Developer Note

ATA Device Software Guide

ATA Device Software for Macintosh Computers

Developer Note

© Apple Computer, Inc. 1996

Apple Computer, Inc.
© 1996 Apple Computer, Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Computer, Inc. Printed in the United States of America.

The Apple logo is a trademark of Apple Computer, Inc.
Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this book. Apple retains all intellectual property rights associated with the technology described in this book. This book is intended to assist application developers to develop applications only for Apple Macintosh computers.

Every effort has been made to ensure that the information in this manual is accurate. Apple is not responsible for printing or clerical errors.

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

Apple, the Apple logo, APDA, AppleLink, Mac, Macintosh, and Power Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Apple Desktop Bus is a trademark of Apple Computer, Inc.

Adobe Illustrator is a trademark of Adobe Systems Incorporated, which may be registered in certain jurisdictions.

Helvetica and Palatino are registered trademarks of Linotype/AG and?or its subsidiaries.

ITC Zapf Dingbats is a registered trademark of International Typeface Corporation.

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 v

Preface About This Note vii

Contents of This Note vii
Supplemental Reference Documents vii
The *Apple Developer Catalog* viii
Apple Developer World Web Site viii
Typographical Conventions viii

Chapter 1 Software for ATA Devices 1

Introduction to ATA Software 2
ATA Disk Driver 3
ATA Manager 4

Chapter 2 ATA Driver Reference 5

High-Level Device Manager Routines 6
ATA Disk Driver Control and Status Functions 8

Chapter 3 ATA Manager Reference 21

The ATA Parameter Block 22
Setting Data Transfer Timing 28
Setting Up PIO Data Transfers 29
Setting Up Multiword and Singleword DMA Data Transfers 29
Functions 29

Appendix Result Code Summary 59

Figures and Tables

Chapter 1 Software for ATA Devices 1

Figure 1-1 Relationship of the ATA Manager to the Macintosh system architecture 2

Chapter 2 ATA Driver Reference 5

Table 2-1 Status functions supported by the ATA disk driver 7
Table 2-2 Control function supported by the ATA disk driver 8

Chapter 3 ATA Manager Reference 21

Table 3-1 Control bits in the `ataPBFlags` field 25
Table 3-2 ATA Manager functions 30
Table 3-3 ATA register selectors for `RegSelect` field 40
Table 3-4 Register mask selectors 40

Appendix Result Code Summary 59

Table A-1 ATA Manager result codes 59

About This Note

This developer note describes the system software that controls ATA (AT-Attachment) devices, such as ATA hard disk drives (sometimes referred to as integrated drive electronics (IDE) drives) installed in a Macintosh computer. This note also provides information for ATAPI (ATA Packet Interface) CD-ROM and PCMCIA (Personal Computer Memory Card International Association) devices. It is intended to help experienced Macintosh hardware and software developers design compatible products. If you are unfamiliar with Macintosh computers or would simply like more technical information, you may wish to read the related technical manuals listed in the section "Supplemental Reference Documents," later in this preface.

To use the information in this developer note, 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 ANSI specification X3.279-1996, "AT Attachment Interface with Extensions (ATA-2)."

Contents of This Note

This note contains three chapters, an appendix, and an index:

- Chapter 1, "Software for ATA Devices," introduces the system software components that control ATA devices installed in a Macintosh computer.
- Chapter 2, "ATA Driver Reference," describes the Macintosh device driver routines provided by the ATA disk driver.
- Chapter 3, "ATA Manager Reference," defines the data structures and functions that are specific to version 3.0 of the ATA Manager.
- The appendix, "Result Code Summary," lists the possible result codes returned by the ATA device software for the Mac OS.

This developer note also contains an index.

Supplemental Reference Documents

Developers may need copies of the appropriate Apple reference books and should have the relevant books of the *Inside Macintosh* series. Developers should also have *Designing PCI Cards and Drivers for Power Macintosh Computers*. These books are available in technical bookstores and through the Apple Developer Catalog.

The *Apple Developer Catalog*

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

To order products or to request a complimentary copy of the *Apple Developer Catalog*, contact

Apple Developer Catalog
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	ORDER.ADC
Internet	http://www.devcatalog.apple.com

Apple Developer World Web Site

The Apple Developer World web site is the one-stop source for finding technical and marketing information specifically for developing successful Macintosh-compatible software and hardware products. Developer World is dedicated to providing developers with up-to-date Apple documentation for existing and emerging Macintosh technologies. Developer World can be reached at http://www.devworld.apple.com.

Typographical Conventions

This developer note uses the following 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.

Note

A note like this contains information that is interesting but not essential for an understanding of the text. ◆

IMPORTANT

A note like this contains important information that you should read before proceeding. ▲

CHAPTER 1

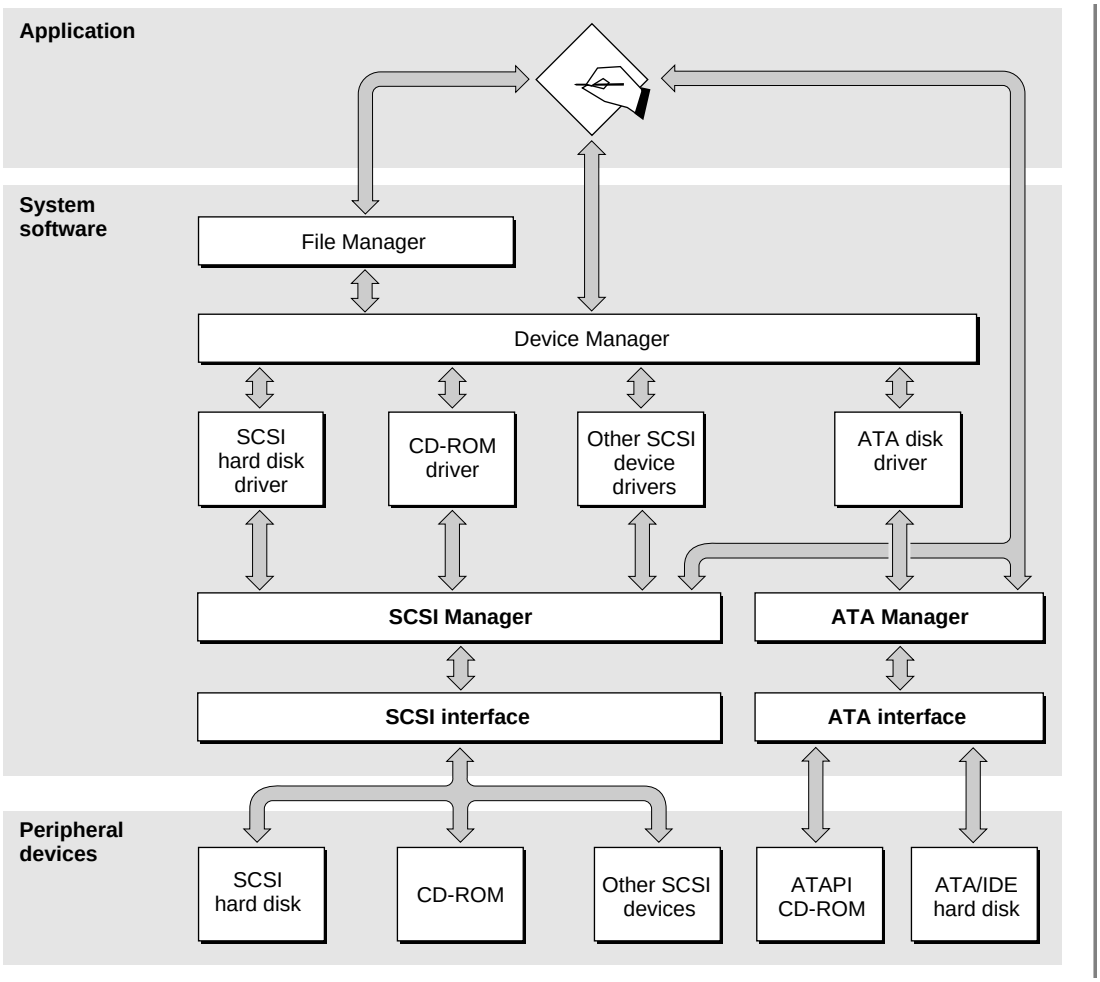
Software for ATA Devices

This chapter introduces the Macintosh system software that controls ATA (AT-Attachment) devices, such as ATA hard disk drives, and provides data transport services for CD-ROM drives that use the ATAPI (ATA Packet Interface) protocol and ATA/PCMCIA (ATA Personal Computer Memory Card International Association) devices that use the Mac OS Card Services.

Introduction to ATA Software

Support for ATA devices is incorporated in the Macintosh ROM software (firmware). System software for controlling ATA devices is provided by the ATA disk driver, which the ATA Manager loads into RAM from the drive media. The relationship of the ATA disk driver and the ATA Manager is shown in Figure 1-1.

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



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

ATA disk drives and CD-ROM drives appear on the desktop the same way SCSI disk drives currently do. Except for applications that perform low-level services, such as formatting and partitioning utilities, applications interact with the ATA disk drives in a device-independent manner through the File Manager.

The ATA Manager provides support for ATAPI and ATA/PCMCIA data transport services, and does not provide access to PCMCIA tuples. Any client that wants to get PCMCIA tuple information must do so through the PCMCIA Card Services.

ATA Disk Driver

The ATA disk driver has the following features:

- It uses the ATA Manager for system and bus independence.
- It supports multiple partitions (volumes).
- It recognizes both partitioned and nonpartitioned Macintosh media.
- It adheres to the new driver rules described in *Designing PCI Cards and Drivers for the Macintosh Family.*
- It supports both synchronous and asynchronous requests from the file system.

The ATA disk driver supports all ATA drives that adhere to the ANSI ATA specification X3.279-1996.

The ATA disk driver relies on the services of the ATA Manager, which provides the ATA protocol engine and relieves the driver of system and bus dependencies. The main functions of the driver are managing the media and monitoring the status of the drive.

The ATA disk driver is responsible for providing block-oriented access to the storage media. The file systems treat the media as one or more logical partitions or volumes in which data at any address can be read or written indefinitely.

The ATA disk driver provides operating system–dependent services through a set of driver routines required to interface with the Mac OS. In addition, the ATA disk driver provides control and status functions that are specific to this implementation of the ATA disk driver. The required disk driver routines, as specified in *Inside Macintosh: Devices*, are `open`, `close`, `prime`, `control`, and `status`.

There are two versions of the ATA disk driver: a RAM-based version, which is installed on the drive media by the Drive Setup application, and a ROM-resident version. At system startup time, if the ATA Manager does not find a RAM-based driver on the ATA drive media, the ATA disk driver in the ROM is selected as the driver for the drive. Note that this is different from the SCSI driver-loading sequence, which always requires that a RAM-based driver be installed on the media. The ATA disk driver in ROM is a subset of

the ATA disk driver on the media and should not be used for normal operation. The ATA disk driver in ROM provides emergency access to the ATA drive. The ATA disk driver installed on the media by the Drive Setup application provides the latest features and optimal performance.

The RAM-based ATA disk driver supports all modes of PIO and DMA operations as defined in the ANSI X3.279-1996 ATA-2 specification. When the driver is opened for an ATA drive, the ATA disk driver configures the ATA Manager and the drive for optimal performance based upon both the system and drive capabilities. Typically, DMA modes are selected over PIO modes.

The ATA disk driver supports conservation of system power by spinning down the disk drive to reduce power consumption. Spinning down the drive also flushes the drive write cache to prevent data loss. The ATA disk driver spins down the disk drive in response to a sleep demand, the "Set Power Mode" control call (`csCode` 70), and system shutdown and restart and when no access has been made to the drive within the time specified in the Energy Saver control panel.

The ATA disk driver usually has a driver reference number of –54 (decimal) but may also have a different reference number if –54 is taken when the driver is loaded. The driver name is `.ATDISK`. Like all Macintosh device drivers, the ATA disk driver can be called by using either the driver reference number or the driver name, `.ATDISK`.

The ATA disk driver does not provide request queuing. All driver requests either are completed immediately or are passed to the ATA Manager for further processing. However, the driver does process asynchronous requests, using the ATA Manager to notify the driver when an operation has completed.

ATA Manager

The ATA Manager manages the ATA controller and its protocol. The ATA Manager provides data transport services between ATA devices and the system, directing commands to the appropriate device and handling interrupts from the devices.

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

When making calls to the ATA Manager, you have to pass and retrieve parameter information through a parameter block. The size and content of the parameter block depend on the function being called. However, all calls to the ATA Manager have 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 ATA Manager functions. The additional parameter block data types, which are specific to the function being called, are described in Chapter 3, "ATA Manager Reference."

CHAPTER 2

ATA Driver Reference

This chapter describes the Macintosh device driver routines provided by the ATA disk driver. The information in this chapter assumes that you are already familiar with how to use device driver services on the Macintosh computer. If you are not familiar with Macintosh device drivers, refer to the chapter "Device Manager" in *Inside Macintosh: Devices* for additional information.

High-Level Device Manager Routines

The ATA disk driver supports the required set of routines for handling requests from the Device Manager, as defined in the chapter "Device Manager" of *Inside Macintosh: Devices*. Those routines are briefly defined here for convenience. Additional control functions supported in the ATA disk driver are defined in "ATA Disk Driver Control and Status Functions" beginning on page 8.

open

The `open` routine should not be called to open the ATA disk driver. The ATA disk driver requires a physical drive ID from the ATA Manager and is called by the ATA Manager after being loaded from the drive media. An open call to the ATA disk driver returns a result of `openErr` if it has not been opened previously and returns a result of `noErr` and does not reopen if it is already open.

When opened, the ATA disk driver initializes itself for the drive specified and registers itself for control of the drive with the ATA Manager. The driver installs itself in the system unit table and installs a system drive queue entry for each file system partition (volume) found on the media. After opening the ATA disk driver is able to respond to all other `close`, `prime`, `status`, and `control` calls.

RESULT CODES

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

close

The `close` routine instructs the ATA disk driver to terminate execution. The driver deregisters for control of the drive with the ATA Manager, removes the drive queue entries for each volume associated with the drive, and deallocates all memory used during operation. The driver does not remove itself from the unit table.

RESULT CODE

`noErr`	0	Successful completion; no error occurred

prime

The `prime` routine performs logical block read and write operations to a specified volume with automatic retries on errors. The driver accepts either the standard 32-bit address or a 64-bit large volume address, both of which must be aligned on a 512-byte boundary representing a logical block address on the volume. The `prime` routine performs either a `read` or `write` command, as specified by the caller.

RESULT CODES

`noErr`	0	Successful completion, no error occurred
`ioErr`	–36	I/O error
`paramErr`	–50	Invalid parameter specified
`nsDrvErr`	–56	No such drive installed

status

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

Table 2-1 shows the status functions supported by the ATA disk driver.

Table 2-1 Status functions supported by the ATA disk driver

Value of csCode	Definition
8	Return drive status information.
43	Return driver gestalt information.
44	Return boot partition.
45	Return partition mounting status.
46	Return partition write protect status.
70	Power mode status information.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`statusErr`	–18	Unimplemented status function; could not complete requested operation
`nsDrvErr`	–56	No such drive installed

control

The ATA driver implements many of the control functions supported by the SCSI hard disk driver and defined in *Inside Macintosh: Devices*. The ATA disk driver also implements several new functions defined in *Designing PCI Cards and Drivers for Power Macintosh Computers*. The control functions are listed in Table 2-2 and described in "ATA Disk Driver Control and Status Functions."

Table 2-2 Control function supported by the ATA disk driver

Value of csCode	Definition
5	Verify media
6	Format media
7	Eject media
21	Return drive icon
22	Return media icon
23	Return drive characteristics
44	Enable partition as startup partition
45	Enable partition to be mounted
46	Set partition write protected
48	Disable partition mounting
49	Disable partition write protection
60	Mount volume
70	Set power mode

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`controlErr`	–17	Unimplemented control function; could not complete requested operation
`nsDrvErr`	–56	No such drive installed

ATA Disk Driver Control and Status Functions

The ATA disk driver supports a standard set of control functions for ATA disk drive devices. The functions are used for control, status, and power management.

In the function descriptions, an arrow preceding a parameter indicates whether the parameter is an input parameter, an output parameter, or both.

Arrow	Meaning
→	Input
←	Output
↔	Both

verify

The `verify` control function requests a read verification of the data on the ATA hard drive media. This function performs no operation and returns `noErr` if the logical drive number is valid.

Parameter block

→	`csCode`	A value of 5.
→	`ioVRefNum`	The logical drive number.
→	`csParam[]`	None defined.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

format

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

Parameter block

→	`csCode`	A value of 6.
→	`ioVRefNum`	The logical drive number.
→	`csParam[]`	None defined.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

eject

The `eject` control function is used by the driver to determine when a volume becomes unmounted. If the unmounted volume was the last mounted volume of the drive, the drive is placed in a low-power mode to conserve power. If the drive is also ejectable (PCMCIA, for example), a drive ejection is initiated.

Parameter block

→	`csCode`	A value of 7.
→	`ioVRefNum`	The logical drive number.
→	`csParam[]`	None defined.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

return drive icon

The `return drive icon` control function returns a pointer to the device icon and the device location string.

Parameter block

→	`csCode`	A value of 21.
→	`ioVRefNum`	The logical drive number.
→	`csParam[]`	None defined.
←	`csParam[0-1]`	Address of drive icon and location string (information is in `ICN#` format).
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

return media icon

The `return media icon` control function returns a pointer to the media icon and the location string. The media icon will differ depending on the media (hard drive, CD-ROM drive, or PCMCIA drive).

Parameter block

→	`csCode`	A value of 22.
→	`ioVRefNum`	The logical drive number.
→	`csParam[]`	None defined.
←	`csParam[0–1]`	Address of drive icon and location string (information is in `ICN#` format).
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

return drive characteristics

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

Parameter block

→	`csCode`	A value of 23.
→	`ioVRefNum`	The logical drive number.
→	`csParam[]`	None defined.
←	`csParam[0–1]`	Drive information. $0601 = primary, fixed, SCSI, internal $0201 = primary, removable, SCSI, internal
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

enable startup partition

The `enable startup partition` control function enables the specified partition to be the startup (boot) partition. The partition is specified by either its logical drive number or its block address on the media. The current entry for the boot partition is cleared. A result of `controlErr` is returned if the partition does not have a partition map entry on the media or could not be enabled as the startup partition.

Parameter block

→	`csCode`	A value of 44.

→	`ioVRefNum`	The logical drive number, or 0 if using partition block address.
→	`csParam[]`	The partition block address (long) if `ioVRefNum` = 0.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`controlErr`	–17	Unimplemented control function; could not complete requested operation
`nsDrvErr`	–56	The specified logical drive number does not exist

enable partition mounting

The `enable partition mounting` control function enables the specified partition to be mounted when the drive is recognized. The partition is specified by either its logical drive number or its block address on the media. A result of `controlErr` is returned if the partition does not have a partition map entry on the media or could not be enabled for mounting.

Parameter block

→	`csCode`	A value of 45.
→	`ioVRefNum`	The logical drive number, or 0 if using partition block address.
→	`csParam[]`	The partition block address (long) if `ioVRefNum` param = 0.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`controlErr`	–17	Unimplemented control function; could not complete requested operation
`nsDrvErr`	–56	The specified logical drive number does not exist

enable partition write protect

The `enable partition write protect` control function enables software write protection on the specified partition. The partition is specified by either its logical drive number or its block address on the media. A result of `controlErr` is returned if the partition does not have a partition map entry on the media or write protection could not be enabled for the partition.

Parameter block

→	`csCode`	A value of 46.

→	`ioVRefNum`	The logical drive number, or 0 if using partition block address.
→	`csParam[]`	The partition block address (long) if `ioVRefNum` = 0.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`controlErr`	–17	Unimplemented control function; could not complete requested operation
`nsDrvErr`	–56	The specified logical drive number does not exist

clear partition mounting

The `clear partition mounting` control function prevents a partition from being mounted when the drive is recognized. The partition is specified by either its logical drive number or its block address on the media. A result of `controlErr` is returned if the partition does not have a partition map entry on the media or partition mounting could not be cleared.

Parameter block

→	`csCode`	A value of 48.
→	`ioVRefNum`	The logical drive number, or 0 if using partition block address.
→	`csParam[]`	The partition block address (long) if `ioVRefNum` param = 0.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`controlErr`	–17	Unimplemented control function; could not complete requested operation
`nsDrvErr`	–56	The specified logical drive number does not exist

clear partition write protect

The `clear partition write protect` control function disables software write protection for the specified partition. The partition is specified by either its logical drive number or its block address on the media. A result of `controlErr` is returned if the partition does not have a partition map entry on the media or write protection could not be cleared.

Parameter block

→	`csCode`	A value of 49.

→	`ioVRefNum`	The logical drive number, or 0 if using partition block address.
→	`csParam[]`	The partition block address (long) if `ioVRefNum` = 0.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`controlErr`	–17	Unimplemented control function; could not complete requested operation
`nsDrvErr`	–56	The specified logical drive number does not exist

mount volume

The `mount volume` control function instructs the drive to post a disk inserted event for the specified partition. The partition is specified by either its logical drive number or its block address on the media.

Parameter block

→	`csCode`	A value of 60.
→	`ioVRefNum`	The logical drive number, or 0 if using partition block address.
→	`csParam[]`	The partition block address (long) if `ioVRefNum` = 0.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`controlErr`	–17	Unimplemented control function; could not complete requested operation
`nsDrvErr`	–56	The specified logical drive number does not exist

set power mode

The `set power mode` control function changes the drive power mode to one of four modes: active, idle, standby, and sleep. This function can be used to reduce drive power consumption.

In active mode the drive interface is active, the spindle motor is running at full speed, and the device is respnding to commands.

In idle mode the nonessential electronics on the ATA hard drive are disabled. For example, the read and write channels are disabled during the idle state. The spindle motor remains enabled during the idle state, so the drive still responds immediately to any commands requesting media access.

In standby mode the head is parked and the spindle motor is disabled. The drive interface remains active and is still capable of responding to commands. However, it can

take several seconds to respond to media access commands, because the drive's spindle motor must return to full speed before media access can take place.

In sleep mode both the drive interface and the spindle motor are disabled. The driver must reset and reconfigure the drive before another access to the drive can be made. Since many drives do not support sleep mode and because there is little power savings difference between standby and sleep modes, the ATA disk driver may put the drive in standby mode instead.

Parameter block

→	`csCode`	A value of 70.
→	`ioVRefNum`	The logical drive number.
→	`csParam[0]`	The most significant byte contains one of the following codes: 0 = enable active mode 1 = enable standby mode 2 = enable idle mode 3 = enable sleep mode
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`controlErr`	–17	The power management information couldn't be returned due to a manager error
`nsDrvErr`	–56	The specified logical drive number does not exist

drive info

The ATA disk driver provides a drive status function for retrieving status information from the drive. The `drive info` status function returns the same type of information that disk drivers are required to return for the `status` function, as described in the chapter "Device Manager" in *Inside Macintosh: Devices*.

Parameter block

→	`csCode`	A value of 8.
→	`ioVRefNum`	The logical drive number.
→	`csParam[]`	Contains status information about the internal ATA disk drive.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

driver gestalt

The `driver gestalt` status function provides the application information about the ATA disk 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 ATA gestalt structure. Refer to *Designing PCI Cards and Drivers for the Macintosh Family* for information related to the ATA gestalt structure.

The fields `driverGestaltSelector` and `driverGestaltResponse` are 32-bit fields that contain the gestalt selector and possible responses. The selectors and responses are defined in the parameter block definition.

Parameter block

→	`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 selectors:
		`sync` Indicates synchronous or asynchronous driver.
		`devt` Specifies type of device the driver is controlling.
		`intf` Specifies the device interface.
		`boot` Specifies PRAM value to designate this driver or device.
		`vers` Specifies the version number of the driver.
		`lpwr` Indicates low-power mode support.
		`purg` Request if the driver can be closed and or purged.
		`wide` Indicates large-volume support.
		`ejec` Specifies eject control function requirements.
←	`driverGestaltResponse`	Returned result based on the driver gestalt selector. The possible four-character return values are:
		`TRUE` If the `sync` driver selector is specified, this Boolean value indicates that the driver is synchronous; a value of `FALSE` indicates asynchronous.
		`'disk'` If the `devt` driver selector is specified, this value indicates a hard disk driver.
		`'ide '` If the `intf` driver selector is specified, this value indicates the interface is ATA.
		nnnn If the `vers` selector is specified, the current version number of the driver is returned.

		TRUE	If the `lpwr` selector is specified, this value indicates the power mode control and status function are supported.
		TRUE	If the `wide` selector is specified, this value indicates the driver supports large volumes.
		value	If the `ejec` selector is specified, this value indicates when the `ejec` call should be made. Bit 0, if set, means don't issue eject call on restart. Bit 1, if set, means don't issue eject call on shutdown. If the `purg` selector is specified, this value indicates whether the driver can close and/or be purged from memory. A value of 0 indicates that the driver cannot be closed. A value of 3 indicates that the driver can be closed but not purged. A value of 7 indicates that the driver can be both closed and purged from memory.
←	`ioResult`		See result codes.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`statusErr`	–18	Unknown selector was specified
`nsDrvErr`	–56	The specified logical drive number does not exist

get startup partition

The `get startup partition` status function returns 1 if the specified partition is the startup partition, 0 if it is not. The partition is specified by either its logical drive number or its block address on the media.

Parameter block

→	`csCode`	A value of 44.
→	`ioVRefNum`	The logical drive number, or 0 if using partition block address.
→	`csParam[]`	The partition block address (long) if `ioVRefNum` = 0.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

get partition mount status

The `get partition mount status` status function returns 1 if the specified partition has mounting enabled, 0 if not enabled or the partition does not have a partition map entry on the media. The partition is specified by either its logical drive number or its block address on the media.

Parameter block

→	`csCode`	A value of 46.
→	`ioVRefNum`	The logical drive number, or 0 if using partition block address.
→	`csParam[]`	The partition block address (long) if `ioVRefNum` = 0.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

get partition write protect status

The `get partition write protect status` status function returns 1 if the specified partition is software write protected, 0 if it is not. The partition is specified by its either logical drive number or its block address on the media.

Parameter block

→	`csCode`	A value of 45.
→	`ioVRefNum`	The logical drive number, or 0 if using partition block address
→	`csParam[]`	The partition block address (long) if `ioVRefNum` = 0.
←	`ioResult`	See result codes.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`nsDrvErr`	–56	The specified logical drive number does not exist

get power mode

The `get power mode` status function returns the current power mode state of the internal hard disk.

Parameter block

→	`csCode`	A value of 70.
→	`ioVRefNum`	The logical drive number.
→	`csParam[]`	None defined.
←	`csParam[]`	The most significant byte 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 result codes.

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`statusErr`	–18	The power management information couldn't be returned due to a manager error
`nsDrvErr`	–56	The specified logical drive number does not exist

CHAPTER 3

ATA Manager Reference

This chapter defines the data structures and functionsspecific to version 3.0 of the ATA Manager. The section "The ATA Parameter Block" shows the data structure of the ATA parameter block. Version 3.0 of the ATA Manager supports DMA data transfers. The section "Setting Data Transfer Timing," discusses how the ATA Manager interacts with ATA devices to set up DMA transfers. 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 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 `ataPBHdr` data type defines the ATA parameter block.

ATA Manager 3.0 defines ATA parameter block version 3, which is required for the specification of ANSI ATA-2 compliant transfer timings, and DMA timing in particular. Parameter block versions 1 and 2 are still supported, but full use of version 3 is recommended when the best data transfer performance of the device is required.

The parameter block includes a field, `ataPBFunctionCode`, in which you specify the function selector for the particular function 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 following unique `typedef` identifiers are used in the ATA Manager parameter block and function definitions:

`SInt8`	A signed 8-bit field
`SInt16`	A signed 16-bit field
`SInt32`	A signed 32-bit field
`UInt8`	An unsigned 8-bit field
`UInt16`	An unsigned 16-bit field
`UInt32`	An unsigned 32-bit field

The ATA parameter block header structure is defined as follows:

```
typedef    struct    ataPBHdr       /* ATA Manager parameter
                                        block header structure */
{
           Ptr       ataPBLink;     /* Reserved, set to 0 */
```

```
            SInt16    ataPBQType;    /* Type byte */
            UInt8     ataPBVers;     /* → Parameter block
                                        version number */
            UInt8     ataPBReserved; /* Reserved */
            Ptr       ataPBReserved2; /* Reserved */
            ProcPtr   ataPBCallbackPtr; /* Universal completion
                                        routine pointer */
            OSErr     ataPBResult;   /* ← Returned result */
            UInt8     ataPBFunctionCode; /* → Manager function
                                         code */
            UInt8     ataPBIOSpeed;  /* → I/O timing class */
            UInt16    ataPBFlags;    /* → Control options */
            SInt16    ataPBReserved3; /* Reserved */
            long      ataPBDeviceID; /* → Device ID */
            UInt32    ataPBTimeOut;  /* → Transaction timeout
                                       value */
            Ptr       ataPBClientPtr1; /* Client storage Ptr 1 */
            Ptr       ataPBClientPtr2; /* Client storage Ptr 2 */
            UInt16    ataPBState;    /* Reserved, init to 0 */
            SInt16    ataPBSemaphores; /* Reserved */
            SInt32    ataPBReserved4; /* Reserved */
} ataPBHdr;
```

Field descriptions

`ataPBLink` 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 the ATA Manager is called and should be ignored upon return. This field should not be changed until the requested operation has completed.

`ataPBQType` This field is the queue type byte for safety check. It should be initialized to 0.

`ataPBVers` This field contains the parameter block structure version number. Values 1 through 3 are currently supported. Any values greater than 3 or a value of 0 result in a `paramErr` error.

`ataPBReserved` This field is reserved for future use. To ensure future compatibility, all reserved fields should be set to 0.

`ataPBReserved2` This field is reserved for future use. To ensure future compatibility, all reserved fields should be set to 0.

`ataPBCallbackPtr`
This field contains the completion routine pointer to be called on completion of the request. When this field is set to zero, it indicates a synchronous I/O request; a nonzero value indicates an asynchronous I/O request. The routine to which this field points is called either 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) (ataPB *)
```

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

`ataPBResult` Completion status. This field is returned by the ATA Manager after the request is completed. The value in this field is invalid until the operation is complete. Refer to Table A-1 on page A-59 for a list of the possible error codes returned in this field.

`ataPBFunctionCode`
This field is the function selector for the ATA Manager. The functions are defined in Table 3-2 on page 3-30. An invalid code in this field results in an `ATAFuncNotSupported` error.

`ataPBPIOSpeed` This field specifies the I/O speed requirement for the ATA device. It is ignored in version 3.0 of the ATA Manager. The method for determining the I/O speed for version 3 of the ATA Manager is provided in the `ATA_SetDevConfig` function description.

For parameter block versions 1 and 2, this field specifies the I/O cycle timing requirement of the specified device. This field should contain the equivalent of word 51 of the identify drive data, as defined in the ATA-2 specification. Values 0 through 3 are supported by version 2 of the ATA Manager. See the ATA-2 specification for the definitions of the timing values. If a timing value higher than one supported is specified, the ATA Manager operates in the fastest timing mode supported. Until the timing value is determined by examining the identify drive data returned by the `ATA_Identify` function, the client should request operations using the slowest mode (PIO mode 0).

In ATA Manager version 1, the value in this field is always valid. That is, this timing value is used to complete the requested operation. With ATA Manager version 2, the value in this field is valid only if the `CurrentSpeed` bit is set to 0 in the `ataFlags` field. If the `CurrentSpeed` bit is set to 1, the manager uses the timing mode set previously by the `ATA_SetDevConfig` command for the device, or the default value, which is mode 0.

`ataPBFlags` This 16-bit field contains control settings that indicate special handling of the requested function. The control bits are defined in Table 3-1 on page 3-25.

`ataPBReserved3` This field is reserved for future use. To ensure future compatibility, all reserved fields should be set to 0.

`ataPBDeviceID` A number that uniquely identifies an ATA device. This field consists of the following structure:

```
typedef struct     /* Device ID structure */
{
UInt16 Reserved;   /* The upper word is reserved */
UInt8 devNum;      /* Consists of device ID */
UInt8 busNum       /* Bus ID number */
} deviceIdentification;
```

Version 3. 0 of the ATA Manager supports two ATA devices per bus. The devices are physically numbered 0 and 1, respectively. Earlier versions of the ATA Manager used an unsigned 16-bit integer to specify the device number. In version 3.0 of the ATA Manager the value of `devNum` is used to distinguish between two devices on the bus specified by `busNum`. In systems with only one ATA device, this value is always 0.

`ataPBTimeOut` This field specifies the transaction timeout value in milliseconds. A value of 0 disables the transaction timeout detection. If a timeout value is not set, the system may halt indefinately if the device fails to respond properly.

`ataPBClientPtr1` This pointer field is available for application use. It is not modified or used by the ATA Manager.

`ataPBClientPtr2` This pointer field is available for application use. It is not modified or used by the ATA Manager.

`ataPBState` This field is used by the ATA Manager to keep track of the current bus state. This field must contain 0 when calling the ATA Manager.

`ataPBSemaphores` This field is reserved. To ensure future compatibility, all reserved fields should be set to 0.

`ataPBReserved4` This field is reserved for future use. To ensure future compatibility, all reserved fields should be set to 0.

Table 3-1 describes the functions of the control bits in the `ataPBFlags` field.

Table 3-1 Control bits in the `ataPBFlags` field

Name	Bits	Definition
`none`	0–2	Reserved.
`RegUpdate`	3	When set to 1, this bit indicates that a set of device registers should be reported back on completion of the request. This bit is valid for the `ATA_ExecIO` function only. Refer to the description on page 3-31 for details. The following device registers are reported back: Sector count register Sector number register Cylinder register(s) SDH register

continued

Table 3-1 Control bits in the `ataPBFlags` field (continued)

Name	Bits	Definition
`ProtocolType`	4–5	These bits specify the type of command: 00 = standard ATA 01 = reserved 11 = ATAPI These bits indicate how the protocol should be handled for the command type. Setting the bits to ATAPI and providing a nonzero packet command pointer indicates that a packet command should be sent prior to any data transfers. For ATA command values of $A0 and $A1, this field should contain the ATAPI setting. For all other ATA commands, the field must contain the ATA setting.
—	6	Reserved.
`UseDMA`	7	When set to 1, this bit indicates the data transfer is to be via DMA. DMA transfers are valid only with version 3.0 or greater of the ATA Manager and on system hardware that supports DMA. DMA transfers to and from ATA devices use different command codes from PIO transfers. The state of this bit must correspond to the command code.
`SGType`	8–9	This 2-bit field specifies the type of scatter-gather list passed in. This field is valid only 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:

Table 3-1 Control bits in the ataPBFlags field (continued)

Name	Bits	Definition

		`typedef struct /* SG entry structure */`
		`{`
		`  uchar* ioBuffer; /* → Data buffer pointer */`
		`  ulong ioReqCount; /* → Byte count */`
		`} IOBlock;`

`QLockOnError`	10	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 is processed following this request. When an error occurs with this bit set to 1, any I/O request without the `Immediate` bit set is halted until an `ATA_QRelease` command is issued. A status code of $717 is returned for subsequent asynchronous I/O requests until the `I/O Queue Release` command is issued. This permits the client application to examine the state at the time of the error. However, use this bit with caution. When it is set to 1 and an error condition is not handled correctly, the system will hang.
`Immediate`	11	When this bit is set to 1, it indicates that the request must be executed as soon as possible and the status of the request must 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 received and is executed according to that order.
`ATAioDirection`	12–13	This bitfield 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 These bits do not need to specify the direction of the ATAPI command packet bytes.

continued

Table 3-1 Control bits in the ataPBFlags field (continued)

Name	Bits	Definition
ByteSwap	14	When set to 1, this bit indicates that every byte of data prior to transmission on write operations and on reception on read operations is to be swapped. When this bit is set to 0, it forces bytes to go out in the LSB-MSB format compatible with PC clones. Typically, this bit should be set to 0. Setting this bit has performance implications because the byte swap is performed by the software. Use this bit with caution. ATAPI command packet bytes are swapped when this bit is set to 1.
UseConfigSpeed	15	When set to 1, this bit indicates that the current I/O speed setting specified in the most recent call to the ATA_SetDevConfig command should be used to transfer data across the ATA interface. If a ATA_SetDevConfig command has not been issued since power on, then the default settings of PIO mode 0 and singleword DMA mode 0 are used.

Setting Data Transfer Timing

This section defines the mechanism used by version 3.0 of the ATA Manager to set up and adjust the system hardware and software for optimized data transfers from and to ATA devices.

Beginning with version 3.0 of the ATA Manager, all cycle timing for data transfer is accomplished through the ATA_SetDevConfig function, defined on page 3-51. The timing values in the ataPIOSpeedMode field (used by ATA Manager version 2.0 to set cycle timing) in the parameter block header are ignored, and PIO, singleword DMA, and multiword DMA data transfer times are specified separately in the ATA_SetDevConfig function parameter block. In addition, minimum cycle times are determined for PIO and multiword DMA transfers with the ATA_SetDevConfig function.

The ATA-2 specification requires that ATA devices report cycle-timing requirements and transfer mode information through the ATA identify device command. In order to synchronize the system ATA controller speed to the device speed, the identify device information must be interpreted by the ATA Manager. The ATA Manager receives the necessary information from the client in the ATA_SetDevConfig function. Five fields in the ATA_SetDevConfig parameter block are used in various combinations to specify the timing and transfer mode values for PIO, multiword DMA, and singleword DMA data transfers.

Setting Up PIO Data Transfers

To set up PIO data transfers, the ATA Manager takes the values specified in the `ataPIOSpeedMode` and `ataPIOCycleTime` fields of the `ATA_SetDevConfig` parameter block to create a cycle time that approximates the specified cycle time and maintains the appropriate device signal-timing requirements for the specified PIO transfer mode.

Setting Up Multiword and Singleword DMA Data Transfers

To set up multiword DMA data transfers, the ATA Manager takes the values in the `ataMultiDMASpeed` and `ataMultiCycleTime` fields of the `ATA_SetDevConfig` parameter block to create a multiword DMA cycle time in system hardware that maintains the timing required by the multiword DMA mode while not exceeding the indicated cycle time.

To set up singleword DMA data transfers, the ATA Manager takes the value specified in the `ataSingleDMASpeed` field of the `ATA_SetDevConfig` parameter block to create the appropriate cycle timing for the device. The ATA-2 specification has no recommended timing values for singleword DMA data transfer modes, only minimum cycle times.

When both the `ataSingleDMASpeed` and `ataMultiDMASpeed` fields in the `ATA_SetDevConfig` function parameter block are set to 0 and the `UseDMA` flag in the `ataPBFlags` field is set to true, the ATA Manager uses singleword DMA mode 0 timing for data transfers.

The `UseConfigSpeed` flag of the `ataPBFlags` field in the `ataPBHdr` parameter block header must be set for both the `ataExecIO` and `ATA_SetDevConfig` functions to utilize new timing configuration information. When the `UseConfigSpeed` flag is not set, new timing values are not calculated and saved during an `ATA_SetDevConfig` function call. When the `UseConfigSpeed` flag is not set and the `UseDMA` flag is specified, timing is set to singleword DMA mode 0. If the `UseConfigSpeed` flag is not set for an `ataExecIO` function, PIO mode 0 timing is used for commands and PIO data transfers.

Additional reference documentation related to Identify Device data transfer timing information for ATA devices can be found in the ANSI ATA-2 specification.

Functions

This section describes the ATA Manager functions that are used to manage and perform data transfers. Each function is requested through a parameter block specific to that service. A request for an ATA function is specified by a function code in the parameter block. The entry point for all the functions is the same.

ATA Manager function names and codes are shown in Table 3-2.

Table 3-2 ATA Manager functions

Function name	Code	Description
ATA_NOP	$00	No operation.
ATA_ExecIO	$01	Execute ATA I/O.
ATA_BusInquiry	$03	Bus inquiry.
ATA_QRelease	$04	I/O queue release.
ATA_Abort	$10	Terminate command.
ATA_ResetBus	$11	Reset ATA bus.
ATA_RegAccess	$12	ATA device register access.
ATA_Identify	$13	Get the drive identification data.
ATA_DrvrRegister	$85	Register the driver reference number.
ATA_FindRefNum	$86	Look up the driver reference number.
ATA_DrvrDeregister	$87	Deregister the driver reference number.
ATA_GetDevConfig	$8A	Get the device configuration.
ATA_SetDevConfig	$8B	Set the device configuration.
ATA_GetLocationIcon	$8C	Get device location icon and string.
ATA_MgrInquiry	$90	ATA Manager inquiry.

ATA_NOP

The ATA_NOP function performs no operation across the interface 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.

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

```
struct         ataNOP            /* ATA NOP structure */
{
   ataPBHdr                      /* ataPBHdr parameter block */
   UInt16      Reserved[24];
}  ataQRelease;
```

Field descriptions

ataPBHdr See the definition of the ataPBHdr structure on page 3-22.

`Reserved[24]` Reserved. All reserved fields should be set to 0.

RESULT CODES

See Table A-1 on page A-59 for possible result codes returned by the ATA Manager.

ATA_ExecIO

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

A prior call to the `ATA_SetDevConfig` function is recommended to obtain the optimal performance from the device. See page 3-51 for information about the `ATA_SetDevConfig` function.

The manager function code for the `ATA_ExecIO` function is $01.

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

```
typedef     struct            /* ATA_ExecIO structure */

{
   ataPBHdr                   /* ataPBHdr parameter block */
   SInt8    ataStatusReg;     /* ← Last device status register
                                 image */
   SInt8    ataErrorReg;      /* ← Last device error register
                                 image (valid if bit 0 of
                                 Status field is set) */
   SInt16   ataReserved;      /* Reserved */
   UInt32   BlindTxSize;      /* → Data transfer size */
   UInt8    *ioBuffer;        /* → Data buffer pointer */
   UInt32   ioReqCount;       /* ↔ Number of bytes to transfer */
   UInt32   ataActualTxCnt;   /* ← Number of bytes transferred */
   UInt32   ataReserved2;     /* Reserved */
   devicePB RegBlock;         /* → Device register images */
   ATAPICmdPacket *packetCDBPtr;  /* → ATAPI packet command
                                     block pointer */
   UInt16   ataReserved3[6];  /* Reserved */

} ataExecIO;
```

Field descriptions

`ataPBHdr` See the definition of the `ataPBHdr` parameter block on page 3-22.

`ataStatusReg` This field contains the last device status register image. See the ATA-2 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-2 specification for error register bit definitions.

`ataReserved` Reserved. All reserved fields are set to 0 for future compatibility.

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

The `BlindTxSize` field is used only for PIO transfers. It is ignored for DMA data transfers.

This field is ignored for ATAPI commands. For the ATAPI ptotocol, the dynamic byte transfer count is specified in the cylinder registers (big-endian format) at the beginning of each interrupt. See the `Cylinder` field in the definition of the `devicePB` structure for the `RegBlock` field.

`ioBuffer` This field contains either the host buffer address for the number of bytes specified in the `ioReqCount` field and the requested transfer length, or a pointer to a scatter-gather list and the number of scatter gather entries. If the `SGType` bits of the `ataPBFlags` field are set, an `IOBlk` structure contains the scatter-gather information. The `IOBlk` structure is defined as follows:

```
typedef        struct
{
      UInt8    *ioBuffer;  /* → Data buffer ptr */
      UInt32   ioReqCount; /* → Transfer length */
} IOBlk;
```

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

`ioReqCount` This field contains the number of bytes to transfer either from or to the buffer specified in `ioBuffer`. On returning, the `ioReqCount` field is updated to reflect data transfers (0 if successful; otherwise, the number of bytes that remained to be transferred prior to the error condition). When the `SGType` bits

of the `ataFlags` field are set, the `ioReqCount` field contains the number of scatter-gather entries in the list pointed to by the `ioBuffer` field.

`ioReqCount` This field contains the number of bytes to transfer either from or to the buffer specified in `ioBuffer`. On returning, this field is updated to reflect data transfers (0 if successful; otherwise, the number of bytes that remained to be transferred prior to the error condition). When the `SGType` bits of the `ataPBFlags` field are set, the `ioReqCount` field contains the number of scatter-gather entries in the list pointed to by the `ioBuffer` field.

`ataActualTxCnt` This field contains the total number of bytes transferred for this request.

`ataReserved2` This field is reserved. To ensure future compatibility, all reserved fields should be set to 0.

`RegBlock` This field contains the ATA device register image structure. 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 the ATA Manager. The ATA device register image structure is defined as follows:

```
typedef     struct     /* Device register images */
{   UInt8   Features;  /* → Features register
                             image */
    UInt8   Count;     /* ↔ Sector count */
    UInt8   Sector;    /* ↔ Sector start/finish */
    UInt8   Reserved;  /* Reserved */
    UInt16  Cylinder;  /* ↔ Cylinder big-endian
                             format */
    UInt8   SDH;       /* ↔ SDH register image */
    UInt8   Command;   /* → Command register image */
} devicePB;
```

For ATAPI commands, the cylinder image must contain the preferred PIO DRQ packet size, which is written out to the cylinder high/low registers at command phase.

`packetCDBPtr` This field contains the pointer to the `ATAPICmdPtr` packet structure for the ATAPI protocol. The ATAPI bit of the `ProtocolType` field in the `ataPBHdr` parameter block must be set for this field to be valid. Setting the ATAPI protocol bit also signals the ATA Manager to initiate the transaction without the DRDY bit set in the status register of the device. The `ATAPICmdPtr` structure is defined as follows:

```
typedef     struct     /* ATAPI command packet */;
{
    SInt16  packetSize;    /* Size of command, in
                              bytes (exclude size) */
```

```
        SInt16   command[8];    /* Actual ATAPI
                                   command packet */
} ATAPICmdPacket;
```

For ATA commands, this field should contain 0 in order to ensure compatibility in the future.

`ataReserved3[6]` These fields are reserved. To ensure future compatibility, all reserved fields should be set to 0.

RESULT CODES

See Table A-1 on page A-59 for possible result codes returned by the ATA Manager.

ATA_BusInquiry

The `ATA_BusInquiry` function returns information about a specific ATA bus for standard ATA and ATAPI interfaces and for the HBA environment. The support for HBA features in this function 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 follows:

```
typedef struct                  /* ATA bus inquiry structure */
{
  ataPBHdr                      /* ataPBHdr parameter block */
  UInt16    ataEngineCount;     /* ← TBD; 0 for now */
  UInt16    ataReserved;        /* Reserved */
  UInt32    ataDataTypes;       /* ← TBD; 0 for now */
  UInt16    ataIOpbSize;        /* ← Size of ATA I/O PB */
  UInt16    ataMaxIOpbSize;     /* ← TBD; 0 for now */
  UInt32    ataFeatureFlags;    /* ← TBD */
  UInt8     ataVersionNum;      /* ← HBA version number */
  UInt8     ataHBAInquiry;      /* ← TBD; 0 for now */
  UInt16    ataReserved2;       /* Reserved */
  UInt32    ataHBAPrivPtr;      /* ← Ptr to HBA private data */
  UInt32    ataHBAPrivSize;     /* ← Size of HBA private data */
  UInt32    ataAsyncFlags;      /* ← Capability for callback */
  UInt8     ataPIOModes;        /* ← PIO modes supported */
  UInt8     ataReserved3;       /* Reserved */
  UInt8     ataSingleDMAModes;  /* ← Singleword DMA mode */
  UInt8     ataMultiDMAModes;   /* ← Multiword DMA mode */
  UInt32    ataReserved4[8];    /* Reserved */
```

```
   SInt8      ataHBAVendor[16];   /* ← HBA Vendor ID */
   SInt8      ataContrlFamily[16];/* ← Family of ATA controller */
   SInt8      ataContrlType[16];  /* ← Controller model number */
   SInt8      ataXPTversion[4];   /* ← Version number of XPT */
   SInt8      ataReserved6[4];    /* Reserved */
   SInt8      ataHBAversion[4];   /* ← Version number of HBA */
   UInt8      ataHBAslotType;     /* ← Type of slot */
   UInt8      ataHBAslotNum;      /* ← Slot number of HBA */
   UInt16     ataReserved7;       /* Reserved */
   UInt32     ataReserved8;       /* Reserved */
} ataBusInquiry;
```

Field descriptions

`ataPBHdr` See the definition of the `ataPBHdr` structure on page 3-22.

`ataEngineCount` This field is currently set to 0.

`ataReserved` Reserved. All reserved fields are set to 0 to ensure future compatibility.

`ataDataTypes` Not supported by the current ATA architecture. Returns a bitmap of data types supported by this HBA. The data types are numbered from 0 through 30; 0 through 15 are reserved for Apple definition and 16 through 30 are available for vendor use. This field is currently not supported and returns 0.

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

`ataMaxIOpbSize` This 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. This field is not supported; it returns a value of 0.

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

`ataHBAInquiry` Reserved.

`ataHBAPrivPtr` This field contains a pointer to the private data area for the HBA. This field is not supported; it returns a value of 0.

`ataHBAPrivSize` This field contains the byte size of the private data area for the HDA. This field is not supported; it returns a value of 0.

`ataAsyncFlags` These flags indicate which types of asynchronous events the HBA is capable of generating. This field is not supported; it returns a value of 0.

`ataPIOModes` This bit-significant field specifies the PIO modes that the ATA bus supports. The least significant bit indicates support for PIO transfer mode 0. Refer to the ATA-2 specification for information on PIO mode timing.

`ataSingleDMAModes` This bit-significant field specifies the singleword DMA transfer modes that the ATA bus supports. The least significant bit indicates

support for singleword DMA transfer mode 0. Refer to the ATA-2 specification for information on DMA mode timing.

`ataMultiDMAModes`
This bit-significant field specifies the multiword DMA transfer modes that the ATA bus supports. The least significant bit indicates support for multiword DMA transfer mode 0. Refer to the ATA-2 specification for information on DMA mode timing.

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

`ataContrlFamily` Reserved.

`ataContrlType` This field identifies the specific type of ATA controller. This field is not supported; it returns a value of 0.

`ataXPTversion` Reserved.

`ataHBAversion` This field specifies the version of the HBA. This field is not supported; it returns a value of 0.

`ataHBAslotType` This field specifies the type of slot. This field is not supported; it returns a value of 0.

`ataHBAslotNum` This field specifies the slot number of the HBA. This field is not supported; it returns a value of 0.

RESULT CODES

See Table A-1 on page A-59 for possible result codes returned by the ATA Manager.

ATA_QRelease

The `ATA_QRelease` function releases the frozen I/O queue of the selected device.

When the ATA Manager detects an I/O error and the `QLockOnError` bit of the parameter block is set for the request, the ATA Manager freezes the queue for the selected device. No pending or new requests are processed or receive status until the queue is released through the `ATA_QRelease` function. Only those requests with the `Immediate` bit set in the `ataPBFlags` field of the `ataPBHdr` parameter block are processed. Consequently, for the ATA I/O queue release command to be processed, it must be issued with the `Immediate` bit set in the parameter block. An ATA I/O queue release command issued while the queue isn't frozen returns the `noErr` status.

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

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

```
struct         ataQRelease       /* ATA QRelease structure */
{
   ataPBHdr                      /* ataPBHdr parameter block */
   UInt16      Reserved[24];
}  ataQRelease;
```

Field descriptions

`ataPBHdr`	See the definition of the `ataPBHdr` structure on page 3-22.
`Reserved[24]`	Reserved. All reserved fields should be set to 0.

RESULT CODES

See Table A-1 on page A-59 for possible result codes returned by the ATA Manager.

ATA_Abort

The `ATA_Abort` function terminates a specified queued I/O request. This function applies to asynchronous I/O requests only. The `ATA_Abort` 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 found request 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. Cleanup includes parameter block deallocation and operating-system reporting.

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

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

```
typedef      struct         /* ATA abort structure */
{
  ataPBHdr                  /* ataPBHdr parameter block */
  ataPB*     AbortPB;       /* Address of the parameter block
                               of the function to be aborted */
  UInt16     Reserved[22];  /* Reserved */
} ataAbort;
```

Field descriptions

`ataPBHdr`	See the definition of the `ataPBHdr` parameter block on page 3-22.
`AbortPB`	This field contains the address of the I/O parameter block to be aborted.
`Reserved[22]`	This field is reserved. To ensure future compatibility, all reserved fields should be set to 0.

RESULT CODES

See Table A-1 on page A-59 for possible result codes returned by the ATA Manager.

ATA_ResetBus

The ATA_ResetBus function performs a soft reset operation to the selected ATA bus. The ATA interface doesn't provide a way to reset individual units on the bus. Consequently, all devices on the bus are reset.

IMPORTANT

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         /* ATA reset structure */
{
  ataPBHdr                 /* ataPBHdr parameter block */
  SInt8     Status;        /* ← Last ATA status register image */
  SInt8     Reserved2;     /* Reserved */
  UInt16    Reserved[23];  /* Reserved */
} ataResetBus;
```

Field descriptions

ataPBHdr	See the definition of the ataPBHdr parameter block on page 3-22.
Status	This field contains the last device status register image following the bus reset. See the ATA-2 specification for definitions of the status register bits.
Reserved[23]	This field is reserved. To ensure future compatibility, all reserved fields should be set to 0.

RESULT CODES

See Table A-1 on page A-59 for possible result codes returned by the ATA Manager.

ATA_RegAccess

The ATA_RegAccess function enables access to a particular device register of a selected device. This function is used for diagnostic and error recovery processes.

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

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

```
typedef        struct            /* Register access structure */
{
   struct      ataPBHdr;;        /* ataPBHdr parameter block */
   UInt16      RegSelect;        /* → Device register selector */
```

```
union{
        UInt8 byteRegValue;     /* ↔ Byte register value to
                                    read or to be written */
        UInt16 wordRegValue;    /* ↔ Word register value to
                                    read or to be written */
   } registerValue;
   UInt16      RegMask;         /* → Mask for registers(s) to
                                    update */
   devicePB    xi;              /* ↔ Register images */
   UInt8       altStatDevCntrReg;/* ↔ Alternate status(R) or
                                    device control(W) register
                                    image */
   UInt8       Reserved[3];     /* Reserved */
   UInt16      Reserved[16];    /* Reserved */
} ataRegAccess;
```

Field descriptions

`ataPBHdr` See the definition of the `ataPBHdr` parameter block on page 3-22.

`RegSelect` This field specifies which device registers to access. The selectors for the registers supported by the `ATA_RegAccess` function are listed in Table 3-3. If `RegSelect` is `0xFFFF`, then `RegMask` describes which register(s) are to be accessed as part of a multiregister access.

`RegValue` This field is either the source or destination of values for individual register accesses. For byte accesses, the upper half of the word is used. Word accesses (such as the data register) use the entire word. This field is the source or destination for the data register component of multiregister accesses.

`regMask` This field is valid only if the `RegSelect` field contains `0xFFFF`. It indicates what combination of the taskfile registers should be accessed. A bit set to 1 indicates either a read or a write to the register. A bit set to 0 performs no operation to the register. The mask bits corresponding to the selected registers are listed in Table 3-4 on page 3-40. Bit 0 is the least significant bit of the field.

xi This field contains register images for error/features, sector count, sector number, cylinder low, cylinder high, SDH, and status/command. Only those register images specified in the `regMask` field are valid. Refer to the description of the `devicePB` structure for the `RegBlock` field on page 3-33.

`altStatDevCntrReg` For multiregister writes, this field is the source for device control writes and the destination for alternate status reads. This field is

valid if the alternate status/device control register bit in the `RegMask` field is set to 1.

Table 3-3 ATA register selectors for `RegSelect` field

Selector name	Selector	Register description
`DataReg`	0	Data register (16-bit access only)
`ErrorReg`	1	Error register (R) or features register (W)
`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` `CmdReg`	7	Status register (R) or command register (W)
`AltStatus` `DevCntr`	14	Alternate status (R) or device control (W)
	0xFFFF	Multiregister access (ataPBVers 2.0 or greater)

The register mask selectors are defined in Table 3-4.

Table 3-4 Register mask selectors

Mask bit	Register description
0	Data register
1	Error register
2	Sector count register
3	Sector number register
4	Cylinder low register
5	Cylinder high register
6	`ataTFSDH` register
7	Status/command register
8–13	Reserved (set to 0)
14	Alternate status/device control register
15	Reserved (set to 0)

RESULT CODES

See Table A-1 on page A-59 for possible result codes returned by the ATA Manager.

ATA_Identify

The `ATA_Identify` function returns the device identification data from the selected device. The identification data contains information necessary to perform I/O to the device. Refer to the ATA-2 specification for the format and the information description 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                      /* ataPBHdr parameter block */
  SInt8     ataStatusReg;       /* ← Last ATA status image */
  sInt8     ataErrorReg;        /* ← Last ATA error image */
  SInt16    ataReserved;        /* Reserved */
  UInt32    BlindTxSize;        /* ← Set to 512 on return */
  UInt8     *DataBuf;           /* ↔ Buffer for the data */
  UInt32    ataRequestCount;    /* ← Indicates remaining
                                   byte count */
  UInt32    ataActualTxCnt;     /* ← Actual transfer count */
  UInt32    ataReserved2;       /* Reserved */
  devicePB  RegBlock;           /* ← taskfile image sent
                                   for the command */
  UInt16    Reserved3[8];       /* Reserved */
} ataIdentify;
```

Field descriptions

`ataPBHdr`	See the definition of the `ataPBHdr` parameter block on page 3-22.
`ataStatusReg`	Last ATA taskfile status register image.
`ataErrorReg`	Last ATA taskfile error register image. This field is only valid if the LSB (error bit) of the `ataStatusReg` field is set .
`BlindTxSize`	Size, in bytes, of the device identify data returned.
`DataBuf`	A pointer to the data buffer for the device identify data. The length of the buffer must be at least 512 bytes.
`ataRequestCount`	Number of remaining bytes to transfer.
`ataActualTxCnt`	Number of bytes transferred.
`RegBlock`	Taskfile image sent to the device.

RESULT CODES

See Table A-1 on page A-59 for possible result codes returned by the ATA Manager.

ATA_DrvrRegister

The `ATA_DrvrRegister` function registers the driver reference number passed in for the selected drive. The function doesn't check for the existence of another driver. Any active driver that controls one or more devices through the ATA Manager must register with the manager to ensure proper operation and notification of events. The first driver to register for the device gets the device. All subsequent registrations for the device are rejected.

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

The parameter block associated with `ataPBVers` of 1 is defined as follows:

```
typedef       struct             /* Driver registration
                                    structure for ataPBVers 1 */
{
   ataPBHdr                      /* ataPBHdr parameter block */
   SInt16     drvrRefNum;        /* → Driver reference number */
   UInt16     FlagReserved;      /* Reserved (should be 0)*/
   UInt16     deviceNextID;      /* Not used */
   SInt16     Reserved[21];      /* Reserved */
} ataDrvrRegister;
```

Field descriptions

`ataPBHdr` See the `ataPBHdr` parameter block definition on page 3-22.

`drvrRefNum` This field specifies the driver reference number to be registered. This value must be less than 0 to be valid.

`FlagReserved` Reserved.

`deviceNextID` Not used by this function.

`Reserved[21]` This field is reserved. To ensure future compatibility, all reserved fields should be set to 0.

The parameter block associated with `ataPBVers` of 2 or greater is defined below:

```
typedef       struct             /* Driver registration
                                    structure for ataPBVers 2
                                    or greater */
{
   ataPBHdr                      /* ataPBHdr parameter block */
   SInt16     drvrRefNum;        /* → Driver reference number */
   UInt16     drvrFlags;         /* → Driver flags, set to 0 */
   UInt16     deviceNextID;      /* Not used */
   SInt16     Reserved;          /* Reserved (should be 0) */
   ProcPtr    ataEHandlerPtr;    /* → Event handler routine
                                    pointer */
   SInt32     drvrContext;       /* → Value to pass in with
```

```
                                    event handler */
   UInt32     ataEventMask;      /* → Masks of various events
                                    for event handlers */
   SInt16     Reserved[14];      /* Reserved */
} ataDrvrRegister;
```

The version 2 parameter block also allows another type of registration: notify-all driver registration. The notify-all driver registration is identified by a value of -1 in the `ataPBDeviceID` field of the header and bit 0 of `drvrFlags` set to 0. The notify-all driver registration is used if notification of all device insertions is desired. Registered default drivers are called if no media driver is found on the media. Typically, an init driver registers as a notify-all driver. The single driver may register as a notify-all driver and then later register for one or more devices on the bus.

Note
All PCMCIA/ATA and notify-all device drivers must register using the parameter block version 2 and utilize the event-handling capability in order to insure proper operation. See the description of the `ataEHandlerPtr`, `drvrContent`, and `ataEventMask` fields for additional information related to event handling. ◆

Field descriptions

`ataPBHdr` See the `ataPBHdr` parameter block definition on page 3-22.

`drvrRefNum` This field specifies the driver reference number to be registered. This value must be less than 0 to be valid.

`drvrFlags` No bit definition has been defined for the field. This field shall be set to 0 in order to ensure compatibility in the future.

`deviceNextID` Not used by this function.

`Reserved` Reserved.

`ataEHandlerPtr` A pointer to an event handler routine for the driver. This routine is called whenever an event happens, and the mask bit for the particular event is set in the `ataEventMask` field.

The calling convention for the event handler is as follows:

```
pascal SInt16 (ataEHandlerPtr) (ATAEventRec*);
```

where `ATAEventRec` structure is defined as follows:

```
typedef     struct
{   UInt16  evenCode;   /* ATA event code */
    UInt16  phyDrvRef;  /* ID associated with
                           the event */
    SInt32  drvrContext;/* Context passed in by
                           the driver */
} ATAEventRec;
```

`drvrContext` A value to be passed in when the event handler is called. This value is loaded in the `ATAEventRec` structure before calling the event handler.

`ataEventMask` The mask defined in this field is used to indicate whether the event handler should be called or not, based on the event. The event handler is called only if the mask for the event has been set (1). If the mask is not set (0) for an event, the ATA Manager takes no action. The following masks have been defined:

Bits	Event mask
0x00	Null event.
0x01	Online event – a device has come online.
0x02	Offline event – a device has gone offline.
0x03	Device removed event – a device has been removed.
0x04	Reset event – a device has been reset.
0x05	Offline request event – a request to put the device offline has been detected.
0x06	Eject request event – a request to eject a device has been detected.
0x07	Configuration update event – the system configuration has changed (more devices).
0x08 – 0x1F	Reserved.

`Reserved[14]` This field is reserved. To ensure future compatibility, all reserved fields should be set to 0.

RESULT CODES

See Table A-1 on page A-59 for possible result codes returned by the ATA Manager.

ATA_FindRefNum

The `ATA_FindRefNum` function allows an application to determine whether a driver has been installed for a given device. You pass in a device ID, and the function returns the current driver reference number registered for the given device. A value of 0 indicates that no driver has been registered. The `deviceNextID` field contains the device ID of the next device in the list. The end of the list is indicated with the constant `kATAEndIterateDeviceID` with a value of `0xFF`.

To create a list of all drivers for the attached devices, pass in the constant `kATAStartIterateDeviceID` with a value of `0xFFFF` for the `ataPBDeviceID` field in the `ataPBHdr` structure. This causes `deviceNextID` field 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 `ataPBDeviceID` field in the `ataPBHdr` parameter block until the function returns `0xFF` 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 for `ataPBVers` version 1 is defined as follows:

```
typedef       struct            /* Driver registration for
                                      ataPBVers version 1 */
{
   ataPBHdr                     /* ataPBHdr parameter block */
   SInt16     drvrRefNum;       /* ← Contains the driver refNum */
   UInt16     FlagReserved;     /* Reserved */
   UInt16     deviceNextID;     /* ← Contains the next drive ID */
   SInt16     Reserved[21];     /* Reserved */
} ataDrvrRegister;
```

The parameter block associated with this function for `ataPBVer` version 2 is defined as follows:

```
typedef       struct              /* Driver registration for
                                      ataPBVers version 2
                                      or greater */
{
   ataPBHdr                       /* ataPBHdr parameter block */
   SInt16     drvrRefNum;         /* ← Driver reference number */
   UInt16     drvrFlags;          /* → Reserved, set to 0 */
   UInt16     deviceNextID;       /* Used to specify the next
                                      device ID */
   SInt16     Reserved;           /* Reserved (should be 0) */
   ProcPtr    ataEHandlerPtr;     /* ← Event handler routine
                                      pointer */
   SInt32     drvrContext;        /* ← Value to pass in with
                                      event handler */
   UInt32     ataEventMask;       /* ← Current setting of the
                                      mask of various events
                                      for event handler */
   SInt16     Reserved[14];       /* Reserved */
} ataDrvrRegister;
```

Field descriptions

`ataPBHdr`	See the `ataPBHdr` parameter block definition on page 3-22.
`drvrRefNum`	On return, this field contains the reference number for the device specified in the `ataPBDeviceID` field of the `ataPBHdr` data.
`drvrFlags`	This field is reserved. To ensure future compatibility, all reserved fields should be set to 0.
`deviceNextID`	On return, this field contains the device ID of the next device on the list.

`ataEHandlerPtr`	Currently registered event handler routine pointer for the selected device. This field is valid only for `ataPBVers` of 2 or greater.
`drvrContext`	Currently registered value to be passed along when the event handler is called. This field is valid only for `ataPBVers` of 2 or greater.
`ataEventMask`	Current event mask value for the selected device. This field is valid only for `ataPBVers` of 2 or greater.
`Reserved[`*nn*`]`	Reserved. To ensure future compatibility, all reserved fields should be set to 0.

RESULT CODES

See Table A-1 on page A-59 for possible result codes returned by the ATA Manager.

ATA_DrvrDeregister

The `ATA_DrvrDeregister` function deregisters the driver reference number passed in for the selected drive. After successful completion of this function, the driver reference number for the drive is set to 0, which indicates that no driver is in control of this device.

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

For notify-all driver deregistration, the `ataEHandlerPtr` field is used to match the entry (the `ataPBDeviceID` field is invalid for the notify-all driver registration/deregistration). If a driver is registered as both a notify-all and for a specific device, the driver must deregister for each separately.

All notify-all device drivers must deregister using the ATA Manager parameter block version 2.

The parameter block associated with this function for `ataPBVers` version 1 is defined as follows:

```
typedef       struct             /* Driver registration */
                                 /* structure for ataPBVers 1 */
{
   ataPBHdr                      /* ataPBHdr parameter block */
   SInt16     drvrRefNum;        /* Not used*/
   UInt16     FlagReserved;      /* Reserved */
   UInt16     deviceNextID;      /* Not used */
   SInt16     Reserved[21];      /* Reserved */
} ataDrvrRegister;
```

The parameter block associated with this function for `ataPBVer` version 2 is defined as follows:

```
typedef       struct             /* Driver registration
                                    structure for ataPBVers 2
                                  or greater */
{
   ataPBHdr                      /* See definition on page 3-22 */
   SInt16     drvrRefNum;        /* → Driver reference number */
   UInt16     drvrFlags;         /* → Driver flags, set to 0 */
   UInt16     deviceNextID;      /* Not used */
   SInt16     Reserved;          /* Reserved (should be 0) */
   ProcPtr    ataEHandlerPtr;    /* → Event handler routine
                                    pointer */
   SInt32     drvrContext;       /* → Value to pass in with
                                    event handler */
   UInt32     ataEventMask;      /* → Masks of various events
                                    for event handlers */
   SInt16     Reserved[14];      /* Reserved */
} ataDrvrRegister;
```

Field descriptions

`ataPBHdr`	See the `ataPBHdr` parameter block definition on page 3-22.
`drvrRefNum`	Not used for this function.
`drvrFlags`	No bits have been defined for this field. This field should be set to 0 in order to ensure compatibility in the future.
`deviceNextID`	Not used for this function.
`Reserved`	Reserved.
`ataEHandlerPtr`	A pointer to the driver event handler routine. This field is only used for notify-all driver deregistration. This field is not used for other driver deregistration. Since this field is used to identify the correct “notify-all” driver entry, this field must be valid for “notify-all” driver deregistration.
`drvrContext`	Not used for this function.
`ataEventMask`	Not used for this function.

RESULT CODES

See Table A-1 on page A-59 for possible result codes returned by the ATA Manager.

ATA_GetDevConfig

The `ATA_GetDevConfig` function allows an application to determine the configuration for a specified socket.

The manager function code for the `ATA_GetDevConfig` function is $8A.

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

```
typedef      struct
{
   ataPBHdr                     /* ataPBHdr parameter block *
   SInt32    ConfigSetting;     /* ↔ 32 bits of configuration
                                   information */
   UInt8     ataPIOSpeedMode;   /* ← Default PIO mode setting */
   UInt8     Reserved3;         /* Reserved for word alignment */
   UInt16    pcValid;           /* ← PCMCIA unique */
   UInt16    RWMultipleCount;   /* Reserved */
   UInt16    SectorsPerCylinder;/* Reserved */
   UInt16    Heads;             /* Reserved */
   UInt16    SectorsPerTrack;   /* Reserved */
   UInt16    socketNum;         /* ← Socket number */
   UInt8     socketType;        /* ← Type of socket */
   UInt8     deviceType;        /* ← Type of active device */
   UInt8     pcAccessMode;      /* ← Access mode of socket */
   UInt8     pcVcc;             /* ← Device voltage */
   UInt8     pcVpp1;            /* ← Vpp 1 voltage */
   UInt8     pcVpp2;            /* ← Vpp 2 voltage */
   UInt8     pcStatus;          /* ← Status register setting */
   UInt8     pcPin;             /* ← Pin register setting */
   UInt8     pcCopy;            /* ← Copy register setting */
   UInt8     pcConfigIndex;     /* ← Option register setting */
   UInt8     ataSingleDMASpeed; /* ← Singleword DMA
                                   timing class */
   UInt8     ataMultiDMASpeed;  /* ← Default Multiword DMA
                                   timing class */
   UInt16    ataPIOCycleTime;   /* ← Default cycle time for
                                   PIO mode */
   UInt16    ataMultiCycleTime; /* ← Default cycle time for
                                   multiword DMA mode */
   UInt16    Reserved[7];       /* Reserved*/
} ataGetDevConfig;
```

Field descriptions

`ataPBHdr` See the `ataPBHdr` parameter block definition on page 3-22.

`ConfigSetting` This 32-bit field contains various configuration information. The bits have the following definitions:

Bits 0–5: Reserved
Bit 6: ATAPI packet DRQ handling setting
0 = wait for an interrupt before sending the ATAPI command packet.
1 = wait for the assertion of DRQ in the status register before sending the ATAPI command packet. This is the default setting.
Bits 7–31: Reserved, set to 0

`ataPIOSpeedMode` This field indicates the value for the PIO mode currently used for commands and PIO data transfers. This value can be modified with the `ATA_SetDevConfig` function. In parameter block versions 1 and 2, this field is an integer. In parameter block versions 3 and greater, this field is bit significant, where the low-order bit indicates that PIO mode 0 is the current mode.

`pcValid` This 16-bit field applies to systems that support PCMCIA card services. It indicates which of the PCMCIA unique fields contain valid information. The following values are defined:
bit 0: When set, the value in the `pcAccessMode` field is valid.
bit 1: When set, the value in the `pcVcc` field is valid.
bit 2: When set, the value in the `pcVpp1` field is valid.
bit 3: When set, the value in the `pcVpp2` field is valid.
bit 4: When set, the value in the `pcStatus` field is valid.
bit 5: When set, the value in the `pcPin` field is valid.
bit 6: When set, the value in the `pcCopy` field is valid.
bit 7: When set, the value in the `pcConfigIndex` field is valid.
bits 14–8: Reserved (set to 0).
bit 15: Reserved.

`RWMultipleCount` This field is reserved for future expansion. To ensure future compatibility, all reserved fields should be set to 0.

`SectorsPerCylinder`
This field is reserved for future expansion. To ensure future compatibility, all reserved fields should be set to 0.

`Heads` This field is reserved for future expansion. To ensure future compatibility, all reserved fields should be set to 0.

`SectorsPerTrack` This field is reserved for future expansion. To ensure future compatibility, all reserved fields should be set to 0.

`socketNum` This field contains the socket number for the device used by the PCMCIA card services. The socket number is required to request card services. A value of `0xFF` indicates the device is not a card services client.

`socketType` This field specifies the type of socket. The values are defined as
00 = unknown socket type
01 = internal ATA bus
02 = media bay socket
03 = PCMCIA socket

`deviceType`	This field specifies the type of device. The possible values are defined as: 00 = unknown or no device present 01 = standard ATA device detected 02 = ATAPI device detected 03 = PCMCIA ATA device detected
`pcAccessMode`	This field specifies the current mode of the socket. This field is valid only when bit 0 of the `pcValid` field is set. The mode values are 0 = I/O mode 1 = memory mode
`pcVcc`	This field specifies the voltage on Vcc in tenths of a volt. The value in this field is valid only valid when bit 1 of the `pcValid` field is set.
`pcVpp1`	This field specifies the voltage of Vpp1 in tenths of a volt. The value in this field is valid only when bit 2 of the `pcValid` field is set.
`pcVpp2`	This field specifies the voltage of Vpp2 in tenths of a volt. The value in this field is valid only when bit 3 of the `pccValid` field is set.
`pcStatus`	This field specifies the current card register setting of a PCMCIA device. The value in this field is valid only when bit 4 of the `pccValid` field is set.
`pcPin`	This field specifies the current card pin register setting of a PCMCIA device. The value in this field is valid only when bit 5 of the `pccValid` field is set.
`pcCopy`	This field specifies the current setting of the card socket/copy register of a PCMCIA device. The value in this field is valid only when bit 6 of the `pccValid` field is set.
`pcConfigIndex`	This field specifies the current setting of the card option register of a PCMCIA device. The value in this field is valid only when bit 7 of the `pccValid` field is set.
`ataSingleDMASpeed`	This bit-significant field indicates which singleword DMA mode, if any, is currently configured for use with DMA transfers. The intitial default value is singleword DMA mode 0. The DMA transfer mode may be modified with the `ATA_SetDevConfig` function. This field is valid only for ATA Manager 3.0 or greater.
`ataMultiDMASpeed`	This bit-significant field indicates which multiword DMA mode, if any, is currently configured for use with DMA transfers. The intitial default value of this field is 0, which indicates that multiword DMA not selected. The DMA transfer mode may be modified by the `ATA_SetDevConfig` function. This field is valid only for ATA Manager 3.0 or greater.
`ataPIOCycleTime`	This word field specifies the minimum cycle time, in microseconds, of mode 3 or greater PIO transfers. For additional information about the contents of this field, see the

ataPIOCycleTime field in the ATA_SetDevConfig description beginning on page 3-51.

The actual cycle time may be higher than this value if the system hardware is unable to create the requested cycle time while maintaining signal timing for the PIO mode in use. This field is valid only for ATA Manager 3.0 or greater.

ataMultiCycleTime

This word field specifies the minimum cycle time, in microseconds, of mode 1 or higher multiword DMA data transfers. For additional information about the contents of this field, see the ataMultiCycleTime field in the ATA_SetDevConfig function description, next.

The actual cycle time may be higher than this value if the system hardware is unable to create the requested cycle time while maintaining signal timing for the multiword DMA mode in use. This field is valid only for ATA Manager 3.0 or greater.

RESULT CODES

See Table A-1 on page A-59 for possible result codes returned by the ATA Manager.

ATA_SetDevConfig

The ATA_SetDevConfig function allows an application to set the configuration parameters of a specified socket. Some of the fields are not appropriate for a particular socket type — for example, setting the voltage for the internal device. Part of the device configuration includes setting up the parameters for I/O transfer mode and timing. The section "Setting Data Transfer Timing" beginning on page 3-28 includes a discussion of how to use the ATA Manager to set up the software for data transfers, including DMA data transfers.

The manager function code for the ATA_SetDevConfig function is $8B.

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

```
typedef      struct
{
   ataPBHdr                      /* ataPBHdr parameter block */
   char      ConfigSetting;      /* → 32 bits of configuration
                                    information */
   ushort    ataPIOSpeedMode;    /* → Default PIO mode setting*/
   ushort    Reserved3;          /* Reserved for word alignment*/
   ulong     pcValid;            /* → PCMCIA unique */
   ulong     RWMultipleCount;    /* Reserved */
   ulong     SectorsPerCylinder; /* Reserved */
   ulong     Heads;              /* Reserved */
```

```
   ulong      SectorsPerTrack;    /* Reserved */
   ulong      Reserved4[2];       /* Reserved */
/* PCMCIA-unique fields are indicated with a pc prefix */
   ushort     pcAccessMode;       /* → Access mode of socket */
   ushort     pcVcc;              /* → Device voltage */
   ushort     pcVpp1;             /* → Vpp 1 voltage */
   ushort     pcVpp2;             /* → Vpp 2 voltage */
   ushort     pcStatus;           /* → Status register setting */
   ushort     pcPin;              /* → Pin register setting */
   ushort     pcCopy;             /* → Copy register setting */
   ushort     pcConfigIndex;      /* → Option register setting */
/* The following fields are valid for parameter block version
   3.0 or greater (ataPBVers 3 or greater) */
   ushort     ataSingleDMASpeed;  /* → Singleword DMA
                                     timing class */
   ushort     ataMultiDMASpeed;   /* → Multiple word DMA
                                     timing class */
   ulong      ataPIOCycleTime;    /* → Cycle time for PIO mode */
   ulong      ataMultiCycleTime;  /* → Cycle time for multiword
                                     DMA mode */
   ulong      Reserved[7];        /* Reserved*/
} ATA_SetDevConfig;
```

Field descriptions

`ataPBHdr` See the `ataPBHdr` parameter block definition on page 3-22.

`ConfigSetting` This 32-bit field controls various configuration settings. The bits have the following definitions:

Bits 0–5: Reserved, set to 0

Bit 6: ATAPI packet DRQ handling setting
0 = wait for an interrupt before sending the ATAPI command packet.
1 = wait for the assertion of DRQ in the status register before sending the ATAPI command packet. This is the default setting.

Bits 7–31: Reserved, set to 0

`ataPIOSpeedMode` This field contains the PIO mode to be used for commands and PIO data transfers for parameter blocks prior to version 3. For parameter block version 3 or greater, the value is bit significant, with the low-order bit signifying PIO mode 0. Be sure to note the difference in bit positions between this field and the corresponding “advanced PIO modes” field of the ATA-2 identify device information.

`pcValid` This 16-bit field applies to systems that support PCMCIA card services. The bits indicate which fields in the parameter block contain valid settings for PCMCIA . The following values are defined:

bit 0: When set, the value in the `pcAccessMode` field is valid.
bit 1: When set, the value in the `pcVcc` field is valid.
bit 2: When set, the value in the `pcVpp1` field is valid.
bit 3: When set, the value in the `pcVpp2` field is valid.
bit 4: When set, the value in the `pcStatus` field is valid.
bit 5: When set, the value in the `pcPin` field is valid.
bit 6: When set, the value in the `pcCopy` field is valid.
bit 7: When set, the value in the `pcConfigIndex` field is valid.
bits 8–14: Reserved (set to 0).
bit 15: Reserved.

`PWMultipleCount` This field is reserved for future expansion. To ensure future compatibility, all reserved fields should be set to 0.

`SectorsPerCylinder`
This field is reserved for future expansion. To ensure future compatibility, all reserved fields should be set to 0.

`Heads` This field is reserved for future expansion. To ensure future compatibility, all reserved fields should be set to 0.

`SectorsPerTrack` This field is reserved for future expansion. To ensure future compatibility, all reserved fields should be set to 0.

`Reserved4[2]` This field is reserved.

`pcAccessMode` This field specifies the mode of the socket. This field is valid only when bit 0 of the `pcValid` filed is set. The mode values are:
0 = I/O mode
1 = memory mode

`pcVcc` This field specifies the new voltage setting for Vcc in tenths of a volt. The value in this field is valid only when bit 1 of the `pcValid` field is set.

`pcVpp1` This field specifies the new voltage setting for Vpp1 in tenths of a volt. The value in this field is valid only when bit 2 of the `pcValid` field is set.

`pcVpp2` This field specifies the new voltage setting for Vpp2 in tenths of a volt. The value in this field is valid only when bit 3 of the `pccValid` field is set.

`pcStatus` This field specifies the new card register setting for a PCMCIA device. The value in this field is valid only when bit 4 of the `pccValid` field is set.

`pcPin` This field specifies the new card pin register setting for a PCMCIA device. The value in this field is valid only when bit 5 of the `pccValid` field is set.

`pcCopy` This field specifies the new card socket/copy register setting for a PCMCIA device. The value in this field is valid only when bit 6 of the `pccValid` field is set.

`pcConfigIndex` This field specifies the new card option register setting for a PCMCIA device. The value in this field is valid only when bit 7 of the `pccValid` field is set.

`ataSingleDMASpeed` This bit-significant field specifies the singleword DMA mode for DMA data transfers. It corresponds to the high-order byte of

word 62 of the identify device data described in the ATA-2 specification. If word 62 is not supported by the device, then it reflects word 52 converted to bit significance. The ATA software supports word 62 modes 0 through 2, as defined in the ATA-2 specification. If the specified timing mode is higher than the values supported by the software, then the highest possible mode is selected for transfers.

The ATA Manager selects the transfer rate that satisfies the requirements of the mode and the system DMA hardware. The rate and mode are used on subsequent DMA transfers until changed by another `ATA_SetDevConfig` function call. The default singleword DMA mode is mode 0. This field is valid for parameter block version 3 or greater.

For additional information related to setting the I/O data transfer speed, see "Setting Data Transfer Timing" beginning on page 3-28.

`ataMultiDMASpeed` This bit-significant field specifies the multiword DMA cycle mode for DMA data transfers. It corresponds to the high-order byte of word 63 of the identify device data described in the ATA-2 specification. If word 63 is not supported by the device, then this value should be 0 which indicates that multiword DMA should not be attempted. The ATA software supports word 63 modes 0 through 2, as defined in the ATA-2 specification. If the specified timing mode is higher than the values supported by the software, then the highest supported mode is selected for DMA transfers. This field is used in conjunction with the value set in the `ataMultiCycleTime` field. The ATA Manager selects the transfer rate that satisfies the requirements of the mode specified in the `ataMultiCycleTime` field and the system DMA hardware. The rate and mode are used on subsequent DMA transfers until changed by another `ATA_SetDevConfig` function call. The default setting for DMA mode is singleword DMA mode 0. This field is valid for parameter block version 3 or greater.

For additional information related to setting the I/O data transfer speed, see "Setting Data Transfer Timing" beginning on page 3-28.

`ataPIOCycleTime` This word field is used in conjunction with the `ataPBIOSpeed` field of the `ataPBHdr` structure to specify the cycle time for command and PIO data transfers. The value in this field represents word 68 of the identify device information, as defined in the ATA-2 specification. If this value is not 0, the ATA Manager selects the closest approximation of the cycle time supported by the system hardware that does not exceed the value and still meets the timing requirements of the selected mode.

If this value is 0, the ATA Manager uses the minimum cycle times from the ATA-2 specification for the mode. The resulting cycle timing represents the maximum timing for PIO mode 2, because that is the highest mode supported without reporting word 68 of the identify device information.

`ataMultiCycleTime` This word field is used in conjunction with the `ataMultiDMASpeed` field to specify the cycle time for multiword DMA data transfers. The value represents the same value reported in word 65 or word 66 of the identify device information, as specified in the ATA-2 specification. If the value specified in this field is not 0, the ATA Manager selects the closest cycle time supported by the system hardware that does not exceed the value and meets the other timing requirements of the mode specified in the `ataMultiDMASpeed` field.

If the value is 0, the ATA Manager uses the minimum cycle times specified in the ATA-2 specification for the selected mode. The resulting cycle timing represents the minimum timing for multiword DMA mode 0, because that is the highest mode supported without reporting word 65 or word 66 of the identify device information.

RESULT CODES

See Table A-1 on page A-59 for possible result codes returned by the ATA Manager.

ATA_GetLocationIcon

The `ATA_GetLocationIcon` function returns a pointer to the structure defining the location icon data for the selected device. The structure contains the icon data and an icon string for the device.

The manager function code for the `ATA_GetLocationIcon` function is $8C.

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

```
typedef      struct
{
   ataPBHdr              /* ataPBHdr parameter block */
   ulong     iconData;   /* Pointer to icon data and
                            the size of the data */
} ATA_GetLocationIcon;
```

Field descriptions

`ataPBHdr` See the `ataPBHdr` parameter block definition on page 3-22.

`iconData` This field contains two fields, a pointer to a structure that contains the icon data and the size, in bytes, of the icon data. The structure that contains the actual icon data is defined as follows:

```
struct DriverLocationIcon
{
ushort  locationIcon[256];
char    locationString;
} DriverLocationIcon;
```

The `locationIcon` field is the device icon data (fixed 256 bytes). The `locationString` field is string in C string format (null terminated).

RESULT CODES

See Table A-1 on page A-59 for possible result codes returned by the ATA Manager.

ATA_MgrInquiry

The `ATA_MgrInquiry` function gets information, such as the version number, about the ATA Manager.

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

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

```
typedef         struct          /* ATA inquiry structure */
{
   ataPBHdr                     /* ataPBHdr parameter block */
   NumVersion   MgrVersion;     /* Version of ATA Manager */
   UInt8        MGRPBVers;      /* ← Manager PB version
                                    number supported */
   UInt8        Reserved1;      /* Reserved */
   UInt16       ataBusCnt;      /* ← Number of ATA buses in
                                    system */
   UInt16       ataDevCnt;      /* ← Number of ATA devices
                                    detected */
   UInt8        ataPIOMaxMode;  /* ← Maximum PIO speed mode */
   UInt8        Reserved2;      /* Reserved */
   UInt16       Reserved3;      /* Reserved */
   UInt8        ataSingleDMAModes; /* ← Singleword DMA modes
                                    supported */
   UInt8        ataMultiDMAModes;/* ← Multiword DMA modes
                                    supported */
   UInt16       Reserved[16];   /* Reserved */
} ataMgrInquiry;
```

Field descriptions

`ataPBHdr` See the definition of the `ataPBHdr` parameter block on page 3-22.

`MgrVersion` On return, this field contains the version number of the ATA Manager present in the system.

`MGRPBVers` This field contains the number corresponding to the latest version of the parameter block supported. A client may use any `ataPBHdr` parameter block definition up to this version.

`Reserved` Reserved. All reserved fields are set to 0 for future compatibility.

`ataBusCnt` On return, this field contains the total number of ATA buses in the system. This field contains a 0 if the ATA Manager has not been initialized. Not all ATA buses reported may be active. However, clients should allocate adequate data storage to handle up to the reported number of buses and/or channels.

`ataDevCnt` On 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 contains a 0 if the ATA Manager has not been initialized. With media bay and PCMCIA sockets, the value reported may change depending on the current configuration.

`ataPIOMaxMode` This field specifies the maximum PIO speed mode that the ATA Manager supports. Refer to the ATA-2 specification for information on mode timing. For additional information about the individual bus capabilities, see the description of the `ATA_BusInquiry` function on page 3-34.

`ataSingleDMAModes`
This bit-significant field specifies the maximum DMA mode that the ATA Manager can support. The least significant bit indicates support for singleword DMA transfer mode 0. Refer to the ATA-2 specification for information on DMA mode timing.

`ataMultiDMAModes`
This bit-significant field specifies the multiword DMA transfer modes that the manager can support. The least significant bit indicates support for multiword DMA transfer mode 0. Refer to ATA-2 specification for information on DMA mode timing.

`Reserved[16]` This field is reserved. To ensure future compatibility, all reserved fields should be set to 0.

RESULT CODES

See Table A-1 on page A-59 for possible result codes returned by the ATA Manager.

Result Code Summary

A summary of the ATA result codes is provided in Table A-1. ATA parameter block versions 2 and greater have a different numbering scheme from that of version 1. The error code number values for parameter block version 1 are contained in parentheses.

Table A-1 ATA Manager result codes

Error code	Error code (version 1)	Name	Description
0	0	`noErr`	Successful completion; no error detected
–50		`paramErr`	Invalid parameter specified
–56		`nsDrvErr`	No such drive installed
–9325		`ATANoDriverErr`	No driver found on media
–9226		`ATANoDDMErr`	No DDM found on media
–9327		`ATAMemoryErr`	Memory allocation error
–9328		`ATAInvalidDrvNum`	Invalid driver number from event
–9337		`ATAMgrConsistencyErr`	Manager detected internal inconsistency
–9338		`ATAXferModeErr`	I/O transfer mode not supported
–9339		`ATAXferParamErr`	I/O transfer parameters inconsistent
–9340		`ATASDFailErr`	Shutdown failure
–9341		`ATAMgrMemoryErr`	Manager memory allocation error
–9342	(–1800)	`CantHandleEvent`	Particular event could not be handled
–9343	(–1799)	`DriverLocked`	Current driver must be removed before adding another
–9344	(–1818)	`AT_NoAddrErr`	Invalid taskfile base address
–9345	(–1817)	`ATABusErr`	Bus error detected on I/O
–9346	(–1816)	`ATAInternalErr`	Card services returned an error
–9347	(–1815)	`ATANoClientErr`	No client present to handle event
–9348	(–1814)	`ATAPIExCntErr`	Warning: overrun/underrun condition detected (data valid)
–9349	(–1813)	`ATAPIPhaseErr`	Unexpected phase detected
–9350	(–1812)	`ATAAbortedDueToRst`	The I/O queue entry aborted due to a bus reset

continued

Table A-1 ATA Manager result codes (continued)

Error code	Error code (version 1)	Name	Description
–9351	(–1811)	`ATAUnableToAbort`	Request to abort couldn't be honored
–9352	(–1810)	`ATAReqAborted`	The request was aborted
–9353	(–1809)	`ATAQLocked`	I/O queue locked—cannot proceed
–9354	(–1808)	`ATAUnknownState`	Device in unknown state
–9355	(–1807)	`ATAReqInProg`	I/O channel in use—cannot proceed
–9356	(–1806)	`ATATransTimeOut`	Timeout: transaction timeout detected
–9357	(–1805)	`ATABusy`	Selected device is busy; device isn't ready to go to next phase yet
–9358	(–1804)	`ATAFuncNotSupported`	An unknown manager function code specified
–9359	(–1803)	`ATAPBInvalid`	Invalid device base address detected (=0)
–9360	(–1802)	`ATAMgrNotInitialized`	ATA Manager not initialized
–9371		`ATAEjectDrvErr`	Could not eject the drive
–9372		`ATADevUnSupported`	Device type not supported
–9373		`ATABufFail`	Device buffer test failed
–9374		`ATAInitFail`	ATA Manager initialization failure
–9375		`NoATAMgr`	No ATA Manager installed in the system (`MgrInquiry` failure)
–9376		`DRVRCantAllocate`	Global memory allocation error
–9396	(–1780)	`AT_AbortErr`	Command-aborted bit set in error register
–9397	(–1781)	`AT_RecalErr`	Recalibrate failure detected by device
–9398	(–1782)	`AT_WrFltErr`	Write fault bit set in status register
–9399	(–1783)	`AT_SeekErr`	Seek complete bit not set on completion
–9400	(–1784)	`AT_UncDataErr`	Uncorrected data bit set in error register
–9401	(–1785)	`AT_CorDataErr`	Data-corrected bit set in status register
–9402	(–1786)	`AT_BadBlkErr`	Bad block bit set in error register
–9403	(–1787)	`AT_DMarkErr`	Data mark not found bit set in error register
–9404	(–1788)	`AT_IDNFErr`	ID not found bit set in error register
–9405	(–1791)	`AT_NRdyErr`	Drive ready condition not detected

Index

A, B

ATA (IDE) hard disk, compared with SCSI drives 3
ATA (IDE) software
  ATA Manager 2, 4
  device driver 2
  hard disk device driver 3
`ATA_Abort` function 37
`ATA_BusInquiry` function 34
`ATA_DrvrDeregister` function 46
`ATA_DrvrRegister` function 42
`ATA_ExecIO` function 31
`ATA_FindRefNum` function 44
`ATA_GetDevConfig` function 48
`ATA_GetLocationIcon` function 55
`ATA_Identify` function 41
`ATA_MgrInquiry` function 56
`ATA_NOP` function 30
`ATA_QRelease` function 36
`ATA_RegAccess` function 38
`ATA_ResetBus` function 38
`ATA_SetDevConfig` function 51
ATA-2 specification 2
ATA disk driver 3, 5 to 19
  `close` routine 6
  control functions 8 to 19
  `control` routine 8
  Device Manager routines 6 to 8
  `drive info` function 15
  `driver gestalt` function 16
  `driverGestalt` parameter block 16
  driver name 4
  driver reference number 4
  `eject` function 10
  `format` function 9
  `get partition mount status` function 18
  `get partition write protect status` function 18
  `get power mode` function 18
  `get startup partition` function 17
  making calls to 4
  `open` routine 6
  `prime` routine 7
  `return drive characteristics` function 11
  `return media icon` function 10
  `set power mode` function 14
  `status` routine 7
  `verify` function 9
ATA Manager 2, 22 to ??
  making calls to 22
  parameter block 4
  purpose of 3, 4
ATA Manager functions
  `ATA_Abort` 37
  `ATA_BusInquiry` 34
  `ATA_DrvrDeregister` 46
  `ATA_DrvrRegister` 42
  `ATA_ExecIO` 31
  `ATA_FindRefNum` 44
  `ATA_GetDevConfig` 48
  `ATA_GetLocationIcon` 55
  `ATA_Identify` 41
  `ATA_MgrInquiry` 56
  `ATA_NOP` 30
  `ATA_QRelease` 36
  `ATA_RegAccess` 38
  `ATA_ResetBus` 38
  `ATA_SetDevConfig` 51
ATA parameter block header 22
`ataPBHdr` structure 22 to 28
`.ATDISK` driver name 4

C

`clear partition mounting` function 13
`clear partition write protect` function 13
`close` routine 6
`control` routine 8

D

data transfer timing 28
device ID list 44
DMA 26, 28, 50
DMA data transfers 51
DMA transfer mode 50
`drive info` function 15
`driver gestalt` function 16
`driverGestalt` parameter block 16
driver list for attached devices 44
driver reference number 44

E

`eject` function 10
`enabale partition write protect` function 12
`enable partition mounting` function 12
`enable startup partition` function 11

F

`format` function 9

G - L

`get partition mount status` function 18
`get partition write protect status` function 18
`get power mode` function 18
`get startup partition` function 17

M, N

`mount volume` function 14
multiword DMA 29, 50

O

`open` routine 6

P, Q

PCMCIA device configuration fields 49
`prime` routine 7

R

`return drive characteristics` function 11
`return drive icon` function 10
`return media icon` function 10

S, T, U

`set power mode` function 14
singleword DMA 29, 50
`status` routine 7

V - Z

`verify` function 9

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Line art was created using Adobe™ Illustrator and Adobe Photoshop.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier.

WRITER
Steve Schwander

COPY EDITOR
John Hammett, Beverly McGuire

PRODUCTION EDITOR
JoAnne Smith

ILLUSTRATOR
Sandee Karr

PRODUCTION EDITOR
Alex Solinski

Special thanks to Rhoads Hollowell, Steve Parsons, Rich Schnell, and Kevin Snow