Developer Note

Macintosh PowerBook Processor Card Upgrade Kit

Macintosh PowerBook Processor Card Upgrade Kit
with PowerPC 603e for Macintosh PowerBook 500 Series

Developer Press

Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

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

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

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

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

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

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

Contents

Figures and Tables

About This Developer Note

This developer note describes the Macintosh PowerBook Processor Card Upgrade Kit with PowerPC 603e for the PowerBook 500 series computers and the system software that accompanies it.

This developer note is intended to help hardware and software developers design products that are compatible with the Macintosh products described in the note. If you are not already familiar with Macintosh computers or if you would simply like more technical information, you may wish to read the supplementary reference documents described in this preface.

This note is published in two forms: an online version included with the Apple Developer CD and a paper version published by APDA. For information about APDA, see "Supplemental Reference Documents."

Contents of This Note

The information in this developer note is arranged in four chapters.

- Chapter 1, "Introduction," introduces the Macintosh PowerBook Processor Card Upgrade Kit and describes its new features.
- Chapter 2, "Architecture," describes the internal logic of the processor upgrade card, including the main ICs that appear in the block diagram.
- Chapter 3, "Software Features," describes the new features of the ROM and system software for the processor upgrade card.
- Chapter 4, "Large Volume Support," describes the modifications that enable the file system to support volumes larger than 4 GB.

This developer note also includes a glossary and an index.

Supplemental Reference Documents

The following documents provide information that complements or extends the information in this developer note.

Apple Publications

Developers should have copies of the appropriate Apple reference books, including the relevant volumes of *Inside Macintosh; Guide to the Macintosh Family Hardware,* second edition; and *Designing Cards and Drivers for the Macintosh Family,* third edition. These Apple books are available in technical bookstores and through APDA.

For information about the Power Manager, developers should have a copy of *Inside Macintosh: Devices.* For information about the control strip, developers should have the Reference Library volume of the Developer CD Series, which contains Macintosh Technical Note OS 06, *Control Strip Modules*.

For information about native drivers and the Driver Services Library, developers should have a copy of *Designing PCI Cards and Drivers for Power Macintosh Computers.*

For information about the Macintosh PowerBook 500 series computers, developers should also have a copy of *Macintosh Developer Notes,* number 9. The developer notes are available on the Developer CD Series and through APDA.

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

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

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

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

Other Publications

For information about programming the PowerPC™ microprocessors, developers should have copies of Motorola's *PowerPC 601 RISC Microprocessor User's Manual* and *PowerPC 603 Microprocessor Implementation Definition Book IV*.

Conventions and Abbreviations

This developer note uses the following typographical conventions and abbreviations.

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 of interest but is not essential for an understanding of the text. ◆

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

▲ **WARNING**
A note like this directs your attention to something that could cause injury to staff, damage to equipment, or loss of data. ▲

Standard Abbreviations

Standard units of measure used in this note include

GB	gigabytes	MB	megabytes
k	1024	MHz	megahertz
KB	kilobytes	ns	nanoseconds
M	1,048,576		

Other abbreviations used in this note include

$*n*	hexadecimal value *n*
ADB	Apple Desktop Bus
API	application program interface
CD	compact disc
CSC	color screen controller
DAA	data access adapter (a telephone line interface)

DAC	digital-to-analog converter
DRAM	dynamic RAM
FPSCR	floating-point status and control register
HFS	hierarchical file system
IC	integrated circuit
I/O	input/output
MMU	memory management unit
PDS	processor-direct slot
POWER	performance optimized with enhanced RISC
RAM	random-access memory
RISC	reduced instruction set computing
ROM	read-only memory
SCC	Serial Communications Controller
SCSI	Small Computer System Interface
SPR	special-purpose register
SWIM	Super Woz Integrated Machine (custom IC that controls the floppy disk interface)

CHAPTER 1

Introduction

The Macintosh PowerBook Processor Card Upgrade Kit replaces the secondary logic card in the Macintosh PowerBook 500 series computers and upgrades the microprocessor to a PowerPC™ 603. The processor upgrade card supports all the hardware features of the Macintosh PowerBook 500 series computer in which it is installed.

Features

Here is a summary of the major features of the Macintosh PowerBook Processor Card Upgrade Kit. Each feature is described more fully later in this developer note.

- **microprocessor**: The processor upgrade card has a PowerPC 603 microprocessor running at a clock frequency of 100 MHz.
- **RAM:** The processor upgrade card includes 8 MB of low-power, self-refreshing dynamic RAM (DRAM).
- **RAM expansion:** The processor upgrade card accepts a RAM expansion card with up to 56 MB, for a total of 64 MB of RAM. The RAM expansion card is the same as the one in an unmodified Macintosh PowerBook 500 series computer.
- **SCSI target mode:** The processor upgrade card is compatible with SCSI target mode (formerly called SCSI disk mode), which allows the user to read and store data on the computer's internal hard disk from another Macintosh computer.

▲ **WARNING**
Installation of a Macintosh PowerBook Processor Card Upgrade Kit must be done by an experienced technician. Care is required to avoid damage to the pins on the secondary logic board connector. ▲

Compatibility Issues

The Macintosh PowerBook Processor Card Upgrade Kit makes many significant changes in the Macintosh PowerBook computer in which it is installed. This section highlights key areas you should investigate in order to ensure that your hardware and software work properly with the upgraded PowerBook models. These topics are covered in more detail in subsequent sections.

POWER-Clean Code

The term *POWER-clean* refers to code that is free of the POWER instructions that would prevent it from running correctly on a PowerPC 603 or PowerPC 604 microprocessor. Applications for computers that have the processor upgrade card with the PowerPC 603 should be free of those instructions.

The instruction set of the PowerPC 601 microprocessor includes some of the same instructions as those found in the instruction set of the POWER processor, and some compilers used to generate native code for the first generation of Power Macintosh models generated some of those POWER-only instructions. However, the PowerPC 603 microprocessor used in the processor upgrade card does not support the POWER-only instructions. When you compile applications for Power Macintosh computers, you should turn off the option that allows the compiler to generate POWER-only instructions.

Emulation for Compatibility

The software for the processor upgrade card includes emulation of the POWER-only instructions of the PowerPC 601. Although the term *POWER emulation* is often used, a more appropriate name for this feature is *PowerPC 601 compatibility.* Rather than supporting the entire POWER architecture, the goal is to support those features of the POWER architecture that are available to programs running in user mode on the PowerPC 601–based Power Macintosh computers.

Because the emulation of the POWER-only instructions degrades performance, Apple Computer encourages developers to revise any applications that use those instructions to conform with the PowerPC architecture. Even though emulation works, performance is degraded; POWER-clean code is better.

Code Fragments and Cache Coherency

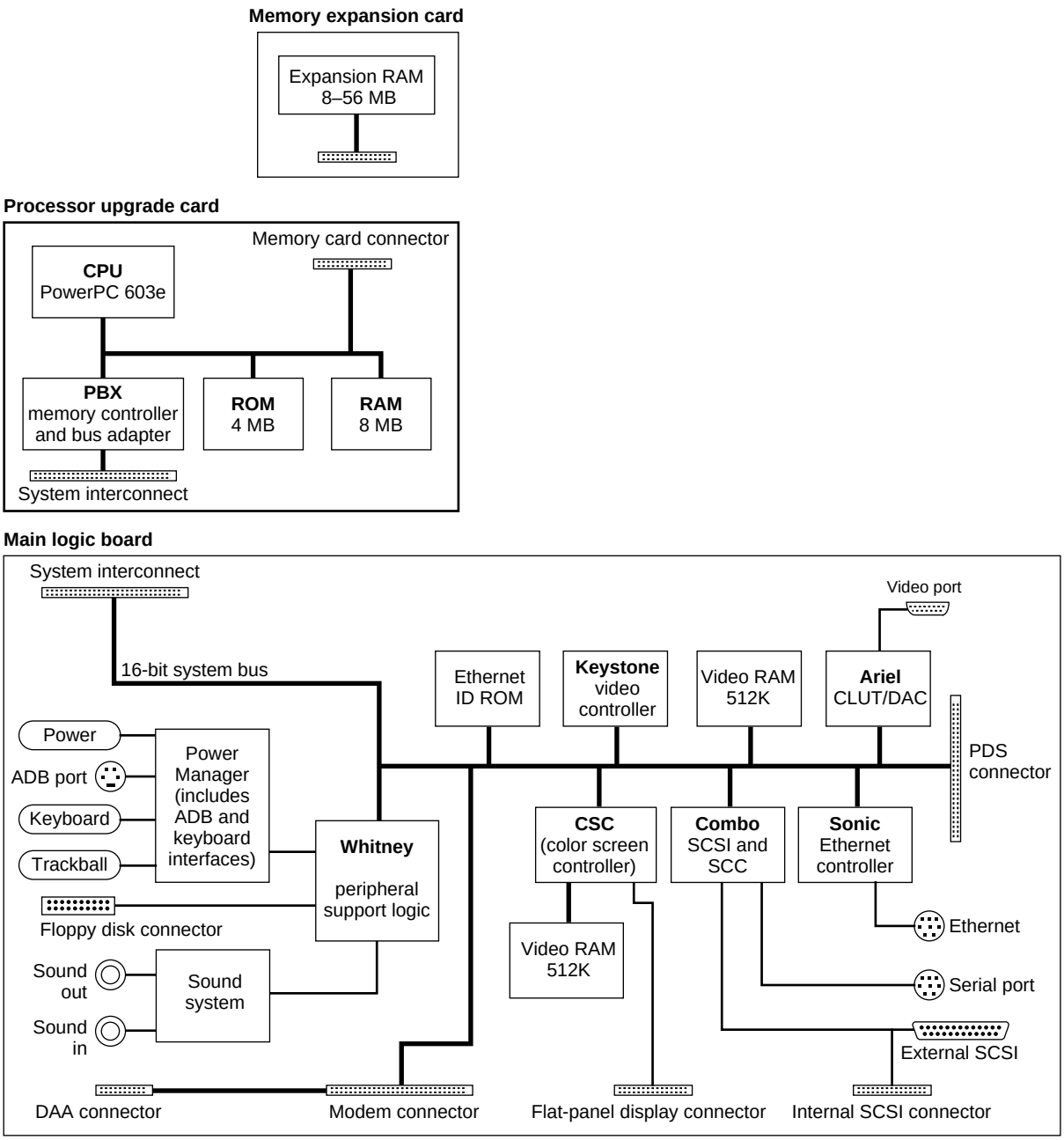
Whereas the PowerPC 601 microprocessor has a single cache for both instructions and data, the PowerPC 603 has separate instruction and data caches. As long as applications deal with executable code by using the Code Fragment Manager, cache coherency is maintained. Applications that bypass the Code Fragment Manager and generate executable code in memory, and that do not use the proper cache synchronization instructions or Code Fragment Manager calls, are likely to encounter problems when running on the PowerPC 603.

CHAPTER 2

Architecture

The Macintosh PowerBook Processor Card Upgrade Kit replaces the secondary logic board that contains the processor/memory subsystem in a Macintosh PowerBook 500 series computer. Figure 2-1 is the block diagram of a PowerBook 500 series computer with a processor upgrade card installed.

Figure 2-1 Block diagram of a computer with the processor upgrade card installed



The processor/memory subsystem on the upgrade card operates at 33 MHz on the PowerPC 603 bus. An Apple custom IC called the PBX IC acts as the bridge to the I/O bus, translating processor bus cycles into single or multiple I/O bus cycles, as needed.

Microprocessor

The microprocessor in the Macintosh PowerBook Processor Card Upgrade Kit is a PowerPC 603e, an enhanced version of the PowerPC 603. Its principal features include

- full RISC processing architecture
- a load-store unit that operates in parallel with the processing units
- a branch manager that can usually implement branches by reloading the incoming instruction queue without using any processing time
- two internal memory management units (MMUs), one for instructions and one for data
- two separate on-chip caches for data and instructions, of 16 KB each

For complete technical details, see *PowerPC 603 Microprocessor Implementation Definition Book IV.*

Main Memory

The Macintosh PowerBook Processor Card Upgrade Kit contains the main RAM, the system ROM, and a connector for an optional RAM expansion card.

RAM

The processor upgrade card contains built-in RAM consisting of 8 MB of dynamic RAM (DRAM). The RAM ICs are low-power, self-refreshing type with an access time of 70 ns.

The PBX custom IC contains bank base registers that are used to make RAM addresses contiguous, starting at address $0000 0000. See “Memory Controller IC” on page 8.

RAM Expansion

A connector on the secondary logic board accepts a RAM expansion card containing from 8 to 56 MB of self-refreshing dynamic RAM.

IMPORTANT
The RAM expansion card for the processor upgrade card is the same as the one for unmodified Macintosh PowerBook 500 series computers. ▲

The RAM expansion card can contain from one to four identical banks, with 2, 4, or 8 MB in each bank. Table 2-1 shows how the banks can be implemented with standard RAM devices.

Table 2-1 Configurations of RAM banks

Size of bank	Number of devices per bank	Device size (bits)
2 MB	4	512K × 8
4 MB	8	1 M × 4
4 MB	2	1 M × 16
8 MB	4	2 M × 8

▲ **WARNING**
Installation of a RAM expansion card must be done by an experienced technician. Care is required to avoid damage to the pins on the RAM expansion connector. ▲

ROM

The ROM in the processor upgrade card is implemented as a 1 M by 32-bit array consisting of two 1 M by 16-bit ROM ICs. The ROM devices support burst mode so they do not degrade the performance of the PowerPC 603 microprocessor. The ROM ICs provide 4 MB of storage, which is located in the system memory map between addresses $3000 0000 and $3FFF FFFF. The ROM data path is 32 bits wide and addressable only as longwords. See Chapter 3, “Software Features,” for a description of the features of this new ROM.

Memory Controller IC

The memory controller in the processor upgrade card is the PBX IC, a custom IC that provides RAM and ROM memory control and also acts as the bridge between the processor bus on the secondary logic board and the 68030-type I/O bus on the main logic board. The PBX IC also provides bus cycle decoding for the SWIM floppy disk controller.

Memory Control

The PBX IC controls the system RAM and ROM and provides address multiplexing and refresh signals for the DRAM devices. For information about the address multiplexing, see “Address Multiplexing” on page 9.

The PBX IC has a memory bank decoder in the form of an indexed register file. Each nibble in the register file represents a 2 MB page in the memory address space (64 MB). The value in each nibble maps the corresponding page to one of the eight banks of physical RAM. By writing the appropriate values into the register file at startup time, the system software makes the memory addresses contiguous.

Bus Bridge

The PBX IC acts as a bridge between the processor bus and the I/O bus. The bridge functions are performed by two converters inside the PBX IC. One converter accepts requests from the processor bus and presents them to the I/O bus in a manner consistent with a 68030 microprocessor. The other converter accepts requests from the I/O bus and provides access to the RAM and ROM on the processor bus.

The bus bridge in the PBX IC runs asynchronously so that the processor bus and the I/O bus can operate at different rates. The processor bus operates at a clock rate of 33 MHz, and the I/O bus operates at 25 MHz.

Address Multiplexing

Different types of DRAM devices require different row and column address multiplexing. The operation of the multiplexing is determined by the way the address pins on the devices are connected to individual signals on the RAM expansion card connector and depends on the exact type of DRAM used.

IMPORTANT

The PBX IC in the processor upgrade card provides exactly the same RAM address multiplexing as that provided by the Pratt IC used in unmodified Macintosh PowerBook 500 series computers. ▲

Table 2-2 (on the next page) shows how the signals on the address bus are connected for several types of DRAM devices. The device types are specified by their size and by the number of row and column address bits they require.

Table 2-2 shows how the signals are multiplexed during the row and column address phases. For each type of DRAM device, the first and second rows show the actual address bits that drive each address pin during row addressing and column addressing, respectively. The third row shows how the device's address pins are connected to the signals on the DRAM_ADDR bus.

IMPORTANT

Some types of DRAM devices don't use all 12 bits in the row or column address. Numbers for the unused address bits are shown in italic type; numbers for the bits that are used are shown in boldface type. ▲

Table 2-2 Address multiplexing for some typical DRAM devices

	Individual signals on DRAM_ADDR bus											
Type of DRAM device	[11]	[10]	[9]	[8]	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
2 M × 8, 12 row bits, 9 column bits												
Row address bits	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**
Column address bits	*19*	*21*	*18*	**22**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Device address pins	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1 M × 16, 12 row bits, 8 column bits												
Row address bits	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**
Column address bits	*19*	*21*	*18*	*22*	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Device address pins	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1 M × 4, 12 row bits, 8 column bits												
Row address bits	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**
Column address bits	*19*	*21*	*18*	*22*	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Device address pins	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
2 M × 8, 11 row bits, 10 column bits												
Row address bits	**21**	**20**	*19*	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**
Column address bits	**19**	*21*	*18*	**22**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Device address pins	**9**	**10**	**—**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1 M × 16, 10 row bits, 10 column bits												
Row address bits	*21*	**20**	**19**	*18*	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**
Column address bits	*19*	**21**	**18**	*22*	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Device address pins	**—**	**9**	**8**	**—**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
512K × 8, 10 row bits, 9 column bits												
Row address bits	*21*	**20**	**19**	*18*	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**
Column address bits	*19*	*21*	**18**	*22*	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Device address pins	**—**	**9**	**8**	**—**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**

CHAPTER 3

Software Features

This chapter describes the new ROM and system software features that the Macintosh PowerBook Processor Card Upgrade Kit brings to the PowerBook 500 series computers.

ROM Software

The ROM software in the Macintosh PowerBook Processor Card Upgrade Kit is based on the ROM used in previous Macintosh PowerBook computers, enhanced with many new features. New features and changes include the following:

- control of PowerPC 603 power modes
- machine identification
- support of new memory controller IC
- new Power Manager software
- new sound features
- Ethernet driver

The following sections describe each of these features.

PowerPC 603 Power Modes

The PowerPC 603 microprocessor has power-saving modes similar to the power-cycling and sleep modes of earlier Macintosh PowerBook models. The ROM has been modified to include the additional traps needed to control the power modes of the microprocessor.

Machine Identification

The ROM includes new tables and code for identifying the machine.

Applications can find out which computer they are running on by using the Gestalt Manager. The `gestaltMachineType` value returned by the processor upgrade card is 120 (hexadecimal $78). *Inside Macintosh: Overview* describes the Gestalt Manager and tells how to use the `gestaltMachineType` value to obtain the machine name string.

New Memory Controller IC

The memory control routines have been rewritten to operate with the PBX memory controller IC, which has a control register configuration different from that of the Pratt memory controller IC used in the Macintosh PowerBook 500 series computers. The memory initialization and size code has been rewritten to deal with

- larger ROM size
- a new type of DRAM devices
- new memory configurations

Power Manager Software

Changes to the Power Manager software include

- power-cycling and sleep mode for the PowerPC 603 microprocessor
- support for the new lithium ion batteries
- support for turning on and off power to the Ethernet interface

Like other current Macintosh PowerBook models, the Macintosh PowerBook Processor Card Upgrade Kit supports the public API for power management, which is described in *Inside Macintosh: Devices.*

Sound Features

The ROM software includes new sound driver software to support the new Sound Manager, which is part of the system software. The driver software also supports the following new features:

- improved sound performance by way of a new interface to the Singer sound IC
- support for 16-bit stereo sound input
- support for automatic gain control in software
- mixing of sound output from the modem

The new ROM software also includes routines to arbitrate the control of the sound hardware between the modem and the Sound Manager.

Ethernet Driver

The driver for the Ethernet interface can now put a sleep task for Ethernet into the Power Manager's sleep table. This sleep task first makes a control call to the Ethernet driver to prepare the Ethernet interface IC for sleep mode. The sleep task then makes a Power Manager call to turn off power to the IC. The sleep task installs a corresponding wake task that turns the interface power back on and reinitializes the interface IC.

System Software

The Macintosh PowerBook Processor Card Upgrade Kit comes with new system software based on System 7.5 and augmented by several new features.

IMPORTANT

Even though the software for the processor upgrade card incorporates significant changes from System 7.5, it is not a reference release: that is, it is not an upgrade for earlier Macintosh models. ▲

The system software includes changes in the following areas:

- control strip support (introduced on the Macintosh PowerBook 280 and 500 models)
- large partition support
- Drive Setup, a new utility
- improved file sharing
- a new Dynamic Recompilation Emulator
- the Resource Manager completely in native code
- an improved math library
- new `BlockMove` extensions
- POWER-clean native code
- POWER emulation
- support for native drivers

These changes are described in the sections that follow.

Control Strip

The Macintosh PowerBook Processor Card Upgrade Kit includes the latest version of the control strip that was introduced in the Macintosh PowerBook 280 and 500 models. It is a strip of graphics with small button controls and indicators in the form of various icons.

For updated developer guidelines, refer to Macintosh Technical Note OS 6, *Control Strip Modules,* on the reference library edition of the developer CD.

Large Partition Support

The largest disk partition supported by System 7.5 is 4 GB. The new system software extends that limit to 2 terabytes.

IMPORTANT

The largest possible file is still just under 2 GB. ▲

The changes necessary to support the larger partition size affect many parts of the system software. The affected software includes system-level and application-level components. See Chapter 4, "Large Volume Support."

64-Bit Volume Addresses

The current disk driver API has a 32-bit volume address limitation. This limitation has been circumvented by the addition of a new 64-bit extended volume API (`PBXGetVolInfo`) and 64-bit data types (`UnsignedWide`, `Wide`, `XVolumeParam`, and `XIOParam`).

For the definitions of the new API and data types, please see “The API Modifications” in Chapter 4, “Large Volume Support.”

System-Level Software

Several system components have been modified to use the 64-bit API to correctly calculate true volume sizes and read and write data to and from large disks. The modified system components are

- virtual memory code
- Disk INIT
- FSM INIT
- Apple disk drivers
- file system ROM code

Application-Level Software

Current applications do not require modification to gain access to disk space beyond the traditional 4 GB limit as long as they do not require the true size of the large partition. Applications that need to obtain the true partition size must be modified to use the new 64-bit API and data structures. Typical applications include utilities for disk formatting, partitioning, initialization, and backup.

The following application-level components of the system software have been modified to use the 64-bit API:

- Finder
- Finder extensions (AppleScript, AOCE Mailbox, and Catalogs)
- HDSC Setup
- Disk First Aid

In the past, the sum of the sizes of the files and folders selected in the Finder was limited to the largest value that could be stored in a 32-bit number—that is, 4 GB. By using the new 64-bit API and data structures, the Finder can now operate on selections whose total size exceeds that limit. Even with very large volumes, the Finder can display accurate information in Folder and Get Info windows and can obtain the true volume size for calculating available space when copying.

The Finder extensions AppleScript, AOCE Mailbox, and Catalogs have been modified in the same way as the Finder because their copy-engine code is similar to that in the Finder.

A later section describes the modified Drive Setup utility.

Limitations

The software modifications that support large partition sizes do not solve all the problems associated with the use of large volumes. In particular, the modifications do not address the following:

- HFS file sizes are still limited to 2 GB or less.
- Large allocation block sizes cause inefficient storage. On a 2 GB volume, the minimum file size is 32 KB; on a 2-terabyte volume, the minimum file size would be a whopping 32 MB.
- Drives with the new large volume driver will not mount on computers running older versions of the Macintosh Operating System.

Drive Setup

The software for the processor upgrade card includes a new disk setup utility named Drive Setup. In addition to the ability to support large volumes, the Drive Setup utility has several other enhancements over the older HDSC Setup utility, including

- an improved user interface
- support for multiple partitions
- the ability to mount volumes from applications
- the ability to start up (boot) from any HFS partition
- support for removable-media drives

Dynamic Recompilation Emulator

The Dynamic Recompilation Emulator (or DR Emulator) is an extension to the current interpretive emulator providing on-the-fly translation of 680x0 instructions into PowerPC instructions for increased performance. The DR Emulator operates as an enhancement to a modified version of the existing interpretive emulator.

The design of the DR Emulator mimics a hardware instruction cache and employs a variable-size translation cache. Each compiled 680x0 instruction requires on average fewer than 20 PowerPC instructions. In operation, the DR Emulator depends on locality of execution to make up for the extra cycles used in translating the code.

The DR Emulator provides a high degree of compatibility for 680x0 code. One area where compatibility will be less than that of the current interpretive emulator is for self-modifying code that does not call the cache flushing routines. Such code also has compatibility problems on Macintosh Quadra models with the cache enabled.

Resource Manager in Native Code

The Resource Manager in the software for the Macintosh PowerBook Processor Card Upgrade Kit is similar to the one in the earlier Power Macintosh computers except that it is completely in native PowerPC code. Because the Resource Manager is intensively used

both by system software and by applications, the native version provides an improvement in system performance.

The Process Manager has been modified to remove patches it formerly made to the Resource Manager.

Math Library

The new math library (MathLib) is an enhanced version of the floating-point library included in the ROM in the first generation of Power Macintosh computers.

The new math library is bit compatible in both results and floating-point exceptions with the math library in the first-generation ROM. The only difference is in the speed of computation.

The new math library has been improved to better exploit the floating-point features of the PowerPC microprocessor. The math library now includes enhancements that assist the compiler in carrying out its register allocation, branch prediction, and overlapping of integer and floating-point operations.

Compared with the previous version, the new math library provides much improved performance without compromising its accuracy or robustness. It provides performance gains for often-used functions of up to 15 times.

The application interface and header files for the math library have not been changed.

New BlockMove Extensions

The system software for the Macintosh PowerBook Processor Card Upgrade Kit includes new extensions to the `BlockMove` routine. The extensions provide improved performance for programs running in native mode.

The new `BlockMove` extensions provide the following benefits for developers:

- They're optimized for the PowerPC 603 and PowerPC 604 microprocessors, rather than the PowerPC 601.
- They're compatible with the new Dynamic Recompilation Emulator.
- They provide a way to handle cache-inhibited address spaces.
- They include new high-speed routines for setting memory to 0.

Note
The new `BlockMove` extensions do not use the string instructions, which are fast on the PowerPC 601 but slow on other PowerPC implementations. ◆

Some of the new `BlockMove` extensions can be called only from native code; see Table 3-1.

Except for `BlockZero` and `BlockZeroUncached`, the new `BlockMove` extensions use the same parameters as `BlockMove`. Calls to `BlockZero` and `BlockZeroUncached` have only two parameters, a pointer and a length; refer to the header file (`Memory.h`).

Table 3-1 summarizes the `BlockMove` routines according to three criteria: whether the routine can be called from 680x0 code, whether it is okay to use for moving 680x0 code, and whether it is okay to use with buffers or other uncacheable destination locations.

Table 3-1 Summary of `BlockMove` routines

`BlockMove` version	Can be called from 680x0 code?	OK to use for moving 680x0 code?	OK to use with buffers?
`BlockMove`	Yes	Yes	No
`BlockMoveData`	Yes	No	No
`BlockMoveDataUncached`	No	No	Yes
`BlockMoveUncached`	No	Yes	Yes
`BlockZero`	No	—	No
`BlockZeroUncached`	No	—	Yes

The fastest way to move data is to use the `BlockMoveData` routine. It is the recommended method whenever you are certain that the data is cacheable and does not contain executable 680x0 code.

The `BlockMove` routine is slower than the `BlockMoveData` routine only because it has to clear out the software cache used by the DR Emulator. If the DR Emulator is not in use, the `BlockMove` routine and the `BlockMoveData` routine are the same.

IMPORTANT

The versions of `BlockMove` for cacheable data use the `dcbz` instruction to avoid unnecessary prefetching of destination cache blocks. For uncacheable data, you should avoid using those routines because the `dcbz` instruction faults and must be emulated on uncacheable or write-through locations, making execution extremely slow. ▲

IMPORTANT

Driver software cannot call the `BlockMove` routines directly. Instead, drivers must use the `BlockCopy` routine, which is part of the Driver Services Library. The `BlockCopy` routine is an abstraction that allows you to postpone binding the specific type of `BlockMove` operation until implementation time. ▲

The Driver Services Library is a collection of useful routines that Apple Computer provides for developers working with the new Power Macintosh models. For more information, please refer to *Designing PCI Cards and Drivers for Power Macintosh Computers.*

POWER-Clean Native Code

The instruction set of the PowerPC 601 microprocessor includes some of the same instructions as those found in the instruction set of the POWER processor, and the compiler used to generate native code for the system software in the first generation of Power Macintosh models generated some of those POWER-only instructions. However, the PowerPC 603 IC used in the processor upgrade card does not support the POWER-only instructions, so a new POWER-clean version of the compiler is being used to compile the native code fragments.

Note
The term *POWER-clean* refers to code that is free of the POWER instructions that would prevent it from running correctly on a PowerPC 603 or PowerPC 604 microprocessor. ◆

Here is a list of the POWER-clean native code fragments in the system software for the processor upgrade card.

- interface library
- private interface library
- native QuickDraw
- MathLib
- Mixed Mode Manager
- Code Fragment Manager
- Font Dispatch
- Memory Manager
- standard text
- the `FMSwapFont` function
- Standard C Library

POWER Emulation

Earlier Power Macintosh computers included emulation for certain PowerPC 601 instructions that would otherwise cause an exception. The emulation code dealt with memory reference instructions to handle alignment and data storage exceptions. It also handled illegal instruction exceptions caused by some PowerPC instructions that were not implemented in the PowerPC 601. With the Macintosh PowerBook Processor Card Upgrade Kit, the emulation code has been enhanced to include the POWER instructions that are implemented on the PowerPC 601 but not on the PowerPC 603 or PowerPC 604.

Note
Although the term *POWER emulation* is often used, a more appropriate name for this feature is *PowerPC 601 compatibility.* Rather than supporting the entire POWER architecture, the goal is to support those features of the POWER architecture that are available to programs running in user mode on the PowerPC 601–based Power Macintosh computers. ◆

POWER-Clean Code

Because the emulation of the POWER-only instructions degrades performance, Apple Computer recommends that developers revise any applications that use those instructions to conform with the PowerPC architecture. POWER emulation works, but at a significant cost in performance; POWER-clean code is preferable.

Emulation and Exception Handling

When an exception occurs, the emulation code first checks to see whether the instruction encoding is supported by emulation. If it is not, the code passes the original cause of the exception (illegal instruction or privileged instruction) to the application as a native exception.

If the instruction is supported by emulation, the code then checks a flag bit to see whether emulation has been enabled. If emulation is not enabled at the time, the emulator generates an illegal instruction exception.

Code Fragments and Cache Coherency

Whereas the PowerPC 601 microprocessor has a single cache for both instructions and data, the PowerPC 603 has separate instruction and data caches. As long as applications deal with executable code by using the Code Fragment Manager, cache coherency is maintained. Applications that bypass the Code Fragment Manager and generate executable code in memory, and that do not use the proper cache synchronization instructions or Code Fragment Manager calls, are likely to encounter problems when running on the PowerPC 603.

IMPORTANT
The emulation software in the Macintosh PowerBook Processor Card Upgrade Kit cannot make the separate caches in the PowerPC 603 behave like the combined cache in the PowerPC 601. Applications that generate executable code in memory must be modified to use the Code Fragment Manager or maintain proper cache synchronization by other means. ▲

Limitations of PowerPC 601 Compatibility

The emulation code in the Macintosh PowerBook Processor Card Upgrade Kit allows programs compiled for the PowerPC 601 to execute without halting on an exception whenever they use a POWER-only feature. For most of those features, the emulation

matches the results that are obtained on a Power Macintosh computer with a PowerPC 601. However, there are a few cases where the emulation is not an exact match; those cases are summarized here.

- **MQ register.** Emulation does not match the undefined state of this register after multiply and divide instructions.
- **`div` and `divo` instructions.** Emulation does not match undefined results after an overflow.
- **Real-time clock registers.** Emulation matches the 0.27 percent speed discrepancy of the Power Macintosh models that use the PowerPC 601 microprocessor, but the values of the low-order 7 bits are not 0.
- **POWER version of `dec` register.** Emulation includes the POWER version, but decrementing at a rate determined by the time base clock, not by the real-time clock.
- **Cache line compute size (`clcs`) instruction.** Emulation returns values appropriate for the type of PowerPC microprocessor.
- **Undefined special-purpose register (SPR) encodings.** Emulation does not ignore SPR encodings higher than 32.
- **Invalid forms.** Invalid combinations of register operands with certain instructions may produce results that do not match those of the PowerPC 601.
- **Floating-point status and control register (FPSCR).** The FPSCR in the PowerPC 601 does not fully conform to the PowerPC architecture, but the newer PowerPC microprocessors do.

Support for Native Drivers

The processor upgrade card uses a new native-driver model for system software and device driver developers. Several components of system software are being modified to support native drivers. The following components have been modified:

- the Device Manager
- interrupt tree services
- driver loader library
- driver support library
- Slot Manager stubs
- Macintosh startup code
- interface libraries
- system registry

For more information, refer to *Designing PCI Cards and Drivers for Power Macintosh Computers.*

CHAPTER 4

Large Volume Support

This chapter describes the large volume file system for the Macintosh PowerBook Processor Card Upgrade Kit. The large volume file system is a version of the hierarchical file system (HFS) that has been modified to support volume sizes larger than the current 4 GB limit. It incorporates only the changes required to achieve that goal.

Overview of the Large Volume File System

The large volume file system includes

- modifications to the HFS ROM code, Disk First Aid, and Disk INIT
- a new extended API that allows reporting of volume size information beyond the current 4 GB limit
- new device drivers and changes to the Device Manager API to support devices that are greater than 4 GB
- a new version of HDSC Setup that supports large volumes and chainable drivers (Chainable drivers are needed to support booting large volumes on earlier Macintosh models.)

API Changes

The system software on the processor upgrade card allows all current applications to work without modifications. Unmodified applications that call the file system still receive incorrect values for large volume sizes. The Finder and other utility programs that need to know the actual size of a volume have been modified to use the new extended `PBXGetVolInfo` function to obtain the correct value.

The existing low-level driver interface does not support I/O to a device with a range of addresses greater than 4 GB because the positioning offset (in bytes) for a read or write operation is a 32-bit value. To correct this problem, a new extended I/O parameter block record has been defined. This extended parameter block has a 64-bit positioning offset. The new parameter block and the extended `PBXGetVolInfo` function are described in "The API Modifications" beginning on page 25.

Allocation Block Size

The format of HFS volumes has not changed. What has changed is the way the HFS software handles the allocation block size. Existing HFS code treats the allocation block as a 16-bit integer. The large volume file system uses the full 32 bits of the allocation block size parameter. In addition, any software that deals directly with the allocation block size from the volume control block must now treat it as a true 32-bit value.

Even for the larger volume sizes, the number of allocation blocks is still defined by a 16-bit integer. As the volume size increases, the size of the allocation block also increases. For a 2 GB volume, the allocation block size is 32 KB, and therefore the smallest file on that disk will occupy at least 32 KB of disk space. This inefficient use of disk space is not addressed by the large volume file system.

The maximum number of files will continue to be less than 65,000. This limit is directly related to the fixed number of allocation blocks.

File Size Limits

The HFS has a maximum file size of 2 GB. The large volume file system does not remove that limit because doing so would require a more extensive change to the current API and would incur more compatibility problems.

Compatibility Requirements

The large volume file system requires at least a 68020 microprocessor or a Power Macintosh model that emulates it. In addition, the file system requires a Macintosh IIci or more recent model. On a computer that does not meet both those requirements, the large volume file system driver will not load.

The large volume file system requires System 7.5 or higher and a new Finder that supports volumes larger than 4 GB (using the new extended `PBXGetVolInfo` function).

The API Modifications

The HFS API has been modified to support volume sizes larger than 4 GB. The modifications consist of two extended data structures and a new extended `PBXGetVolInfo` function.

Data Structures

This section describes the two modified data structures used by the large volume file system:

- the extended volume parameter block
- the extended I/O parameter block

Extended Volume Parameter Block

In the current `HVolumeParam` record, volume size information is clipped at 2 GB. Because HFS volumes can now exceed 4 GB, a new extended volume parameter block is needed in order to report the larger size information. The `XVolumeParam` record contains 64-bit integers for reporting the total bytes on the volume and the number of free bytes available (parameter names `ioVTotalBytes` and `ioVFreeBytes`). In addition, several of the fields that were previously signed are now unsigned (parameter names `ioVAtrb`, `ioVBitMap`, `ioAllocPtr`, `ioVAlBlkSiz`, `ioVClpSiz`, `ioAlBlSt`, `ioVNxtCNID`, `ioVWrCnt`, `ioVFilCnt`, and `ioVDirCnt`).

```
struct XVolumeParam {
   ParamBlockHeader
   unsigned long     ioXVersion;     // XVolumeParam version == 0
   short             ioVolIndex;     // volume index
   unsigned long     ioVCrDate;      // date & time of creation
   unsigned long     ioVLsMod;       // date & time of last modification
   unsigned short    ioVAtrb;        // volume attributes
   unsigned short    ioVNmFls;       // number of files in root directory
   unsigned short    ioVBitMap;      // first block of volume bitmap
   unsigned short    ioAllocPtr;     // first block of next new file
   unsigned short    ioVNmAlBlks;    // number of allocation blocks
   unsigned long     ioVAlBlkSiz;    // size of allocation blocks
   unsigned long     ioVClpSiz;      // default clump size
   unsigned short    ioAlBlSt;       // first block in volume map
   unsigned long     ioVNxtCNID;     // next unused node ID
   unsigned short    ioVFrBlk;       // number of free allocation blocks
   unsigned short    ioVSigWord;     // volume signature
   short             ioVDrvInfo;     // drive number
   short             ioVDRefNum;     // driver reference number
   short             ioVFSID;        // file-system identifier
   unsigned long     ioVBkUp;        // date & time of last backup
   unsigned short    ioVSeqNum;      // used internally
   unsigned long     ioVWrCnt;       // volume write count
   unsigned long     ioVFilCnt;      // number of files on volume
   unsigned long     ioVDirCnt;      // number of directories on volume
   long              ioVFndrInfo[8]; // information used by the Finder
   uint64            ioVTotalBytes;  // total number of bytes on volume
   uint64            ioVFreeBytes;   // number of free bytes on volume
};
```

Field descriptions

`ioVolIndex`	An index for use with the `PBHGetVInfo` function.
`ioVCrDate`	The date and time of volume initialization.
`ioVLsMod`	The date and time the volume information was last modified. (This field is not changed when information is written to a file and does not necessarily indicate when the volume was flushed.)
`ioVAtrb`	The volume attributes.
`ioVNmFls`	The number of files in the root directory.
`ioVBitMap`	The first block of the volume bitmap.
`ioAllocPtr`	The block at which the next new file starts. Used internally.
`ioVNmAlBlks`	The number of allocation blocks.
`ioVAlBlkSiz`	The size of allocation blocks.
`ioVClpSiz`	The clump size.

`ioAlBlSt`	The first block in the volume map.
`ioVNxtCNID`	The next unused catalog node ID.
`ioVFrBlk`	The number of unused allocation blocks.
`ioVSigWord`	A signature word identifying the type of volume; it's $D2D7 for MFS volumes and $4244 for volumes that support HFS calls.
`ioVDrvInfo`	The drive number of the drive containing the volume.
`ioVDRefNum`	For online volumes, the reference number of the I/O driver for the drive identified by `ioVDrvInfo`.
`ioVFSID`	The file-system identifier. It indicates which file system is servicing the volume; it's zero for File Manager volumes and nonzero for volumes handled by an external file system.
`ioVBkUp`	The date and time the volume was last backed up (it's 0 if never backed up).
`ioVSeqNum`	Used internally.
`ioVWrCnt`	The volume write count.
`ioVFilCnt`	The total number of files on the volume.
`ioVDirCnt`	The total number of directories (not including the root directory) on the volume.
`ioVFndrInfo`	Information used by the Finder.

Extended I/O Parameter Block

The extended I/O parameter block is needed for low-level access to disk addresses beyond 4 GB. It is used exclusively by `PBRead` and `PBWrite` calls when performing I/O operations at offsets greater than 4 GB. To indicate that you are using an `XIOParam` record, you should set the `kUseWidePositioning` bit in the `ioPosMode` field.

Because file sizes are limited to 2 GB, the regular `IOParam` record should always be used when performing file-level I/O operations. The extended parameter block is intended only for Device Manager I/O operations to large block devices at offsets greater than 4 GB.

The only change in the parameter block is the parameter `ioWPosOffset`, which is of type `int64`.

```
struct XIOParam {
   QElemPtr qLink;   // next queue entry
   short    qType;   // queue type
   short    ioTrap;  // routine trap
   Ptr      ioCmdAddr;// routine address
   ProcPtr  ioCompletion;// pointer to completion routine
   OSErr    ioResult;// result code
   StringPtrioNamePtr;// pointer to pathname
   short    ioVRefNum;// volume specification
   short    ioRefNum;// file reference number
   char     ioVersNum;// not used
```

```
   char     ioPermssn;// read/write permission
   Ptr      ioMisc;  // miscellaneous
   Ptr      ioBuffer;// data buffer
   unsigned longioReqCount;// requested number of bytes
   unsigned longioActCount;// actual number of bytes
   short    ioPosMode;// positioning mode (wide mode set)
   int64    ioPosOffset;// wide positioning offset
};
```

Field descriptions

`ioRefNum` The file reference number of an open file.

`ioVersNum` A version number. This field is no longer used; you should always set it to 0.

`ioPermssn` The access mode.

`ioMisc` Depends on the routine called. This field contains either a new logical end-of-file, a new version number, a pointer to an access path buffer, or a pointer to a new pathname. Because `ioMisc` is of type `Ptr`, you'll need to perform type coercion to interpret the value of `ioMisc` correctly when it contains an end-of-file (a `LongInt` value) or version number (a `SignedByte` value).

`ioBuffer` A pointer to a data buffer into which data is written by `_Read` calls and from which data is read by `_Write` calls.

`ioReqCount` The requested number of bytes to be read, written, or allocated.

`ioActCount` The number of bytes actually read, written, or allocated.

`ioPosMode` The positioning mode for setting the mark. Bits 0 and 1 of this field indicate how to position the mark; you can use the following predefined constants to set or test their value:

```
CONST
fsAtMark = 0;     {at current mark}
fsFromStart = 1;  {from beginning of file}
fsFromLEOF = 2;   {from logical end-of-file}
fsFromMark = 3;   {relative to current mark}
```

You can set bit 4 of the `ioPosMode` field to request that the data be cached, and you can set bit 5 to request that the data not be cached. You can set bit 6 to request that any data written be immediately read; this ensures that the data written to a volume exactly matches the data in memory. To request a read-verify operation, add the following constant to the positioning mode:

```
CONST
rdVerify = 64; {use read-verify mode}
```

You can set bit 7 to read a continuous stream of bytes, and place the ASCII code of a newline character in the high-order byte to terminate a read operation at the end of a line.

`ioPosOffset` The offset to be used in conjunction with the positioning mode.

New Extended Function

This section describes the extended `PBXGetVolInfo` function that provides volume size information for volumes greater than 4 GB.

Before using the new extended call, you should check for availability by calling the `Gestalt` function. Make your call to `Gestalt` with the `gestaltFSAttr` selector to check for new File Manager features. The response parameter has the `gestaltFSSupports2TBVolumes` bit set if the File Manager supports large volumes and the new extended function is available.

PBXGetVolInfo

You can use the `PBXGetVolInfo` function to get detailed information about a volume. It can report volume size information for volumes up to 2 terabytes.

```
pascal OSErr PBXGetVolInfo (XVolumeParam paramBlock, Boolean async);
```

`paramBlock`	A pointer to an extended volume parameter block.
`async`	A Boolean value that specifies asynchronous (`true`) or synchronous (`false`) execution.

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

Arrow	Meaning
→	Input
←	Output
↔	Both

Parameter block

→	`ioCompletion`	`ProcPtr`	Pointer to a completion routine.
←	`ioResult`	`OSErr`	Result code of the function.
↔	`ioNamePtr`	`StringPtr`	Pointer to the volume's name.
↔	`ioVRefNum`	`short`	On input, a volume specification; on output, the volume reference number.
→	`ioXVersion`	`unsigned long`	Version of `XVolumeParam` (value = 0).
→	`ioVolIndex`	`short`	Index used for indexing through all mounted volumes.
←	`ioVCrDate`	`unsigned long`	Date and time of initialization.
←	`ioVLsMod`	`unsigned long`	Date and time of last modification.

continued

←	`ioVAtrb`	`unsigned short`	Volume attributes.
←	`ioVNmFls`	`unsigned short`	Number of files in the root directory.
←	`ioVBitMap`	`unsigned short`	First block of the volume bitmap.
←	`ioVAllocPtr`	`unsigned short`	Block where the next new file starts.
←	`ioVNmAlBlks`	`unsigned short`	Number of allocation blocks.
←	`ioVAlBlkSiz`	`unsigned long`	Size of allocation blocks.
←	`ioVClpSiz`	`unsigned long`	Default clump size.
←	`ioAlBlSt`	`unsigned short`	First block in the volume block map.
←	`ioVNxtCNID`	`unsigned long`	Next unused catalog node ID.
←	`ioVFrBlk`	`unsigned short`	Number of unused allocation blocks.
←	`ioVSigWord`	`unsigned short`	Volume signature.
←	`ioVDrvInfo`	`short`	Drive number.
←	`ioVDRefNum`	`short`	Driver reference number.
←	`ioVFSID`	`short`	File system handling this volume.
←	`ioVBkUp`	`unsigned long`	Date and time of last backup.
←	`ioVSeqNum`	`unsigned short`	Used internally.
←	`ioVWrCnt`	`unsigned long`	Volume write count.
←	`ioVFilCnt`	`unsigned long`	Number of files on the volume.
←	`ioVDirCnt`	`unsigned long`	Number of directories on the volume.
←	`ioVFndrInfo[8]`	`long`	Used by the Finder.
←	`ioVTotalBytes`	`uint64`	Total number of bytes on the volume.
←	`ioVFreeBytes`	`uint64`	Number of free bytes on the volume.

DESCRIPTION

The `PBXGetVolInfo` function returns information about the specified volume. It is similar to the `PBHGetVInfo` function described in *Inside Macintosh: Files* except that it returns additional volume space information in 64-bit integers.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for `PBXGetVolInfo` are

Trap macro	Selector
`_HFSDispatch`	$0012

RESULT CODES

`noErr`	0	Successful completion; no error occurred
`nsvErr`	–35	No such volume
`paramErr`	–50	No default volume

Glossary

ADB See **Apple Desktop Bus.**

APDA Apple Computer's worldwide direct distribution channel for Apple and third-party development tools and documentation products.

API See **application program interface.**

Apple Desktop Bus (ADB) An asynchronous bus used to connect relatively slow user-input devices to Apple computers.

AppleTalk Apple Computer's local area networking protocol.

application programming interface (API) The calls and data structures that allow application software to use the features of the operating system.

color depth The number of bits required to encode the color of each pixel in a display.

DAC See **digital-to-analog converter.**

data burst Multiple longwords of data sent over a bus in a single, uninterrupted stream.

data cache In a PowerPC microprocessor, the internal registers that hold data being processed.

digital-to-analog converter (DAC) A device that produces an analog electrical signal in response to digital data.

direct memory access (DMA) A process for transferring data rapidly into or out of RAM without passing it through a processor or buffer.

DLPI Data Link Provider Interface, the standard networking model used in Open Transport.

DMA See **direct memory access.**

DRAM See **dynamic random-access memory.**

DR Emulator The Dynamic Recompilation Emulator, an improved 680x0-code emulator for the PowerPC microprocessors.

dynamic random-access memory (DRAM) Random-access memory in which each storage address must be periodically interrogated ("refreshed") to maintain its value.

Ethernet A high-speed local area network technology that includes both cable standards and a series of communications protocols.

GeoPort A software and hardware solution for digital telecom and wide-area connectivity using the serial port.

input/output (I/O) Parts of a computer system that transfer data to or from peripheral devices.

I/O See **input/output.**

LocalTalk The cable terminations and other hardware that Apple supplies for local area networking from Macintosh serial ports.

native code Instructions that run directly on a PowerPC microprocessor. See also **680x0 code.**

nonvolatile RAM RAM that retains its contents even when the computer is turned off; also known as *parameter RAM*.

Open Transport A networking architecture that allows communications applications to run independently of the underlying network; formerly known as *Transport-Independent Interface (TII).*

PBX The custom IC that provides the interface between the PowerPC 603 bus and the I/O bus on the Macintosh PowerBook Processor Card Upgrade Kit with PowerPC 603e for the Macintosh PowerBook 500 series.

POWER A RISC architecture developed by IBM; the name is an acronym for performance optimized with enhanced RISC.

POWER-clean Refers to PowerPC code free of instructions that are specific to the PowerPC 601 and POWER instruction sets and are not found on the PowerPC 603 and PowerPC 604 microprocessors.

PowerPC Trade name for a family of RISC microprocessors. The PowerPC 601, 603, and 604 microprocessors are used in Power Macintosh computers.

PBX The custom IC that provides the interface between the PowerPC 603 bus and the I/O bus on the Macintosh PowerBook Processor Card Upgrade Kit with PowerPC 603e for the Macintosh PowerBook 500 series.

reduced instruction set computing (RISC) A technology of microprocessor design in which all machine instructions are uniformly formatted and are processed through the same steps.

RISC See **reduced instruction set computing.**

SCSI See **Small Computer System Interface.**

SIMM See **Single Inline Memory Module.**

Single Inline Memory Module (SIMM) A plug-in card for memory expansion, containing several RAM ICs and their interconnections.

680x0 code Instructions that can run on a PowerPC microprocessor only by means of an emulator. See also **native code.**

Small Computer System Interface (SCSI) An industry standard parallel bus protocol for connecting computers to peripheral devices such as hard disk drives.

Versatile Interface Adapter (VIA) The interface for system interrupts that is standard on most Apple computers.

VIA See **Versatile Interface Adapter**.

video RAM (VRAM) Random-access memory used to store both static graphics and video frames.

VRAM See **video RAM**.

Index

N, O

P, Q

R

S, T, U

V, W

X, Y, Z

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Proof pages and final pages were created on an Apple LaserWriter Pro printer. Line art was created using Adobe Illustrator™ and Adobe Photoshop™. PostScript™, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

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
Allen Watson

DEVELOPMENTAL EDITOR
Beverly Zegarski

ILLUSTRATOR
Sandee Karr

Special thanks to Paul Freeburn, Mike Puckett, and Charlie Tritschler