2;  /* reserved */
            ProcPtr  ataCompletion; /* completion routine Ptr */
            short    ataResult;     /* ← returned result */
            uchar    MgrFCode;      /* → manager function code */
            uchar    ataIOSpeed;    /* → I/O timing class */
            ushort   ataFlags;      /* → control options */
            short    hdrRevered3;   /* reserved */
            long     deviceID;      /* → device ID*/
            ulong    TimeOut;       /* → transaction timeout */
            Ptr      ataPtr1;       /* client storage Ptr 1 */
            Ptr      ataPtr2;       /* client storage Ptr 2 */
            ushort   ataState;      /* reserved */
            short    hdrReserved4;  /* reserved */
            long     hdrReserved5;  /* reserved */
} ataPBHdr;
```

Field descriptions

`ataLink` This field is reserved for use by the ATA Manager. It is used internally for queuing I/O requests. It must be initialized to 0 before calling the ATA Manager.

`ataQType` This field is the queue type byte. It should be initialized to 0 before calling the ATA Manager.

`ataPBVers` This field contains the parameter block version number. A value of 1 is the only value currently supported. Any other value results in a `paramErr`.

`hdrReserved` Reserved field for future use. In order to ensure future compatibility, all reserved fields should be set to 0.

`hdrReserved2` Reserved field for future use. In order to ensure future compatibility, all reserved fields should be set to 0.

`ataCompletion` This field contains the completion routine pointer that is to be called upon completion of a request. When this field is set to 0, it indicates a synchronous I/O request; a nonzero value indicates an asynchronous I/O request. The routine pointed by this field is called when the request has finished without error, or when the request has terminated due to an error. This field is valid for any manager request. The completion routine is called as follows:

```
pascal void (*RoutinePtr) (ataIOPB *)
```

The completion routine is called with the associated manager parameter block in the stack.

`ataResult` Completion status. This field is returned to the caller by the ATA Manager after the request has been completed. Refer to Table 3-2 for a list of the possible error codes returned in this field.

`MgrFCode` This field is the manager function code for the ATA Manager. The functions that the ATA Manager uses for these services are described

in “Functions” beginning on page 3-18. An invalid code in this field results in an `ATAFuncNotSupported` error.

`ataIOSpeed` This field specifies the I/O cycle timing requirement of the specified IDE hard drive. This field should contain word 51 of the identify drive data. Currently values 0 through 32 are supported, as defined in the ATA specification. If a timing value higher than 1 supported is specified, then the manager shall operate in the fastest timing mode supported by the manager. Until the timing value is determined by examining the identify drive data returned by the `ATA_Identify` routine, you should request operations using the slowest mode (mode 0).

`ataFlags` This 16-bit field contains control settings that indicate special handling of the requested function. The control bits are defined as follows:

Bit	Name	Definition
0	reserved	
3	`RegUpdate`	When set to 1 this bit indicates that a set of device registers should be reported back upon completion of the request. This bit is valid for the `Execute I/O` function only. Refer to the description of `Execute I/O` for detail. The following device registers are reported back: Sector count register Sector number register Cylinder register SDH register
8, 9	`SGType`	This 2-bit field specifies the type of scatter gather list passed in. This field is only valid for read / write operations. The following types are defined: 00 = Scatter gather disabled 01 = Scatter gather type I enabled 10 = Reserved 11 = Reserved When set to 0, this field indicates that the `ioBuffer` field contains the host buffer address for this transfer, and the `ioReqCount` field contains the byte transfer count. When set to 1, this field indicates that the `ioBuffer` and the `ioReqCount` fields of the parameter block for this request point to a host scatter gather list and the number of scatter gather entries in the list, respectively.

The format of the scatter gather list is a series of the following structure definition:

```
typedef struct
{
  uchar* ioBuffer; /* pointer */
  ulong ioReqCount;/* byte
                        count */
} IOBlock;
```

10 `QLockOnError`
When set to 0, this bit indicates that an error during the transaction should not freeze the I/O queue for the device. When an error occurs on an I/O request with this bit set to 0, the next queued request will be processed following this request. When an error occurs on an I/O with this bit set to 1, the user must issue a I/O queue release command to continue. A status code of `$717` is returned for subsequent asynchronous I/O requests until the I/O queue release command is issued.

11 `Immediate` When this bit is set to 1, it indicates that the request shall be executed as soon as possible and the status of the request shall be returned. It forces the request to the head of the I/O queue for immediate execution. When this bit is set to 0, the request is queued in the order it is received and shall execute the request according to that order.

12, 13 `ATAioDirection`
This bit field specifies the direction of data transfer. Bit values are binary and defined as follows:

00 = No data transfer
10 = Data direction in (read)
01 = Data direction out (write)
11 = Reserved

15 `ByteSwap` When set to 1, this bit indicates that every byte of data prior to transmission on write and upon reception on read should be swapped. When this bit is set to 0, it forces bytes to go out in LSB-MSB format which is compatible with IBM clones. Typically, this bit should be set to 0. Setting this bit does have some performance implications, since the byte swap is performed by the software. Use this bit with caution.

`deviceID` A short word that uniquely identifies an IDE device. The field consists of the following structure:

```
typedef struct
{
  ushort Reserved;   /* reserved */
  ushort deviceNum;  /* device ID and bus ID */
} deviceIdentification;
```

Bit 15 of `deviceNum` field indicates master (=0) / slave (=1) selection. Bits 14 through 0 contain the bus ID (for example, 0x0 = master unit of bus 0, 0x80 = slave unit of bus 0). The JeDI implementation only allows one device in the master configuration. This value is always 0.

`TimeOut` This field specifies the transaction timeout value in milliseconds. A value of zero disables the transaction timeout detection.

`ataPtr1` This pointer field is available for application use. It is not modified by the ATA Manager.

`ataPtr2` This pointer field is available for application use. It is not modified by the ATA Manager.

`ataState` This field is used by the ATA Manager to keep track of the current bus state. This field must contain 0 when calling the manager. The following states are defined:

Code	Name	Description
$00	Initial	Parameter block processing started
$01	Started	Command delivery state
$0F	Data	Data delivery state
$1F	Status	Status delivery state
$3F	Complete	Bus transaction complete state
$FF	Idle	Waiting to return state

Functions

This section describes the ATA Manager services that are used to manage and perform data transfers. Each service is requested through a parameter block specific to it. A request for an IDE service is specified by a manager function code within the parameter block. The entry point for all of the functions is the same. The functions provided by the ATA Manager are listed in Table 3-1.

Table 3-1 ATA Manager functions

Function name	Function code	Description
ATA_NOP	$00	No operation
ATA_ExecIO	$01	ATA I/O execution
ATA_BusInquiry	$03	Bus inquiry
ATA_QRelease	$04	I/O queue release
ATA_Abort	$10	Command termination
ATA_ResetBus	$11	IDE bus reset
ATA_RegAccess	$12	ATA device register access
ATA_Identify	$13	Get the drive identify data
ATA_DrvrRegister	$85	Driver refNum registration
ATA_FindRefNum	$86	Driver refNum lookup
ATA_DrvrDeregister	$87	Driver refNum deregistration
ATA_MgrInquiry	$90	ATA Manager inquiry
ATA_MgrInit	$91	ATA Manager initialization
ATA_MgrShutDown	$92	ATA Manager shutdown

ATA I/O Execution

You can use the ATA_ExecIO function to execute all data I/O transfers to or from an IDE device. Your application must provide all of the parameters needed to complete the transaction prior to calling the ATA Manager. Upon return, the parameter block contains the result of the request.

A prior call to the ATA_MgrInit function to initialize the ATA Manager must be performed before issuing the ATA_ExecIO function. See “ATA Manager Initialization” on page 3-23 for information about calling the ATA_MgrInit function.

The manager function code for the ATA_ExecIO function is 1.

The parameter block associated with the ATA_ExecIO function is defined below:

```
typedef      struct
{
   ataPBHdr
   char      ataStatusReg;   /* ← last device status register
                                 image */
   char      ataErrorReg;    /* ← last device error register
                                 image (valid if bit 0 of Status
                                 register is set) */
   short     ataReserved;    /* reserved */
   ulong     BlindTxSize;    /* → data transfer size */
   IOBlock   IOBlk;          /* → data transfer
                                 descriptor block */
```

```
   ulong     ataActualTxCnt; /* ← actual number of bytes
                                transferred */
   ulong     ataReserved2;   /* reserved */
   devicePB  RegBlock;       /* → device register images */
   uchar*    packetCDBPtr;   /* ATAPI packet command block
                                pointer */
   ushort    ataReserved3[6];/* reserved */

} ATA_ExecIO;
```

Field descriptions

`ataPBHdr` See the definition of the `ataPBHdr` structure on page 3-14.

`ataStatusReg` This field contains the last device status register image. See the ATA specification for status register bit definitions.

`ataErrorReg` This field contains the last device error register image. This field is valid only if the Error bit (bit 0) of the `Status` register is set. See the ATA specification for error register bit definitions.

`BlindTxSize` This field specifies the maximum number of bytes that can be transferred per interrupt or detection of DRQ. Bytes are transferred in blind mode (no byte-level handshake). Once an interrupt or a DRQ condition is detected, the manager transfers up to the number of bytes specified in the field from or to the selected device. The typical number of bytes transferred is 512 bytes.

`IOBlk` This field contains the I/O block, which provides either the host buffer address and the requested transfer length, or contains a pointer to a scatter gather list and the number of scatter gather entries. If the `SGType` bits of the `ataFlags` field are set, this field contains the scatter gather information. The I/O block is defined by the `IOBlk` data structure as follows:

```
typedef      struct
{
   uchar*    ioBuffer;   /* ↔ data buffer Ptr */
   ulong     ioReqCount; /* ↔ transfer length */
} IOBlk;
```

`ioBuffer` This field contains the host buffer address for the number of bytes specified in the `ioReqCount` field. Upon return, this field is updated to reflect data transfers. When the `SGType` bits of the `ataFlags` field are set, this field points to a scatter gather list. The scatter gather list consists of series of `IOBlk` entries.

`ioReqCount` This field contains the number of bytes to transfer either from or to the buffer specified in `ioBuffer`. Upon return, this field is updated to reflect data transfers (0 if successful; otherwise the number of

bytes still to be transferred). When the `SGType` bits of the `ataFlags` field are set, this field contains the number of scatter gather entries in the list pointed by the `ioBuffer` field.

`ataActualTxCnt` This field contains the total number of bytes transferred for this request. Currently not supported.

`RegBlock` This field contains the IDE device register image structure. The values contained in this structure are written out to the device during the command delivery state. The caller must provide the image prior to calling ATA Manager. The device register images are defined by the `RegBlock` structure as follows:

```
typedef  struct
{
   uchar  Features;  /* → features register */
   uchar  Count;     /* ↔ sector count */
   uchar  Sector;    /* ↔ sector start/finish */
   uchar  Reserved;  /* reserved */
   ushort Cylinder;  /* ↔ cylinder 68000 format */
   uchar  SDH;       /* ↔ SDH register image
   uchar  Command;   /* → command register image */
} RegBlock;
```

`packetCDCPtr` This field contains the packet pointer for ATAPI. The current version of ATA Manager doesn't support the ATAPI protocol. For ATA commands, this field should contain 0 in order to ensure compatibility in the future.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	-56	Specified logical drive number does not exist
`AT_AbortErr`	-1780	Command Aborted bit set in error register
`AT_RecalErr`	-1781	Track 0 Not Found bit set in error register
`AT_WrFltErr`	-1782	Write Fault bit set in status register
`AT_SeekErr`	-1783	Seek Complete bit not set upon completion
`AT_UncDataErr`	-1784	Uncorrected Data bit set in error register
`AT_CorDataErr`	-1785	Data Corrected bit set in status register
`AT_BadBlkErr`	-1786	Bad Block bit set in error register
`AT_DMarkErr`	-1787	Data Mark Not Found bit set in error register
`AT_IDNFErr`	-1788	ID Not Found bit set in error register
`ATABusyErr`	-1790	Selected device busy (BUSY bit set)
`ATAMgrNotInitialized`	-1802	ATA Manager not initialized
`ATAPBInvalid`	-1803	Invalid device base address detected
`ATATransTimeOut`	-1806	Timeout: transaction timeout detected
`ATAReqInProg`	-1807	I/O channel in use - can not proceed
`ATAUnknownState`	-1808	Device in unknown state
`ATAQLocked`	-1809	I/O queue locked - can not proceed

ATA Manager Inquiry

You can use the `ATA_MgrInquiry` function to get information about the ATA Manager such as the version number. This function may be called prior to the manager initialization, however, the system configuration information may be invalid.

The manager function code for the `ATA_MgrInquiry` function is $90.

The parameter block associated with this function is defined as:

```
typedef         struct
{
   ataPBHdr
   NumVersion   MgrVersion
   ushort       MGRPBVers;        /* ← manager PB version number
                                     supported */
   ushort       Reserved;         /* reserved */
   ushort       ataBusCnt;        /* ← number of ATA buses in
                                     system */
   ushort       ataDevCnt;        /* ← number of ATA devices
                                     detected */
   unchar       ataMaxMode;       /* ← maximum I/O speed mode */
   ushort       Reserved;         /* reserved */
   ushort       IOClkResolution;  /* ← IO clock resolution */
   ushort       Reserved[17];     /* reserved */
} ATA_MgrInquiry;
```

Field descriptions

`ataPBHdr` See the `ataPBHdr` parameter block definition on page 3-14.

`MgrVersion` Upon return, this field contains the version number of the ATA Manager.

`MGRPBVers` This field contains the latest version number of the `ataPBHder` parameter block supported by the ATA Manager. Applictions can use any parameter block definition up to the version returned.

`ataBusCnt` Upon return, this field contains the total number of ATA buses in the system. This field contains a zero if the ATA Manager has not been initialized.

`ataDevCnt` Upon return, this field contains the total number of ATA devices detected on all ATA buses. The current architecture allows only one device per bus. This field will contain a zero if the ATA Manager has not been initialized.

`ataMaxMode` This field specifies the maximum I/O speed mode that the ATA Manager supports. Refer to ATA specification for information on mode timing.

`IOClkResolution` This field contains the I/O clock resolution in nanoseconds. The current implementation doesn't support the field (returns 0).

RESULT CODES

noErr	0	Successful completion, no error occurred

ATA Manager Initialization

You must call the ATA_MgrInit function prior to issuing ATA_ExecIO, ATA_BusInquiry, ATA_QRelease, or ATA_Abort command. Otherwise, an ATAMgrNotInitialized error is returned. This call initializes internal variables and IDE hard disk drive hardware. Consecutive initialization calls from either the same or another application are ignored and the noErr result is returned.

The manager function code for the ATA_MgrInit function is $91.

There are no additional function-specific variations on ataPBHdr for this call.

RESULT CODES

noErr	0	Successful completion, no error occurred
ATAInitFail	-1795	ATA Manager initialization failure

ATA Bus Inquiry

You can use the ATA_BusInquiry function to get information on a specific ATA bus. This call is provided for possible future expansion of the Macintosh ATA architecture.

The manager function code for the ATA_BusInquiry function is $03.

The parameter block associated with this function is defined as:

```
typedef struct
{
   ataPBHdr
   ushort     ataEngineCount;     /* ← returns zero */
   ushort     ataReserved;        /* reserved */
   ulong      ataDataTypes;       /* ← returns zero */
   ushort     ataIOpbSize;        /* ← size of ATA IO PB */
   ushort     ataMaxIOpbSize;     /* ← returns zero */
   ulong      ataFeatureFlags;    /* ← reserved */
   uchar      ataVersionNum;      /* ← HBA version number */
   uchar      ataHBAInquiry;      /* ← returns zero */
   ushort     ataReserved2;       /* reserved */
   ulong      ataHBAPrivPtr;      /* ← Ptr to HBA private data */
   ulong      ataHBAPrivSize;     /* ← size of HBA private data */
   ulong      ataAsyncFlags;      /* ← capability for callback */
   ulong      ataReserved3[4];    /* reserved */
   ulong      ataReserved4;       /* reserved */
   char       ataReserved5[16];   /* reserved */
   char       ataHBAVendor[16];   /* ← HBA vendor ID */
```

```
   char       ataContrlFamily[16];/* ← family of ATA Controller */
   char       ataContrlType[16];  /* ← model num of controller */
   char       ataXPTversion[4];   /* ← version number of XPT */
   char       ataResrved6[4];     /* reserved */
   char       ataHBAversion[4];   /* ← version number of HBA */
   uchar      ataHBAslotType;     /* ← type of slot */
   uchar      ataHBAslotNum;      /* ← slot number of the HBA */
   ushort     ataReserved7;       /* reserved */
   ulong      ataReserved8;       /* reserved */
} ATA_BusInquiry;
```

Field descriptions

`ataPBHdr` See the `ataPBHdr` definition on page 3-14.

`ataEngineCount` Currently 0.

`ataReserved` Reserved. All reserved fields are set to 0.

`ataDataTypes` Not supported by current ATA architecture. Returns a bit map of data types supported by this HBA. The data types are numbered from 0 to 30, of which 0 through 15 are reserved for Apple definition and 16 through 30 are available for vendor use. Returns 0.

`ataIOpbSize` This field specifies the size of the parameter block supported. This field contains the size of the I/O parameter block.

`ataMaxIOpbSize` The field specifies the maximum I/O size for the HBA. This field is currently not supported and returns 0.

`ataFeatureFlags` This field specifies supported features. Not supported; returns 0.

`ataVersionNum` The version number of the HBA is returned. The current version returns a 1.

`ataHBAInquiry` Reserved

`ataHBAPrivPtr` This field contains a pointer to the HBA's private data area. Not supported; returns 0.

`ataHBAPrivSize` This field contains the byte size of HBA's private data area. Not supported; returns 0.

`ataAsyncFlags` These flags indicate which types of asynchronous events the HBA is capable of generating. Not supported; returns 0.

`ataHBAVendor` This field contains the vendor ID of HBA. This is an ASCII text field. Not supported.

`ataContrlFamily` Reserved.

`ataContrlType` This field identifies the specific type of ATA controller. Not supported; returns 0.

`ataXPTversion` Reserved.

`ataHBAvesion` This field specifies the version of the HBA. Not supported; returns 0.

`ataHBAslotType` This field specifies the type of slot. Not supported; returns 0.

`ataHBAslotNum` This field specifies the slot number of the HBA. Not supported; returns 0.

RESULT CODES

noErr	0	Successful completion, no error occurred
ATAMgrNotInitialized	-1802	ATA Manager not initialized

ATA I/O Queue Release

You can use the ATA_QRelease function to release the frozen I/O queue of the selected device.

When an I/O error is detected by the ATA Manager and the QLockOnError bit of the parameter block is set for the request, the queue for the selected device will be frozen. No pending or new requests can be processed or receive status until the queue is released through this function. Only those requests with the Immediate bit set in the ataFlags field of the ataPBHdr parameter block will be processed. Consequently, the ATA_QRelease function is issued with the Immediate bit set in the parameter block. The noErr status is returned for ATA I/O queue release commands issued while the queue isn't frozen.

The manager function code for the ATA_QRelease function is $04.

There are no additional function-specific variations on ataPBHdr for this call.

RESULT CODES

noErr	0	Successful completion, no error occurred
nsDrvErr	-56	Specified drive does not exist
ATAMgrNotInitialized	-1802	ATA Manager not initialized

IDE NOP

The ATA_NOP function performs no operation across the interface at all and does not change the state of either the manager or the device. It returns noErr if the drive number is valid.

The manager function code for the ATA_NOP function is $00.

There are no additional function-specific variations on ataPBHdr for this call.

RESULT CODES

noErr	0	Successful completion, no error occurred
nsDrvErr	-56	Specified drive does not exist

ATA Manager Command Termination

You can use the ATA_Abort function to terminate a specified queued I/O request. This function applies to asynchronous I/O requests only. This function searches through the I/O queue associated with the selected device and aborts the matching I/O request. The current implementation does not abort if the specified command is in progress. If the specified I/O request is not found or has started processing, an ATAUnableToAbort status is returned. If aborted, the ATAReqAborted status is returned.

It is up to the application that called the ATA_Abort function to clean up the aborted request, which includes parameter block deallocation and O/S reporting.

The manager function code for the ATA_Abort function is $10.

The parameter block associated with the ATA_Abort function is defined as follows:

```
typedef      struct
{
   ataPBHdr
   ATA_PB*   AbortPB       /* → address of the parameter block of
                              the function to be aborted */
   ushort    Reserved      /* reserved */
} ATA_Abort;
```

Field descriptions

ataPBHdr	See the ataPBHdr parameter block definition on page 3-14.
AbortPB	This field contains the address of the I/O parameter block to be aborted.

RESULT CODES

noErr	0	Successful completion, no error occurred
nsDrvErr	-56	Specified drive does not exist
ATAMgrNotInitialized	-1802	ATA Manager not initialized
ATAReqAborted	-1810	The request was aborted
ATAUnableToAbort	-1811	Request to abort couldn't be honored

Device Registers Access

You can use the ATA_RegAccess function to enable access to a particular device register of a selected device. This function is used for diagnostic and for error recovery processing.

The manager function code for the ATA_RegAccess function is $12.

The parameter block associated with this function is defined as follows:

```
typedef        struct
{
   ataPBHdr
   ushort      RegSelect;      /* → device register selector */
   uchar       RegValue;       /* ↔ register value
                                  to read or to be written */
   uchar       Reserved;       /* used for data register access */
   uchar       Reserved[22]    /* reserved */
} ATA_RegAccess;
```

Field descriptions

`ataPBHdr` See the `ataPBHdr` parameter block definition on page 3-14

`RegSelect` This field specifies which device registers to access. The selectors for the registers supported by the `ATA_RegAccess` function are listed here:

Selector name	Selector	Register description
`DataReg`	0	Data register (16-bit access only)
`ErrorReg`	1	Error register (read) or features register (write)
`SecCntReg`	2	Sector count register
`SecNumReg`	3	Sector number register
`CylLoReg`	4	Cylinder low register
`CylHiReg`	5	Cylinder high register
`SDHReg`	6	SDH register
`StatusReg`	7	Status register (read) or command register (write) `CmdReg`
`AltStatus`	14	Alternate status (read) or device control (write) `DevCntr`
`AddrReg`	15	Digital input register

`RegValue` This field contains the value to be written (`IDEioDirection` has a value of `01` binary for writes) or the value read from the selected register (`ATAioDirection` has a value of `10` binary for reads). If the `DataReg` register selector is specified, this field as a 16-bit field; for the other register selectors it is an 8-bit field.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	-56	Specified drive does not exist

ATA Drive Identification

You can use the `ATA_Identify` function to retrieve the device identification data from the selected hard drive. The device identification data contains the information necessary to perform I/O with the device. Refer to the ATA specification for the format of the data returned and the information provided by the data.

The manager function code for the `ATA_Identify` function is $13.

The parameter block associated with this function is defined as follows:

```
typedef     struct
{
  ataPBHdr
  ushort    Reserved1[4];     /* reserved */
  uchar     *DataBuf;         /* ↔ buffer for the data */
  ushort    Reserved2[18];    /* reserved */
} ATA_Identify;
```

Field descriptions

`ataPBHdr` See the `ataPBHdr` parameter block definition on page 3-14.

`DataBuf` Pointer to the data buffer for the device identify data. The length of the buffer must be at least 512 bytes.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	-56	Specified drive does not exist

IDE Bus Reset

You can use the `ATA_ResetBus` function to reset the specified IDE bus. This function performs a soft reset operation to the selected IDE bus. The ATA interface doesn't provide a means of resetting individual units on the bus, so all devices on the bus are affected.

Note
This function should be used with caution since it may terminate any active requests to devices on the bus. ◆

The manager function code for the `ATA_ResetBus` function is $11.

The parameter block associated with this function is defined as follows:

```
typedef     struct
{
  ataPBHdr
  char      Status;       /* ← last ATA status register image */
  char      Reserved;     /* reserved */
  ushort    Reserved[23]; /* reserved */
} ATA_ResetBus;
```

Field descriptions

`ataPBHdr` See the `ataPBHdr` parameter block definition on page 3-14.

`Status` This field contains the last device status register image following the bus reset. See the ATA specification for definitions of the status register bits.

RESULT CODES

noErr	0	Successful completion, no error occurred
nsDrvErr	-56	Specified drive does not exist

ATA Manager Shutdown

You can use the ATA_MgrShutDown function to shut down the ATA Manager. It is the complement to the ATA_Init function. This function deallocates all of the global space currently in use by the ATA Manager. After calling ATA_MgrShutDown, the ATA Manager must be reinitialized before any IDE I/O requests can take place.

Note
This function should be used with caution if multiple client applications are present. This function always returns a status of noErr. ◆

The manager function code for the ATA_MgrShutdown function is $92.

There are no additional function-specific variations on ataPBHdr for this call.

RESULT CODES

noErr	0	Successful completion, no error occurred

Driver Reference Number Registration

You can use the ATA_DrvrRegister function to register the driver reference number for the selected drive. The function doesn't check for the existence of another driver.

The manager function code for the ATA_DrvrRegister function is $85.

The parameter block associated with this function is defined as follows:

```
typedef       struct
{
   ataPBHdr
   short      drvrRefNum;      /* → driver reference number */
   ushort     FlagReserved;    /* reserved */
   ushort     deviceNextID;    /* not used */
   short      Reserved[21];    /* reserved */
} ATA_DrvrRegister;
```

Field descriptions

ataPBHdr	See the ataPBHdr parameter block definition on page 3-14.
drvrRefNum	This field specifies the driver reference number to be registered. This value must be less than 0 to be valid.
deviceNextID	Not used by this function.

RESULT CODES

noErr	0	Successful completion, no error occurred
nsDrvErr	-56	Specified drive does not exist

Driver Reference Number Deregistration

You can use the ATA_DrvrDeregister function to deregister the driver reference number for the selected drive. After successful completion of this function, the value for the driver reference number is set to 0, which indicates that there is no driver in control of this device.

The manager function code for the ATA_DrvrDeregister function is $87.

There are no additional function-specific variations on ataPBHdr for this call.

RESULT CODES

noErr	0	Successful completion, no error occurred
nsDrvErr	-56	Specified drive does not exist

Driver Reference Number Retrieval

You can use the ATA_FindRefNum function to determine if a driver has been installed for a given device. You pass a device ID and the function returns the reference number for the current driver registered for the specified device. A value of 0 indicates that no driver has been registered. The deviceNextID field contains a device ID of the next device in the list. The end of the list is indicated with a value of 0x 00FF.

To create a list of all drivers for the attached devices, pass in 0x 00FF for deviceID. This causes deviceNextID to be filled with the first device in the list. Each successive driver can be found by moving the value returned in deviceNextID into the deviceID field until the function returns 0x 00FF in deviceNextID, which indicates the end of the list.

The manager function code for the ATA_FindRefNum function is $86.

The parameter block associated with this function is defined as follows:

```
typedef       struct
{
   ataPBHdr
   short      drvrRefNum;     /* ← driver reference number */
   ushort     FlagReserved;   /* reserved */
   ushort     deviceNextID;   /* ← next drive ID */
   short      Reserved[21];   /* reserved */
} ATA_FindRefNum;
```

Field descriptions

`ataPBHdr`	See the `ataPBHdr` parameter block definition on page 3-14.
`drvrRefNum`	Upon return, this field contains the reference number for the device specified in the `deviceID` field of the `ataPBHdr` data.
`deviceNextID`	Upon return, this field contains the `deviceID` of the next device on the list.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`nsDrvErr`	-56	Specified drive does not exist

Error Code Summary

A summary of the ATA Manager error codes is provided in Table 3-2.

Table 3-2 IDE hard disk drive error codes

Error code	Name	Description
0	`noErr`	Successful completion, no error occurred
-17	`controlErr`	Unimplemented control call; could not complete requested operation
-18	`statusErr`	Unimplemented status call; could not complete requested operation
-23	`openErr`	Unimplemented open call; could not complete requested operation
-36	`ioErr`	I/O error
-50	`paramErr`	Invalid parameter specified
-56	`nsDrvErr`	Specified drive does not exist; no device attached to the specified port
-65	`offLinErr`	No disk in drive (removable media)
-1780	`AT_AbortErr`	Command aborted bit set in error register
-1781	`AT_RecalErr`	Track 0 Not Found bit set in error register
-1782	`AT_WrFltErr`	Write fault detected by device
-1783	`AT_SeekErr`	Seek Complete bit not set upon completion
-1784	`AT_UncDataErr`	Uncorrected Data bit set in status register; possible bad data

continued

Table 3-2 IDE hard disk drive error codes (continued)

Error code	Name	Description
-1785	AT_CorDataErr	Data Corrected bit set in error register; was corrected (good data) - notification
-1786	AT_BadBlkErr	Bad Block bit set in error register
-1787	AT_DMarkErr	Data Mark Not Found bit set in error register
-1788	AT_IDNFErr	ID Not Found bit set in error register
-1789	AT_DRQErr	Timeout waiting for DRQ active
-1790	ATABusyErr	Selected device busy (BUSY bit set)
-1793	DRVRCantAllocate	Global memory allocation error
-1794	NoATAMgr	No ATA Manager installed in the system (MgrInquiry failure)
-1795	ATAInitFail	ATA Manager initialization failed
-1796	ATABufFail	Device buffer test failed
-1802	ATAMgrNotInitialized	ATA Manager not initialized
-1803	ATAPBInvalid	Invalid device base address detected
-1804	ATAFuncNotSupported	An unknown manager function code was specified
-1805	ATABusy	Selected device is busy; the device isn't ready to go to next phase yet
-1806	ATATransTimeOut	Timeout: transaction timeout detected
-1807	ATAReqInProg	I/O channel in use - can not proceed
-1808	ATAUnknownState	The device status register reflects an unknown state
-1809	ATAQLocked	I/O queue for the port is locked due to a previous I/O error (must be unlocked before continuing)
-1810	ATAReqAborted	The I/O queue entry was aborted due to an abort command
-1811	ATAUnableToAbort	The I/O queue entry could not be aborted; too late to abort or the entry point was not found

Index

N

O

P, Q

R

S–U

V–Z