Developer Note

PowerBook Duo 280 and 280c Computers

Developer Note

May 1994
Developer Press

Contents

Figures and Tables

About This Note

This developer note describes the PowerBook Duo 280 and PowerBook Duo 280c computers, emphasizing the features that are new or different from those of the original PowerBook Duo computers. This developer note is a supplement to the *Macintosh PowerBook Duo Developer Note*, described in the section “Supplementary Documents,” later in this preface.

This note provides the hardware or software developer with the information needed to design hardware and software elements for the PowerBook Duo 280 and 280c computers. The note 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 “Supplementary Documents.”

Contents of This Note

This developer note contains four chapters and an appendix:

- Chapter 1, “Introduction,” describes the PowerBook Duo 280 and PowerBook Duo 280c computers and compares them with other PowerBook Duo models.
- Chapter 2, “Hardware Features,” describes the hardware features that are specific to the PowerBook Duo 280 and 280c computers.
- Chapter 3, “Software Features,” describes the software features that are specific to the PowerBook Duo 280 and 280c computers.
- Chapter 4, “Power Manager Interface,” describes the individual calls in the application programming interface for the Power Manager software.
- The Appendix, “Color Lookup Table,” describes the color lookup table used with the built-in color display.

The chapters and appendix are followed by an index.

Supplementary Documents

To supplement the information in this developer note, developers should have copies of the appropriate Motorola reference books for the MC68040 microprocessor. Software developers should have a copy of Motorola’s *MC68040 Programmer’s Reference Manual.* Hardware developers should have copies of Motorola’s *MC68040 User’s Manual* and *MC68040 Designer’s Handbook.*

Developers should also have copies of the appropriate Apple reference books, including *Inside Macintosh: Overview; Inside Macintosh: Processes; 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 original Macintosh PowerBook Duo computers, developers should have a copy of the *Macintosh PowerBook Duo Developer Note*, available on Apple's Developer CD Series as well as through APDA (order *Macintosh Developer Notes, Number 2,* APDA publication number R0457LL/A).

APDA is Apple's worldwide source for over three hundred development tools, technical resources, training products, and information for anyone interested in developing applications on Apple platforms. Customers receive the quarterly *APDA Tools Catalog* featuring all current versions of Apple development tools and the most popular third-party development tools. Ordering is easy; there are no membership fees, and application forms are not required for most of our products. 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

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.

Sidebar

A sidebar is used for information that is not part of the main discussion. A sidebar may contain information about a related subject or technical details that are not required reading.

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

A slash in front of a signal name (/RESET) indicates an active low signal.

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 information that is essential to an understanding of the text or of the PowerBook Duo 280 and 280c computers. ▲

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

Abbreviations

Standard units of measure used in this note include

K	1024	mm	millimeters
KB	kilobytes	ms	milliseconds
mA	milliamperes	ns	nanoseconds
MB	megabytes	V	volts
MHz	megahertz	VRMS	volts root-mean-square

Other abbreviations used in this note include

$*n*	hexadecimal value *n*
AC	alternating current
ADB	Apple Desktop Bus
API	application programming interface
C	capacitance
CAS	column address strobe (a memory control signal)
CCFL	cold cathode fluorescent lamp
CLUT	color lookup table
CPU	central processing unit (the main microprocessor)

CRT	cathode ray tube (video display device)
CSC	color screen controller (a custom IC)
CTB	Communications Toolbox
DC	direct current
DRAM	dynamic RAM
ECC	error checking and correction
FPU	floating-point unit
FSTN	film supertwist nematic (a type of LCD)
GSC	grayscale controller
IC	integrated circuit
I/O	input/output
LCD	liquid crystal display
LED	light-emitting diode
MBT	microprocessor bus translator (a custom IC)
MMU	memory management unit
MSC	main system controller (a custom IC)
NiCad	nickel cadmium
NiMH	nickel metal hydride
PDS	processor-direct slot
PRAM	parameter RAM (non-volatile RAM)
RAM	random-access memory
ROM	read-only memory
SCC	Serial Communications Controller
SCSI	Small Computer System Interface
SIMM	Single Inline Memory Module
TFT	thin-film transistor (a type of LCD)
VRAM	video RAM

When unusual abbreviations appear in this developer note, the corresponding terms are spelled out. Standard units of measure and other widely used abbreviations are not spelled out.

CHAPTER 1

Introduction

The Macintosh PowerBook Duo 280 and PowerBook Duo 280c computers are the first of a new generation of PowerBook Duo computers using the powerful Motorola MC68LC040 microprocessor. In addition to all the basic features of the PowerBook Duo computer, the PowerBook Duo 280 and 280c computers also have certain new features described in this developer note.

Note
Except for the MC68LC040 microprocessor and the modifications that accompany it, the features of the PowerBook Duo 280 and 280c computers are similar to those of the PowerBook Duo 250 and PowerBook Duo 270c computers. ◆

Features

The following summary of features gives a general description of the PowerBook Duo 280 and 280c computers. Those computers have several new features that distinguish them from the original Macintosh Duo computers described in *Macintosh Developer Note Number 2*. The new features are listed first and are described later in this developer note.

- new: a Motorola MC68LC040 microprocessor running at 66/33 MHz, described on page 9 (see sidebar)
- new: microprocessor bus translator (MBT), a custom IC that provides an MC68030 bus for compatibility with existing docking modules
- new: active matrix flat panel display, either grayscale or color, with backlight; described beginning on page 11 (the same displays used in the PowerBook Duo 250 and Duo 270c.)
- new: color screen controller (CSC) IC replaces the gray-scale controller; the CSC is described on page 11
- new: a 240 MB or a 320 MB hard disk, described on page 13
- new: an inverter/speaker board compatible with both FSTN (film supertwist nematic) and active matrix TFT color displays, described on page 20
- new: enhanced system ROM, described beginning on page 24
- new: software support for color video, described on page 24
- new: software support for battery recharging, described on page 26

Two clock speeds

The MC68LC040 uses two processor clocks: one for the system bus and another, at twice the speed, for its internal circuits. Thus, an MC68LC040 with a system bus clock of 33 MHz runs its internal processor at 66 MHz.

- main system controller (MSC): a custom IC that controls DRAM, ROM, built-in I/O, sound, and some power-saving features
- Power Manager IC: a custom microcontroller that provides intelligent power management
- 4 MB of RAM on the main logic board
- 8 MB memory expansion card expands RAM capacity up to 12 MB; third-party expansion card expands RAM up to 40 MB
- 1 MB of ROM
- Combo custom IC: includes the Serial Communications Controller (SCC) and Small Computer System Interface (SCSI) controller
- sound circuits: provide 8-bit monaural sound input and output
- integral microphone and speakers
- Express Modem: internal modem/fax card
- keyboard: integral full-function keyboard with trackball
- I/O ports: one 152-pin connector for expansion devices, one mini-DIN 8-pin serial port, and one modem port
- nickel metal hydride battery: 4.5 ampere-hour removable and rechargeable battery

Configurations

The PowerBook Duo 280 and 280c computers are available in two configurations each, as shown in Table 1-1.

Table 1-1 Models and configurations

Model	Display type	Size of RAM	Size of hard disk	Modem included
PowerBook Duo 280	Grayscale, active matrix	4 MB	200 MB	No
PowerBook Duo 280	Grayscale, active matrix	12 MB	200 MB	Yes
PowerBook Duo 280	Grayscale, active matrix	4 MB	240 MB	No
PowerBook Duo 280	Grayscale, active matrix	12 MB	240 MB	Yes
PowerBook Duo 280c	Color, active matrix	4 MB	320 MB	No
PowerBook Duo 280c	Color, active matrix	12 MB	320 MB	Yes

Accessory Devices

In addition to the devices that are included with the PowerBook Duo 280 and 280c computers, the following accessory devices are available:

- The PowerBook Duo 8 MB Memory Expansion Kit expands the RAM in the computers to 12 MB.
- The PowerBook Duo Battery Type III is available separately as an additional or replacement battery.
- The Power Adapter II, the AC adapter that comes with the computers, is also available separately.

All the other accessories for the PowerBook Duo family will work with the PowerBook Duo 280 and 280c computers, including memory expansion, docking units, and modems.

Compatibility Issues

The PowerBook Duo 280 and 280c computers have several new features that distinguish them from the first models in the PowerBook Duo family. This section highlights key areas you should investigate in order to ensure that your hardware and software work properly with these new computers.

Number of Colors

The controller circuitry for the flat panel display includes a 256-entry color lookup table (CLUT) and is compatible with software that uses QuickDraw and the Palette Manager. The controller supports a palette of 32,768 colors. However, due to the nature of color LCD technology, some colors are dithered or exhibit noticeable flicker. Apple has developed a new gamma table for the color displays that minimizes flicker and optimizes the available colors. For the active matrix color display, the effective range of the CLUT is about 4,000 colors.

Note
The color display on the PowerBook Duo 280c is similar to the one on the PowerBook Duo 270c. ◆

See the section “Displays” beginning on page 11 for more information about the internal display hardware and LCD screen.

Power Manager Interface

Starting with the PowerBook Duo 250 and 270c models and continuing with the PowerBook Duo 280 and 280c, the system software includes interface routines for program access to the functions of the Power Manager. Henceforth, developers should not depend on the Power Manager's internal data structures being the same on different PowerBook models. In particular, developers should beware of making any of the following assumptions regarding different PowerBook models:

- assuming that timeout values such as the hard disk spindown time reside at the same locations in parameter RAM
- assuming that the power cycling process works the same way or uses the same parameters
- assuming that direct commands to the Power Manager microcontroller are supported on all models

For more information, see Chapter 4, "Power Manager Interface."

RAM Expansion

In the PowerBook Duo 280 and 280c computers, the RAM expansion connector has an additional signal, RAS(7) on pin 24, that provides addressing for up to 36 MB of expansion RAM. A RAM expansion card for the newer machines also works in the older machines, except that any RAM over 32 MB is ignored.

MacsBug Version 6.2.2

MacsBug version 6.2.2 does work on the PowerBook Duo 280 and 280c computers because it assumes that all MC68040 microprocessors have built-in FPUs. That problem is corrected in newer versions of MacsBug.

The PDS and the MC68030 Bus

Even though the PowerBook Duo 280 and 280c computers have MC68LC040 microprocessors, they support processor-direct slot (PDS) cards designed for the MC68030 bus. The MBT custom IC provides the bus interface, as described in the section "Bus Translator IC" on page 10. The MBT was designed to support the MC68030 bus, but some subtle features are not supported. Here is a summary of the differences.

- Late retry or late bus error operations are not supported. If a device asserts the /BERR signal or /BERR and /HALT without other termination, those signals must conform to the /DSACK timing. If a device asserts /BERR or /BERR and /HALT along with /STERM, those signals must be valid on the same rising edge as /STERM.
- A device should not assert the /HALT signal by itself; doing so does not halt the processor.
- When a device asserts the cache burst acknowledge signal, the device must perform a burst of four longwords, wrapping addresses when necessary, unless it asserts the bus error signal along with the /STERM signal.
- A device must not assert retry after the initial cycle of a burst; the MBT and the MC68LC040 interpret that condition as a bus error and abort the burst.

- The MBT does not support burst cycles of one clock width. The fastest burst cycle allowed consists of 2-2-2-2 clocks.
- The MC68LC040 microprocessor and the MBT do not support the coprocessor interface of the MC68030 microprocessor.
- When the processor has the bus, the MBT drives the function code 0 signal high and the function code 1 signal low, which designates all cycles as data space cycles.
- The cache inhibit out signal on the PDS is low, preventing external caches.
- Signals ECS, OCS, FC2, and DBEN are not provided; they have never been provided on the PDS.
- Interrupt acknowledge cycles are not present on the PDS.
- Breakpoint acknowledge cycles are not present on the PDS.

Future models of the PowerBook Duo will continue to support the 152-pin PDS, but their architecture may change such that some MC68030 modes may not be supported. The following microprocessor modes may not be supported in future models.

- The bus grant signal may not always be asserted within 3.5 clocks of the assertion of the bus request signal even if /RMC is not asserted.
- Ownership of the MC68030 bus extends to all other buses. Relinquish and retry operations will not be supported during dynamic bus sizing or while /RMC is asserted.
- Burst mode from PDS space will not be possible; the computer will never assert the cache burst request signal to the PDS.
- Alternative bus masters will not receive a cache burst acknowledge when they read or write to ROM or DRAM address space.

C H A P T E R 2

Hardware Features

Hardware Features

This chapter describes hardware features of the PowerBook Duo 280 and 280c computers that make them different from the original Macintosh PowerBook Duo computers described in *Macintosh Developer Note Number 2*. The new features include several changes to the main logic board as well as changes to the display, the internal hard disk, the inverter/speaker board, and the clamshell housing.

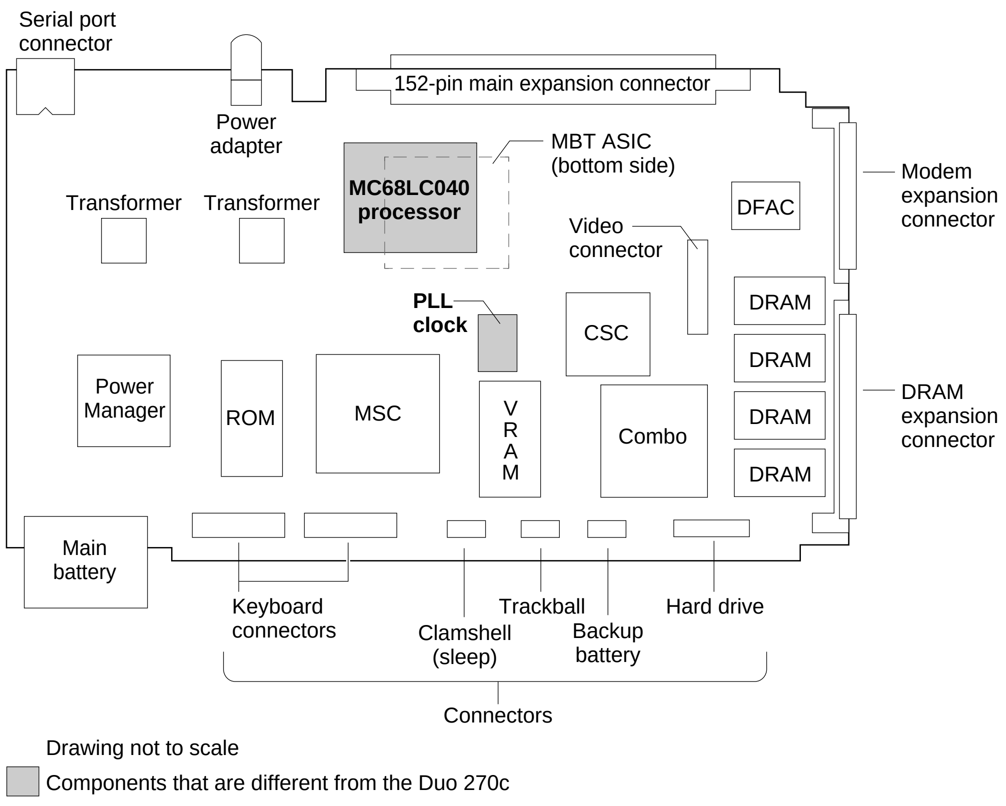
Changes on the Main Logic Board

The outline of the main logic board in the PowerBook Duo 280 and 280c computers is identical to the outline of the main logic board in the original PowerBook Duo models. However, there are several component changes, as described in the following section. Figure 2-1 is a diagram of the main logic board, with the new components shaded.

The changes to the main logic board board are

- replacement of the MC68030 microprocessor with the MC68LC040
- addition of the MBT (microprocessor bus translator)
- replacement of the GSC (grayscale controller) with the CSC (color screen controller)

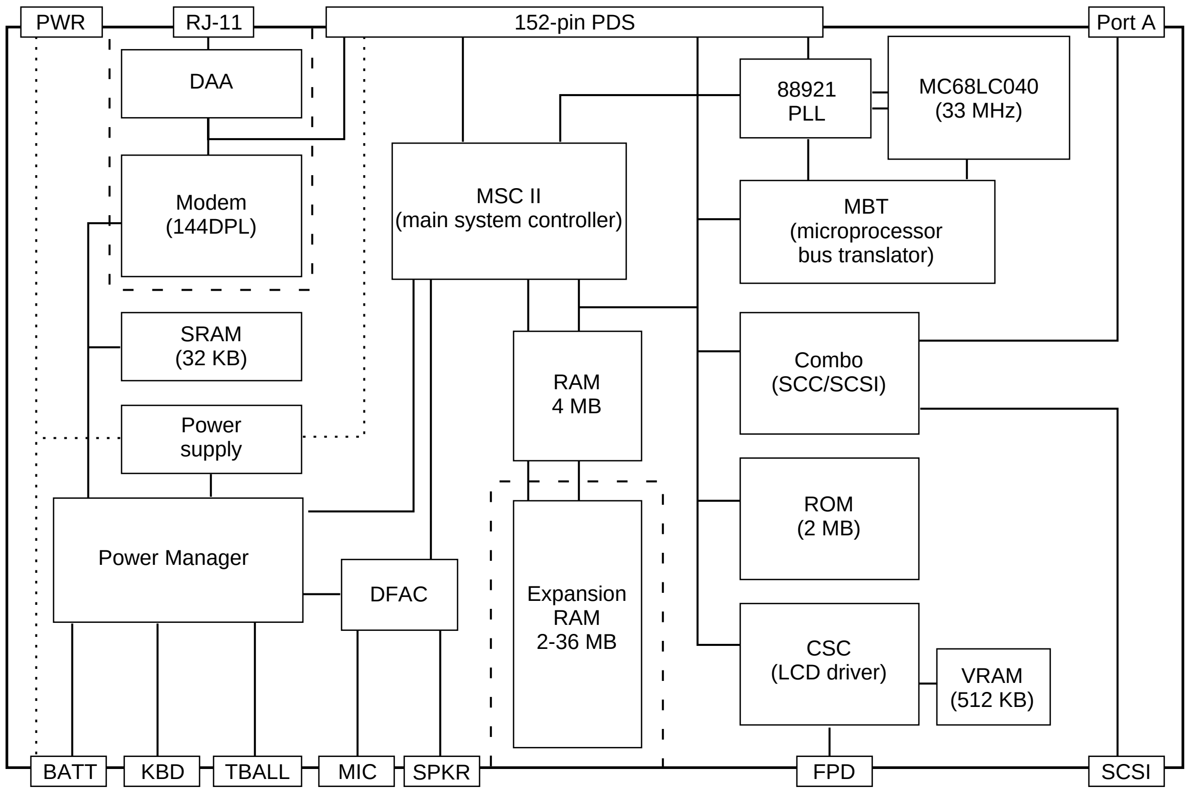
Figure 2-1 Main logic board



- rearrangement of the on-board DRAM
- rearrangement of the on-board VRAM and ROM

Figure 2-2 is a block diagram showing the relationship of the microprocessor and the MBT to the other ICs on the logic board.

Figure 2-2 Block diagram



MC68LC040 Microprocessor

The microprocessor used in the PowerBook Duo 280 and 280c computers is the MC68LC040. The MC68LC040 includes a built-in MMU (memory management unit) that performs the memory-mapping functions.

IMPORTANT
The MC68LC040 does not contain an FPU (floating-point unit). There is no FPU upgrade for the PowerBook Duo 280 and 280c computers and those computers cannot use the external FPU in the Duo Dock and Duo Dock II. ▲

The MC68LC040 microprocessor runs at an internal clock rate that is double its external clock rate. The MC68LC040 in the PowerBook Duo 280 and 280c computers has an external clock rate of 33 MHz and an internal clock rate of 66 MHz.

The PowerBook Duo 280 and Duo 280c do not support external direct-mapped caches and will not make use of the 32KB SRAM cache in the Duo Dock II.

Bus Translator IC

A custom IC called the MBT (microprocessor bus translator) provides the translation necessary for the MC68LC040 to work with devices designed for the MC68030. The MBT translates all MC68040 cycles except acknowledge cycles to MC68030 cycles.

Because the MC68LC040 does not support dynamic bus sizing as the MC68030 did, the MBT generates the multiple MC68030 cycles needed to obtain longword data from byte and word devices. For byte and word reads, the MC68LC040 obtains the data in the appropriate byte lanes based on the low-order address lines. The MBT maps the MC68030 read data from the high byte lanes to the correct byte lanes. For write operations, the MBT performs the byte smearing needed for byte and word operations.

The MBT handles retry requests without involving the MC68LC040. It also performs relinquish and retry operations to prevent stalemate conditions that might arise with alternate bus masters.

The MC68LC040 supports burst mode on both read and write operations. The MBT supports burst-mode write operations to the first 64 MB of RAM because the MSC acknowledges that address space and supports burst-mode writes. On the other hand, the MC68030 did not support burst-mode writes, so devices designed for the MC68030 bus may not handle the /CBREQ signal on write operations. To avoid potential problems with such devices, the MBT blocks burst writes to locations other than RAM.

Burst cycles must wrap to address four longwords. The MC68LC040 has one valid bit per cache line and each cache line contains four longwords. The MC68LC040 looks at the transfer burst inhibit signal at the first assertion of the transfer acknowledge signal to determine whether the device indicates its ability to support a burst. If the transfer burst inhibit signal is negated (high), the MC68LC040 requires the responding device either to wrap the addresses around so that the entire four longwords in the cache line are filled by the single operation or else to terminate early with a bus error.

On the MC68LC040, both the transfer acknowledge signal and the transfer error acknowledge signal must be valid 10 ns before a rising clock edge. To avoid having the MBT add an extra clock to every cycle, the corresponding signals on the MC68030 bus must be valid on the previous rising edge. That means the MBT does not support late retry or bus errors. If a device asserts the /BERR signal with /STERM, or /BERR and /HALT with /STERM, those signals must meet the timing specification for /STERM with respect to the rising clock edge. If a device asserts /BERR or /BERR and /HALT by themselves or in combination with /DSACK, the signals must meet the timing specification for /DSACK with respect to the *falling* clock edge. A device should not assert the /HALT signal by itself; doing so does not terminate the cycle or halt the MC68LC040.

Color Screen Controller IC

A custom IC called the color screen controller (CSC) replaces the grayscale controller (GSC) used in the original PowerBook Duo. Figure 2-1 shows the location of the CSC on the main logic board. The CSC can support either supertwist or active matrix displays, in color or grayscale. The CSC operates in either of two display configurations: 640 by 480 and 640 by 400. It supports the following pixel depths:

- 1 bit per pixel (black-and-white only)
- 2 bits per pixel (4 colors or gray levels)
- 4 bits per pixel (16 colors or gray levels)
- 8 bits per pixel (256 colors or gray levels)
- 16 bits per pixel (32K colors), with 640-by-400-pixel display

DRAM Locations

The main logic board in the PowerBook Duo 280 and 280c computers contains 8 DRAM chips, which make up the 4 MB of onboard DRAM. Four of the DRAM chips are located on the front of the main logic board, and four are located on the back of the board. This is the same arrangement used in the PowerBook Duo 250 and 270 models.

The expansion RAM connector has one additional signal, RAS(7) on pin 24, that allows the PowerBook Duo 280 and Duo 280c computers to address up to 36 MB of expansion memory and gives them a total RAM capability of 40 MB. For information about the expansion RAM connector, see pages 65–72 of *Macintosh Developer Note Number 2*.

VRAM Locations

The 512KB display buffer on the main logic board consists of 2 VRAM chips. One of the VRAM chips is located on the front of the main logic board and one is located on the back of the board.

Displays

The PowerBook Duo 280 computer has a grayscale display and the PowerBook Duo 280c has a color display. Both are liquid-crystal flat panel displays using active-matrix TFT (thin-film transistor) technology, and both have a built-in backlight using a CCFL (cold cathode fluorescent) lamp.

The active matrix technology used in both displays provides a high contrast ratio (60:1) and response time of approximately 60 milliseconds, for performance similar to a CRT video display and with no cursor smearing or cursor submarining.

Types of Displays

Flat-panel displays come in two types: TFT or active matrix, and FSTN or passive matrix. The displays in the PowerBook Duo 280 and 280c computers are the active matrix type.

Active matrix displays, also called thin-film transistor (TFT) displays, have a driving transistor for each individual pixel. The driving transistors give active matrix displays high contrast and fast response time.

Passive matrix refers to a display technology that does not have individual transistors. That technology is also called FSTN, for film supertwist nematic, sometimes shortened to just supertwist.

Both displays normally display black characters on a white background, simulating the appearance of a printed page. The next two sections describe the features of the two types of displays.

Note
The grayscale display in the PowerBook Duo 280 computer is exactly like the display in the PowerBook Duo 250; the color display in the PowerBook Duo 280c computer is exactly like the display in the PowerBook Duo 270c. ◆

Grayscale Display

The grayscale display in the PowerBook Duo 280 is an active matrix liquid crystal display. It meets the same form factors as the basic PowerBook Duo display, so no modifications are required to the computer housing. Each pixel in the grayscale display's 640-by-400-pixel array is controlled by its corresponding transistor, for a total of 256,000 transistors.

Color Display

The color display in the PowerBook Duo 280c computer is an active matrix TFT color display. It is 3.8 millimeters thicker than the grayscale display used in the original PowerBook Duo, requiring the PowerBook Duo 280c computer's case to be thicker than that of the original PowerBook Duo. (See the section "Clamshell Case" on page 21.)

The color display can operate in either of two modes. In 8-bit mode the display has a 640-by-480-pixel area and can display up to 256 different colors at a time. In 16-bit mode the display has a 640-by-400-pixel area and can display thousands of colors.

Note
The number of colors available in 16-bit mode is less than the theoretical maximum due to the limitations in the color LCD technology. Many color values exhibit noticeable flicker. The computer's CLUT omits the unsatisfactory colors, making about 4,000 available. See the Appendix, "Color Lookup Table," for more information. ◆

Smearing and Submarining

Older types of flat-panel dislpays have much slower response times than the active matrix displays used in the PowerBook Duo 280 and 280c. On those older displays, the pixels showing the cursor in one position do not clear quickly when the cursor moves to another position; as a result, the cursor appears smeared out in the direction of motion.

When the cursor is moving rapidly, the pixels may not have time to respond to a newly-drawn cursor before the cursor moves to another position. In that case, the cursor seems to disappear behind the screen, an effect known as submarining.

The displays on the PowerBook Duo 280 and 280c do not have these anomalies.

The user can select either color display mode by means of the Monitors control panel. Because the VRAM is a fixed size, 256K by 16 bits, it can handle only a certain amount of data. When the user selects 16-bit mode, the system software resizes the display area down to 400 lines instead of 480 and centers the display area on the screen, leaving black bands of 40 lines each at the top and bottom of the screen.

240 MB Hard Disk Drive

This section describes the 240 MB hard disk drive. It includes the following information about the drive:

- environmental specifications
- housing requirements
- interface requirements

Note
The physical requirements for the 320 MB drive are the same as for the 240 MB drive. ◆

Environmental Specifications

Table 2-1 on page 14 provides a summary of the environmental specifications for the 240 MB hard disk drive.

Table 2-1 Environmental specifications for 240 MB hard disk drive

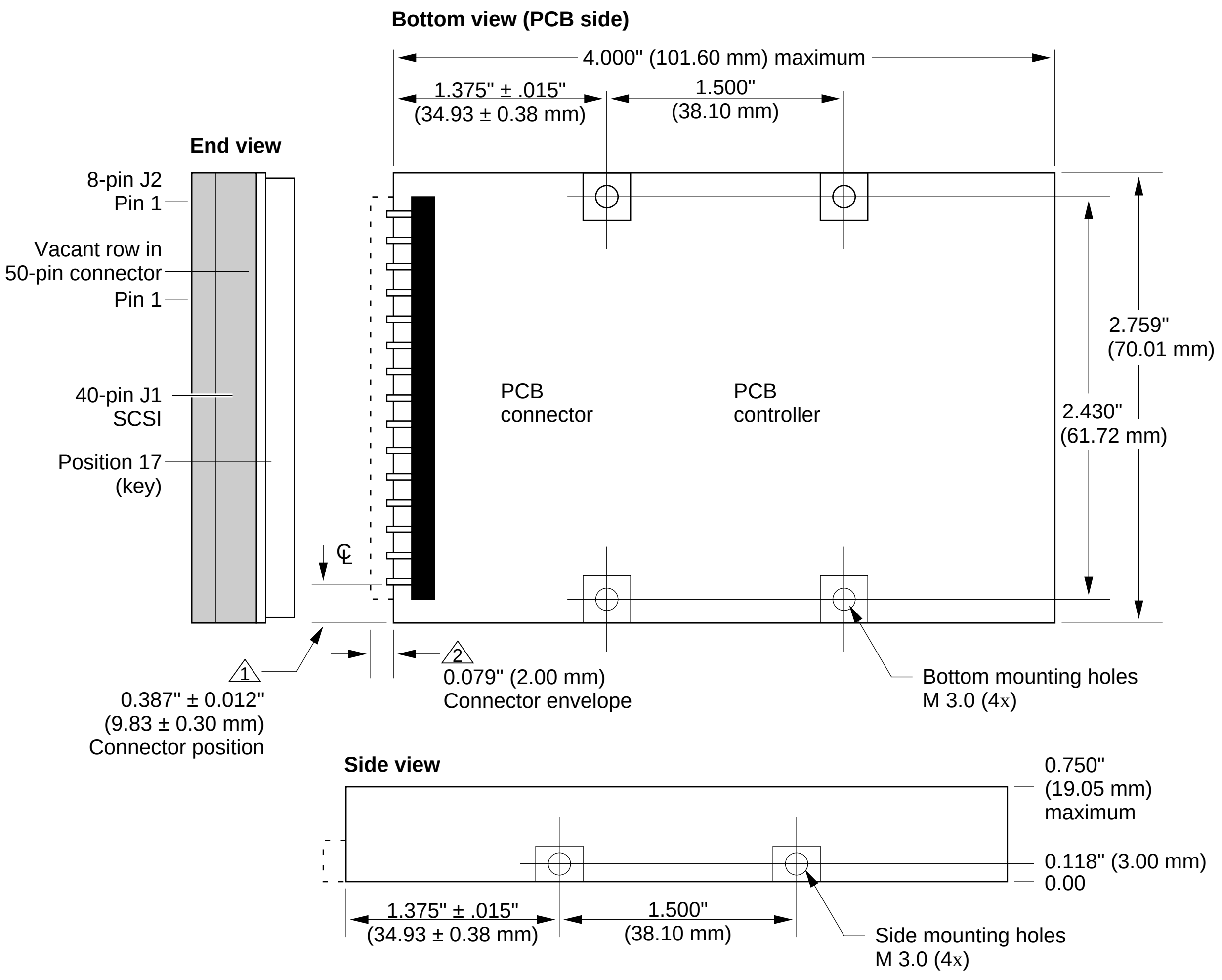
Category	Condition	Specification
Temperature of ambient air inside a low-airflow thermal chamber, noncondensing	Operating limits	5 to 55° C
	Nonoperating and storage	–40 to 60° C
Temperature rate of change	Operating	20° C per hour
	Nonoperating	Below rate causing condensation
Relative humidity	Operating	10 to 90%, noncondensing, maximum wet bulb 20° C
	Nonoperating	Noncondensing, maximum wet bulb 35° C
Altitude	Operating	–200 to 15,000 feet
	Shipping	–200 to 15,000 feet

Installation

The PowerBook Duo 280 and 280c computers accommodate either a 240 MB or a 320 MB hard disk drive. Figure 2-3 shows the dimensions of the 240 MB hard disk. The height of the disk drive is critical, and must not exceed 19 mm (0.75 inches.)

The drive may be installed by means of holes in either the side or the bottom of the disk drive housing. A bracket, shown in Figure 2-4, is attached to the side of the hard disk and holds it in place in the PowerBook Duo 280 and 280c computers.

Figure 2-3 Hard disk drive installation


Bottom view (PCB side)
4.000" (101.60 mm) maximum
1.375" ± .015" (34.93 ± 0.38 mm)
1.500" (38.10 mm)
End view
8-pin J2
Pin 1
Vacant row in 50-pin connector
Pin 1
40-pin J1 SCSI
Position 17 (key)
PCB connector
PCB controller
2.759" (70.01 mm)
2.430" (61.72 mm)
℄
1
0.387" ± 0.012" (9.83 ± 0.30 mm) Connector position
2
0.079" (2.00 mm) Connector envelope
Bottom mounting holes M 3.0 (4x)
Side view
0.750" (19.05 mm) maximum
0.118" (3.00 mm)
0.00
1.375" ± .015" (34.93 ± 0.38 mm)
1.500" (38.10 mm)
Side mounting holes M 3.0 (4x)

Note:

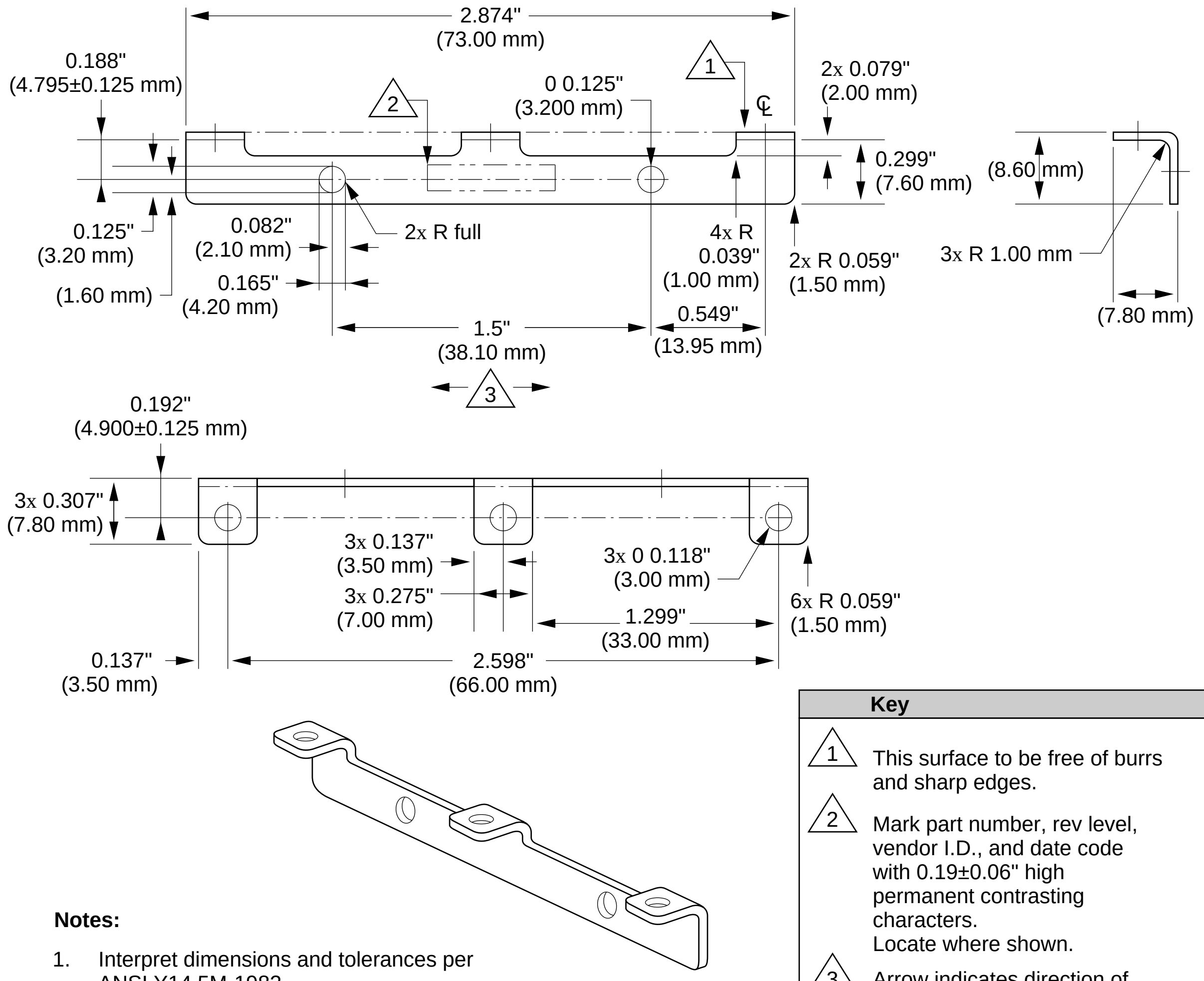
1. Tolerances (unless otherwise noted): .XX = +/–0.25 mm (.XXX = +/– .010 inches).

Key:

⚠1 Connector position from edge of drive to center line of first connector pin (39).

⚠2 Connector envelope does not include flex cable or mating receptacle. Connector pins are to be flush with drive envelope dimensions.

Figure 2-4 Bracket for the hard disk drive



Key

1 This surface to be free of burrs and sharp edges.

2 Mark part number, rev level, vendor I.D., and date code with 0.19±0.06" high permanent contrasting characters.
Locate where shown.

3 Arrow indicates direction of material grain.

Notes:

1. Interpret dimensions and tolerances per ANSI Y14.5M-1982
2. Material: CRS 1010-1020, 1.00 ± 0.05 mm (.0394 ± .0020 mm) thick.
3. Finish: Zinc preplate per Mil.Spec. QQ-Z-325a Class 3 (0.00020) type II.
4. Maximum burr allowance is 15% of material thickness.
5. Tooling required to make this part to be property of Apple Computer, Inc. and shall be permanently marked with Apple's name and appropriate part number.
6. All dimensions apply after finish.

Hard Disk Interface

This section describes the interface requirements for the 240 MB hard disk drive and provides specifications and signal assignments for the SCSI connector.

The interface to the hard disk is an ANSC X3T9.2 SCSI interface. It implements the Apple SCSI command protocol and diagnostic command set. Buffer size supports a 1:1 interleave. The drive supports the SCSI asynchronous information transfer. The transfer rate is 1.5 MB per second (minimum). An embedded controller provides error recovery algorithms, which include error check and correction (ECC), seek retry, head offset (for open-loop systems), and defect management. The SCSI interface provides a SCSI ID that can be detected by the hardware.

SCSI Connector

The disk drive connector comprises two segments, as shown in Figure 2-5. The first segment has 40 pins, arranged in two rows. It transfers SCSI signals between the CPU and the hard disk drive, and it also supplies power to the drive. The second segment of the connector has 8 pins and provides the SCSI ID encoding.

Figure 2-5 Connector for the hard disk drive

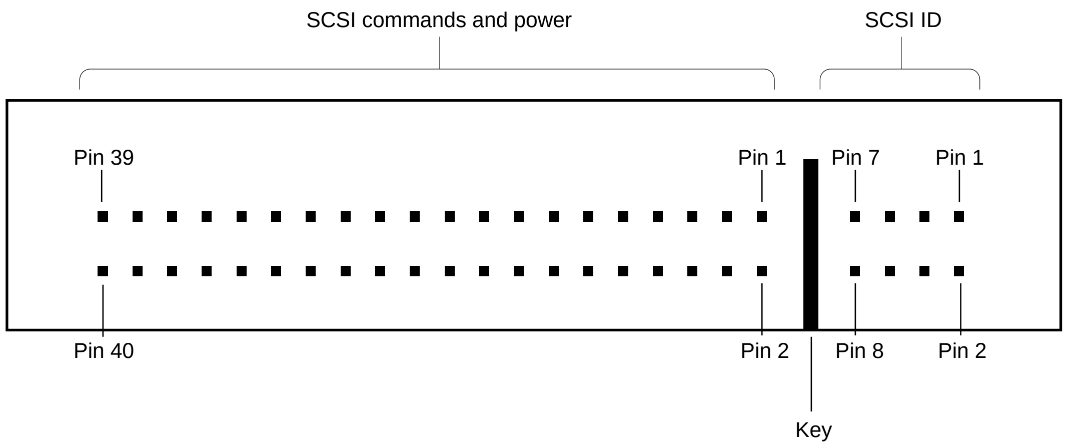


Table 2-2 lists the interface signals for the 40-pin segment of the connector.

Table 2-2 Signal assignments on the hard disk drive connector

Pin	Signal name	Description
1, 2	+5V LOGIC	+5 V power supply
3, 4	LOGIC RET	+5 V return
5, 7, 9, 11, 13, 15, 19, 21, 23, 27, 31, 35	GND	Ground
6	/DB0	Data bus bit 0
8	/DB1	Data bus bit 1
10	/DB2	Data bus bit 2
12	/DB3	Data bus bit 3
14	/DB4	Data bus bit 4
16	/DB5	Data bus bit 5
17	KEY	Not connected; used as connector key
18	/DB6	Data bus bit 6
20	/DB7	Data bus bit 7
22	/PARITY	Data bus parity
24	TERM PWR	Terminator power; pulls up termination resistors for all signal lines
25	/ATN	Attention indicator
26	/BSY	Busy signal
28	/ACK	Acknowledge (handshake signal); asserted in response to a request for access (/REQ)
29	/RST	SCSI bus reset
30	/MSG	Message phase
32	/SEL	SCSI select
33	/I/O	Controls the direction of data movement: when this signal is low, data is output from the disk drive; when it is high, data is input
34	/C/D	Indicates whether data or control signals are on the SCSI bus: when this signal is low, data is on the bus; when it is high, control signals are on the bus
36	/REQ	Access request; the CPU asserts this signal to request access to the hard disk
37, 38	MOTOR RET	Return for +5 V power supply for motor
39, 40	+5V MOTOR	+5 V power supply for the motor

Table 2-3 lists and describes the interface signals for the 8-pin segment of the connector.

Table 2-3 Signal assignments on the SCSI ID connector

Pin	Signal name	Description
1, 2, 3, 4, 8	Unused	These pins are not used and are not connected electrically to the CPU
5	/ID1	SCSI ID 1
6	/ID2	SCSI ID 2
7	/ID4	SCSI ID 4

The internal hard disk is assigned a SCSI ID number (0-7). Pins 5-7 on the SCSI connector are encoded and allow the CPU to select the appropriate device. Table 2-4 shows how the SCSI ID signals are encoded.

Table 2-4 SCSI ID encoding

SCSI ID	ID1: pin 5	ID2: pin 6	ID4: pin 7
0	High	High	High
1	Low	High	High
2	High	Low	High
3	Low	Low	High
4	High	High	Low
5	Low	High	Low
6	High	Low	Low
7	Low	Low	Low

Terminator

The hard disk has 1000-ohm termination resistors for all I/O signal lines. The lines are pulled up through the resistors to the terminator power signal.

Power Requirements

Power drawn by the hard disk signals in each operating mode must be less than or equal to the values shown in Table 2-5. All measurements are under nominal environmental and voltage conditions. The limits include 1000-ohm pull-up resistors on all signal lines.

Table 2-5 Hard disk power requirements

	Current (amperes)	
Mode	Mean	Maximum
Startup*	—	1.30
Random operation†	0.50	0.60
Idle	0.30	0.35

* Startup values are peak values during response time of power on to power ready.
† Random operation values are RMS values with a 40 percent random seek, 40 percent write/read (1 write in 10 reads), and 20 percent idle mode.

Inverter/Speaker Board

The inverter/speaker board is located in the clamshell housing, directly under the display. It is the interface between the main logic board and the display and performs the following basic functions:

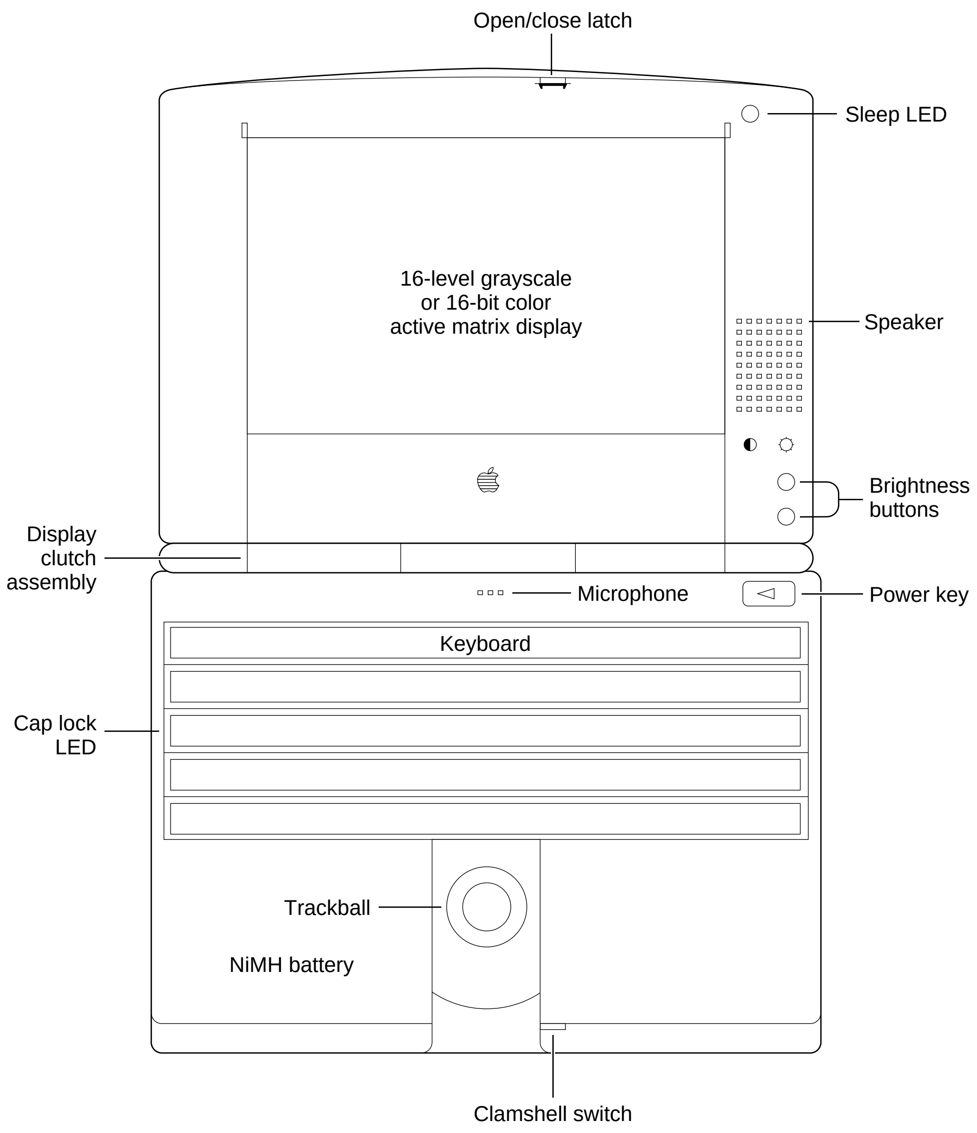
- It converts the DC power supplied by the computer's battery to the AC power required to drive the cold cathode fluorescent lamp (CCFL) that provides the backlighting for the active-matrix LCD display. Typical CCFL drive power is 400 VRMS (volts root mean square) at 3 mA, with maximum output of 2000 V peak to peak at 6 mA.
- It provides pass-through circuitry, both for the data and timing signals from the CSC to the LCD display, and for the +5 V from the power supply.
- It converts the DC voltage supplied by the computer's battery to the voltage level required for the LCD bias, which is typically +5 V at 200 mA.

▲ **WARNING**
You should not open or modify any of the circuitry associated with the inverter/speaker board. The flat-panel display is assembled into the clamshell housing in a clean room environment. Opening up the equipment in any other environment could cause damage to the unit. The high voltage on the inverter/speaker board may pose a risk to someone handling the board. The display is also susceptible to damage from electrostatic discharge. ▲

Clamshell Case

The PowerBook Duo 280 and 280c computers are housed in a clamshell case. Because the display in those computers is 3.8 millimeters (0.149 inches) thicker than the one in the original PowerBook Duo, the case is also thicker than the original PowerBook Duo case. When the case is closed, it measures 203.2 millimeters by 274.32 millimeters (8 inches by 10.8 inches) and is 36.8 millimeters (1.449 inches) deep. Figure 2-6 shows a view of an open clamshell case.

Figure 2-6 The computer in open position



IMPORTANT

Because the clamshell case is slightly deeper than the case of the original PowerBook Duo, you cannot use a standard Duo Dock with the PowerBook Duo 280 and 280c computers. The slot in the Duo Dock is not deep enough to accommodate the computer’s case. There are two solutions. You may upgrade the top shell of an existing Duo Dock to enlarge the slot, or you may purchase the new Duo Dock II, which can accommodate the deeper case without modification. ▲

C H A P T E R 3

Software Features

Software Features

This chapter describes the new ROM and system software features of the PowerBook Duo 280 and 280c computers. It also tells how to add modules to the control strip.

ROM Software

The ROM software in the PowerBook Duo 280 and 280c computers is based on the ROM used in previous PowerBook computers, with enhancements to support the many new features of this computer. Some of the features this ROM was designed to support include the following:

- MC68LC040 microprocessor
- built-in color and grayscale displays
- refresh operation for up to 40 MB of DRAM
- extended power management capabilities
- improved support for the AppleTalk network

The ROM also supports several docking station features, such as an external cache, and enhancements to the docking process.

The ROM version number in the PowerBook Duo 280 and 280c computers is $67C and the CPU ID bits are $A55A 1000. The box flag for the PowerBook Duo 280 is 96; for the PowerBook Duo 280c it is 97. The corresponding `gestaltMachineID` values are 102 and 103 respectively. Notice that the CPU ID bits are the same for both models; the box flag and `gestaltMachineID` values are determined by the display types.

MC68LC040 Microprocessor

The MC68LC040 microprocessor used in the PowerBook Duo 280 and 280c computers differs from the MC68030 used in earlier PowerBook Duo models in several important ways. For example, there are differences in the sizes of its caches and in the configurations of its control registers. The differences make it necessary to use a different power cycling scheme with the MC68LC040. The MC68LC040 also requires new versions of the MMU tables that control the way the system addresses memory and I/O space in 24-bit and 32-bit addressing modes.

Display Driver

Support for both color and grayscale displays is provided by a display driver for the CSC (color screen controller). The display driver resides in the slot 0 declaration ROM, which is part of the system ROM on the main logic board. The driver supports grayscale, 8-bit color, and 16-bit color.

Grayscale Display

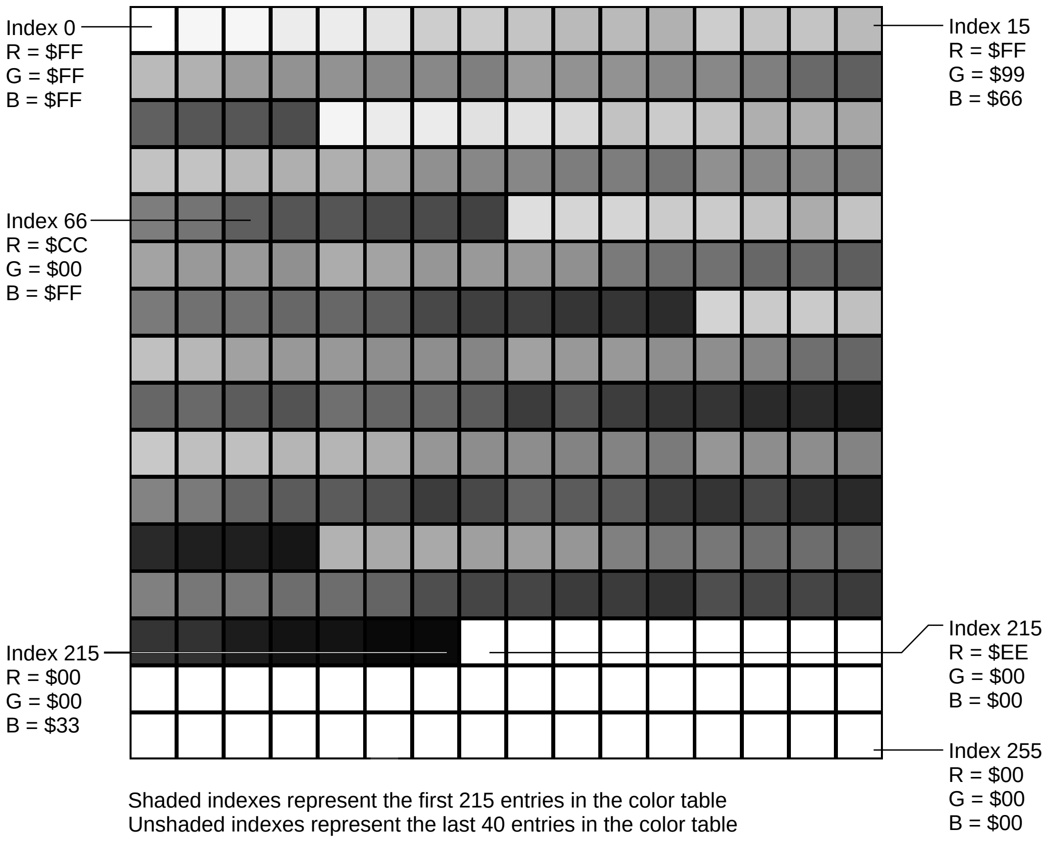
The grayscale display in the PowerBook Duo 280 and 280c computers has the same features as the display in the PowerBook Duo 250, but the driver is different because the PowerBook Duo 280 and 280c computers use the CSC instead of the GSC (grayscale controller).

8-Bit Color

The 8-bit-per-pixel configuration produces 256 colors. Figure 3-1 shows the color lookup table with values for sample indexes (pixels). You are advised to select colors from the first 215 entries in the color table. The values of the entries are based upon combinations of $00, $33, $66, $99, $CC, and $FF.

The last 40 entries in the table are assigned to shades of pure red, green, blue, and gray, and are based upon combinations of $00, $11, $22, $44, $55, $77, $88, $AA, $BB, and $EE. These entries are ramped and dithered to produce the various shades. Dithering, which may be implemented spatially or temporally, mixes primary colors to produce the effect of a range of different shades.

Figure 3-1 Color lookup table



The Appendix, “Color Lookup Table,” includes a table showing the color values that correspond to each of the 256 index numbers.

16-Bit Color

When the user selects 16-bit color (“Thousands”) from the Monitors control panel, the image area on the LCD display shrinks slightly, to 640 by 400 pixels, and narrow black bands appear at the top and bottom of the screen. In 16-bit color mode, the color values are stored as 5-bit R, G, and B components (only 15 bits are used). The CLUT is used as a gamma table to optimize the color values for best appearance on the color LCD.

Support for Extended DRAM

The main system controller (MSC) installed on the PowerBook Duo 280 and 280c computers’ main logic board supports up to 40 MB of self-refreshing DRAM. The computer’s hardware provides decoding for up to 32 MB of DRAM.

Note
The PowerBook Duo 280 and 280c computers support both 24-bit addressing and 32-bit addressing, but the ROM software defaults to 32-bit addressing mode. Whenever the system detects invalid PRAM, it initializes the PRAM byte to select 32-bit addressing. Making 32-bit addressing the default avoids possible confusion about memory size in a 12 MB configuration. ◆

Extended Power Management Capabilities

The Power Manager code, which resides in the ROM, allows you to modify the system time without affecting the battery-charging algorithm. Firmware features added for the PowerBook Duo 280 and 280c computers also support a sleep LED, version dependent code, support for a 4 A or 5 A battery, and battery conditioning.

The Power Manager section of the ROM provides a standard set of routines that developers may use (see Chapter 4, “Power Manager Interface”). All other routines are private.

Network Support

With the PowerBook Duo 280 and 280c computers, the latest version of AppleTalk is included in both the computer’s ROM and on the System disk. The new version of AppleTalk is more efficient in remembering network settings when the computer is being docked or undocked. AppleTalk resources have been removed from the System Enabler file. The Installer application installs AppleTalk files directly into the System file when needed.

System Software

The PowerBook Duo 280 and 280c computers are shipped with system software version 7.1.1. The system software includes a new System Enabler file, which is required for the PowerBook Duo 280 and 280c computers.

Identifying the PowerBook Duo 280 and 280c Computers

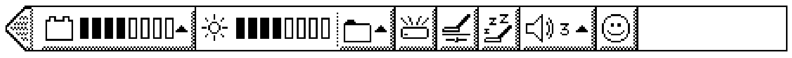
To identify the Macintosh model it is running on, your application should use the Gestalt Manager routines described in *Inside Macintosh: Overview*.

The Gestalt Manager returns a `gestaltMachineType` value of 102 for the PowerBook Duo 280 and a value of 103 for the PowerBook Duo 280c. Those values can be used to obtain the machine name strings as described in *Inside Macintosh: Overview*.

Control Strip

The desktop on the PowerBook Duo 280 and 280c computers has a new status and control element called the control strip. It is a strip of graphics with small button controls and indicators in the form of various icons. Figure 3-2 shows the control strip.

Figure 3-2 Control strip



The control strip is a system extension (INIT) that provides the operating environment for control strip modules. It runs on any Macintosh computer with System 7.0 or later.

The control strip is implemented in a private layer that appears in front of the windows in all the application layers so that the windows will not obscure it. The user can move the window for the control strip to any location on the display as long as the right or left edge of the strip is attached to the right or left edge of the display.

The control strip has a tab on its unattached end. The user can drag the tab to adjust the length of the strip or hold down the Option key and drag the tab to move the strip to a new position. The user can hide the control strip, except for the tab, by clicking the tab. Clicking the tab when the control strip is hidden makes the control strip visible again. To make the control strip disappear completely, the user can click the Hide button in the control strip control panel, described on page 31.

The different parts of the control strip either display status information or act as buttons. When the user clicks a button, it is highlighted; some buttons also display additional elements such as pop-up menus.

By holding down the Option key and clicking a display area, the user can drag the display area to another position in the control strip. After the user rearranges the parts of the control strip, the new arrangment is saved when the computer is shut down and restarted.

The control strip software provides a standard screen location for a collection of individual modules that provide status and control functions. The control strip functions include

- AppleTalk Switch: shows whether AppleTalk is on or off and lets the user turn AppleTalk on or off without having to open the Chooser.
- Battery Monitor: displays the status of the battery or batteries.
- File Sharing: displays the state of file sharing (on, off, or users connected), lets the user turn file sharing on or off, and lets the user open the Sharing Setup control panel.
- HD Spin Down: shows whether the internal hard disk is on or off; lets the user turn off the hard disk.
- Power Settings: lets the user select between maximum conservation or maximum performance without opening the PowerBook control panel; also lets the user open the PowerBook control panel.
- Sleep Now: puts the computer into sleep mode.
- Sound Volume: lets the user select the sound volume.
- Video Mirroring: lets the user turn video mirroring on or off if an external monitor is connected.

Note
Several of the functions of the control strip were implemented in separate control panels on earlier PowerBook models. ◆

Developers can add modules to the control strip. For information, see the section “Adding Control Strip Modules” beginning on page 31.

Control Panels

Several control panels are new or revised for the PowerBook Duo 280 and 280c computers. The following sections describe the new control panels.

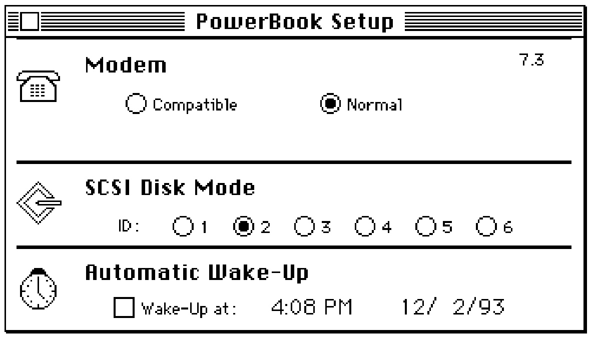
PowerBook Setup Control Panel

The PowerBook Setup control panel controls the setup functions for the modem port configuration, SCSI disk mode, and automatic wakeup.

The PowerBook Setup control panel in the PowerBook Duo 280 and 280c computers is a modified version of the PowerBook control panel that shipped with System 7.1. It is essentially the same control panel, but with all power conservation features removed.

The PowerBook Setup control panel has been further modified to accommodate different modem configurations. The modem controls in the PowerBook Setup control panel distinguish between the Express Modem and other modems. Figure 3-3 shows the PowerBook Setup control panel with modem controls.

Figure 3-3 PowerBook Setup control panel



PowerBook Setup

Modem 7.3

Compatible [✓ Normal]

SCSI Disk Mode

ID: 1 [✓ 2] 3 4 5 6

Automatic Wake-Up

☐ Wake-Up at: 4:08 PM 12/ 2/93

The PowerBook Setup control panel determines what type of modem is installed and chooses the appropriate control titles. If a modem other than the Express Modem is present, “Compatible” is changed to “Internal Modem” and “Normal” to “External Modem.” The functions associated with the radio buttons have not changed. However, the titles are different to remove the confusion generated by the incorrect use of internal/ external modem when an Express Modem is installed.

If you have a third-party internal modem installed, the control titles will be Internal Modem and External Modem, and they will behave as expected, with Internal Modem selecting the modem installed in the system. These titles are the same as those used in System 7.1.

IMPORTANT

If you select Compatible, port A (Printer/Modem) is not available for serial connections. However, AppleTalk is still available. ▲

PowerBook Control Panel

The PowerBook control panel includes several controls that allow the user to balance performance against battery conservation. The PowerBook control panel contains the battery conservation controls, including sleep and processor cycling. It controls backlight dimming, and it can also automatically change the Power Manager configuration based on the machine’s power source.

Note

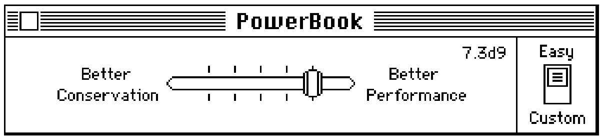
The PowerBook Duo 280 and 280c computers do not provide the Economode reduced speed feature found on earlier PowerBook Duo models. ◆

Information about power management is stored in the Preferences file in the System Folder. When the system is booted, the file is read and the contents are stored permanently in memory.

Custom and Easy Controls for Battery Conservation

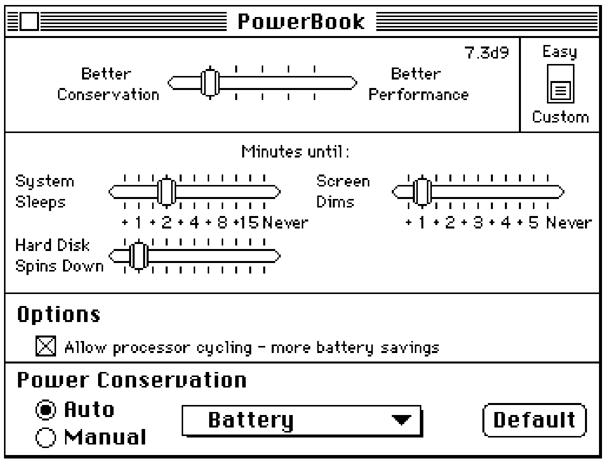
The PowerBook control panel has two modes of operation. The first time you open the PowerBook control panel, you see one simple slider switch, as shown in Figure 3-5. The user may adjust the slider, as required, to improve battery conservation or system performance, or leave it in the default position.

Figure 3-4 PowerBook control panel in easy mode



When you click the Easy/Custom box in the control panel, you gain access to three additional sliders, as shown in Figure 3-5. They are System Sleeps, Hard Disk Spins Down, and Backlight Dims. If you move any of those sliders, the change will be reflected in the Better Conservation/Better Performance slider, alerting you as to whether the change improves performance or provides better conservation. The controls revert to the single slider when you toggle the Easy/Custom control again. The PowerBook control panel preserves the state in which the controls were left the last time it was used.

Figure 3-5 PowerBook control panel in custom mode



☒ Allow processor cycling – more battery savings

[✓ Auto] Manual

Battery

Power Conservation

There are two modes of battery conservation. One is used when the computer is plugged into AC main power, and the other is used when the computer is running on its battery. A Time Manager task installed by system extension in the PowerBook Duo file can automatically change the power management settings based on the machine environment.

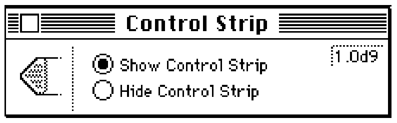
When the PowerBook Duo 280 and 280c computers are shipped, the Power Conservation part of the PowerBook control panel is set with the default settings for Battery and Auto enabled, as shown in Figure 3-5. While the control panel is in easy mode, the user cannot edit those settings. When the PowerBook control panel is in custom mode, the user can enable or disable automatic conservation (Auto), select either Battery or Power Adapter as the power source, or revert to the default settings. The changes the user makes in the PowerBook panel are stored in the Preferences file.

At certain times, the PowerBook control panel compares the current Power Manager settings with both sets of parameters. It does this during open and activate events, when you switch from Manual to Auto power conservation, and when you switch to custom mode. If a match is found, the name of the matching set is displayed in the pop-up menu, and both the menu and the default button are active. If no match is found, No Set Selected is shown in the pop-up menu, and both menu and default buttons are inactive. Once a valid set is selected, No Set Selected is removed from the pop-up menu.

Control Strip Control Panel

Figure 3-6 shows the Control Strip control panel. The user can hide or show the control strip by clicking the corresponding button in the control panel.

Figure 3-6 Control Strip control panel



Adding Control Strip Modules

The control strip is implemented in software as a shell with individual control and status modules added. The control strip software draws the strip that acts as the background for the individual modules. Each module is responsible for drawing the icons and other objects that make up its user interface.

Contents of Module Files

The only required resource in a module file is a resource containing the code necessary for the module to interact with the control strip. A module file may contain more than one code resource if it is to provide multifunctional support. In that case, each module in the file is loaded and initialized separately and treated as an independent entity.

If a file contains only a single code resource, the resource may be unnamed, and the module will be referenced by its filename. If more than one module is contained within a module file, each module is required to have a unique name describing its functionality.

All other resources in a module file are optional, but there are several that are recommended in order to support a custom icon and version information. The recommended resources are

- `'BNDL'`
- `'FREF'`
- `'ICN#'`, `'icl4'`, `'icl8'`, `'ics#'`, `'ics4'`, `'ics8'`
- signature resource (same type as file's creator)
- `'vers'`, ID=1

Developers should confine their resources to the range 256–32767.

Module Interface

Each module's interface to the control strip consists of a code resource of type `'sdev'`. This code is responsible for performing all the functions required by the control strip as well as any functions that are custom to the module itself. The module's entry point is at the beginning of the resource and is defined as

```
pascal long ControlStripModule(long message,
                               long params,
                               Rect *statusRect,
                               GrafPtr statusPort);
```

Interactions between a module and the control strip are managed by passing messages to the module to tell it what to do or to obtain information about the module and its capabilities. Each module is required to observe Pascal register saving conventions. A module may trash registers D0, D1, D2, A0, and A1, but must preserve all other registers across its call.

Field descriptions

`message`	A message number, from the list in the section “Control Strip Module Messages”, that tells the module what action to perform.
`params`	This is the result returned by the initialize call to the module. This would typically be the handle to the module’s private variables since modules can’t have global variables. It will be passed to the module on all subsequent calls.
`statusRect`	A pointer to a rectangle defining the area that a module may draw within.
`statusPort`	A pointer to the control strip’s graphics port. This will be either a color or black-and-white graphics port depending on which PowerBook model the control strip is running on.

The result value returned by the module will vary depending on the message sent to it. Results for each message are described in the sections on the individual messages.

Module Reentrancy

Any module that makes calls to routines such as `GetNextEvent`, `ModalDialog` or `pop-upMenuSelect` should assume that it could be called reentrantly; that is, the module could be called again while the initial call is still in progress. Situations to avoid are such things as reusing a single parameter block for multiple calls, or indiscriminately locking and unlocking your global variables around the module’s invocation.

Instead of using a single parameter block, it’s better, if possible, to allocate the parameter block on the stack. In the case of asynchronous calls, using the stack could cause problems; in that case, preventing the block’s reuse should be sufficient.

If you need to lock and unlock your global variables, it’s better to use `HGetState` and `HLock` at the beginning of the call, and `HSetState` at the end, so that the state is restored to what it was on entry.

Control Strip Module Reference

Control strip modules interact with the control strip software in three ways: by accepting messages, by calling utility routines, and by calling `Gestalt` seletors. The next three sections describe each of those interactions.

Control Strip Module Messages

All control strip modules must respond to messages from the control strip. The following messages have been defined:

Message name	Message number	Description
`sdevInitModule`	0	Initialize the module
`sdevCloseModule`	1	Clean up before being closed
`sdevFeatures`	2	Return the feature bits
`sdevGetDisplayWidth`	3	Return the width of the module’s display
`sdevPeriodicTickle`	4	Periodic tickle when nothing else is happening
`sdevDrawStatus`	5	Update the interface in the control strip
`sdevMouseClick`	6	User has clicked on the module’s display area
`sdevSaveSettings`	7	Save any changed settings in thee module’s preferences file
`sdevShowBalloonHelp`	8	Display a help balloon, if the module has one

sdevInitModule

The `sdevInitModule` message is the first message sent to a module after the module has been loaded from its file. Initialization allows the module to initialize its variables and to determine whether it can run on a particular machine: for example, if the module’s function is to display battery information it can run only on a PowerBook.

The module needs to load and detach any resources in the module’s resource file that will be used, because the resource file will not be kept permanently open. What that means is that your code can't use `GetResource()` or the like to retrieve the handle to one of the module's resources on a subsequent call. Typically you would allocate space in your global variables for handles to those detached resources.

The `sdevInitModule` message returns a result depending on its success at installing itself. A positive result (≥0) indicates successful installation. This result value will be passed to the module on all subsequent calls. A negative result indicates an error condition, and installation of the module is aborted by the control strip software. The module will not receive a close message when installation has been aborted.

sdevCloseModule

The `sDevCloseModule` message is sent to a module when it should be closed. Typically the module itself will decide when this ought to happen. When the module receives this message, it should dispose of all the detached resources it loaded as well as its global storage. No result is expected.

sdevFeatures

The `sdevFeatures` message queries the module for the features it supports. It returns as its result a bitmap consisting of 1 bits for supported features and 0 bits for unsupported features. All undefined bits are reserved by Apple for future features, and must be set to 0. The bits are defined as

`sdevWantMouseClicks`	0	If this bit is set, the control strip will notify the module of mouse down events. If this bit is not set, the control strip assumes that the module only displays status information with no user interaction.
`sdevDontAutoTrack`	1	If this bit is set, the control strip highlights the module's display and then calls the module to perform mouse tracking; this bit is usually set when, for example, a module has a pop-up menu associated with it. If this bit is cleared, the control strip tracks the cursor until the mouse button is released, then sends an `sdevMouseClick` message to the module to notify it that there was a mouse-down event.
`sdevHasCustomHelp`	2	If this bit is set, the module is responsible for displaying its own help messages, which can be customized depending on its current state. If the bit is cleared, the control strip will display a generic help message when the cursor passes over the module's display area and Balloon Help is on.
`sdevKeepModuleLocked`	3	If this bit is set, the module's code will be kept locked in the heap. This bit should be set only if the module is passing the address of one of its routines to the outside world (for example, installing itself in a queue).

sdevGetDisplayWidth

The `sdevGetDisplayWidth` message is sent to a module to determine how much horizontal space (in pixels) its display currently requires on the control strip. The module should return the number of pixels as its result. The returned width should not be the maximum width it requires for any configuration, but should reflect how much space it currently requires, because it’s possible for a module to request that its display be resized.

IMPORTANT
You should be conservative in your use of control strip display space, which is limited. Because several modules could be requesting space, it’s possible that your module could be shoved off the end. ▲

sdevPeriodicTickle

The `sdevPeriodicTickle` message is passed to the module periodically to allow the module to update its display due to changes in its state. You should not assume any minimum or maximum interval between tickles. The module should return, as its result, some bits that signal requests for actions from the control strip software. All undefined bits in the result are reserved for future use by Apple and must be set to 0. The bits are defined as

`sdevResizeDisplay`	0	If this bit is set, the module needs to resize its display. The control strip sends a `sdevGetDisplayWidth` message to the module and then update the control strip on the display.
`sdevNeedToSave`	1	If this bit is set, the module needs to save changed settings to disk. The control strip software will mark the request but may defer the actual save operation to a better time (for example, when the hard disk is spinning).
`sdevHelpStateChange`	2	If this bit is set, the module’s state has changed so it needs to update its help message. If a help balloon is being displayed for this module, the control strip software will remove it and put up a new help balloon for the current state.
`sdevCloseNow`	3	If this bit is set, the module is requesting to be closed. The control strip software will call the module to save its settings, then call it again to close itself.

sdevDrawStatus

The `sdevDrawStatus` message indicates that the module has to redraw its display to reflect the most recent state. This message is typically sent when the user clicks on the module's display area, when any of the module’s displays is resized, or when the control strip itself needs to be updated, perhaps in response to a screen saver deactivation.

The `statusRect` parameter points to a rectangle bounding the module's display area, in local coordinates. All drawing done by a module within the bounds of the control strip must be limited to the module's display rectangle. The graphics port's `clipRgn` will be set to the visible portion of this rectangle so you can draw all the elements in the display. If you need to change the `clipRgn`, you should observe the initial `clipRgn` to avoid drawing over other items in the control strip.

sdevMouseClick

When the user clicks in a module's display area, the control strip software calls the module with the `sdevMouseClick` message if the `sdevWantMouseClicks` bit is set in the module's features.

If the `sdevDontAutoTrack` bit is also set, the control strip draws the module's display in its highlighted state and then sends the `sdevMouseClick` message to the module. If the `sdevDontAutoTrack` bit is not set, the control strip software tracks the cursor until the mouse button is released. If the cursor is still within the module's display area, the control strip software sends the `sdevMouseClick` message to notify the module that a click occurred. In either case, the module can then perform the appropriate function in response to a mouse-down event.

This message returns the same result as the `sdevPeriodicTickle` message.

sdevSaveSettings

The `sdevSaveSettings` message is passed to the module when the control strip software has determined that it's a good time to save configuration information to the disk. This message will be sent only if the module had previously set the `sdevNeedToSave` bit in the result of a `sdevPeriodicTickle` or `sdevMouseClick` message. The call returns an error code (File Manager, Resource Manager, or the like) indicating the success of the save operation. The control strip software will continue to send this message to the module until the module returns a result of 0, indicating a successful save.

sdevShowBalloonHelp

The control strip software calls the module with the `sdevShowBalloonHelp` message if Balloon Help is turned on, the module has previously set the `sdevHasCustomHelp` bit in its features, and the cursor is over the module's display area. The module should then call the Help Manager to display a help balloon describing the current state of the module. The module should return a value of 0 if it's successful or an appropriate error result if not.

Utility Routines

The control strip software provides a set of utility routines that are available to control strip modules. The utility routines are provided to promote a consistent user interface within the control strip and to reduce the amount of duplicated code that each module would have to include to support common functions.

The utility routines are called through a selector-based trap, `_ControlStripDispatch` ($AAF2). If an unimplemented routine is called, it will return `paramErr` as the result.

IMPORTANT
These routines should not be called at interrupt time because they all move memory. ▲

SBIsControlStripVisible

You can use the `SBIsControlStripVisible` routine to find out whether the control strip is visible.

```
pascal Boolean SBIsControlStripVisible();
```

The `SBIsControlStripVisible` routine returns a Boolean value indicating whether or not the control strip is currently visible. It returns a value of `true` if the control strip is visible, or a value of `false` if it's hidden.

It is possible for this call to return a value of `true` even when the control strip is not visible. That happens whenever the control strip is not accessible in the current environment. As soon as that situation changes, the control strip becomes visible again and the returned value correctly reflects the actual state.

SBShowHideControlStrip

You can use the `SBShowHideControlStrip` routine to show or hide the control strip.

```
pascal void SBShowHideControlStrip(Boolean showIt);
```

The `SBShowHideControlStrip` routine determines the visibility state for the control strip based on the value of the `showIt` parameter. Passing a value of `true` makes the control strip visible, and passing a value of `false` hides it. Modules shouldn't typically need to call this routine, but it's provided as a means for other software to hide the control strip when it might get in the way.

Calling `SBShowHideControlStrip` with a `showIt` value of `true` may or may not show the control strip, depending on the current environment: if the control strip is not accessible, it does not become visible. If a `showIt` value of `true` is passed to this routine, then when the environment changes, the control strip will become visible.

SBSafeToAccessStartupDisk

You can use the `SBSafeToAccessStartupDisk` routine to find out whether the internal hard disk is spinning so that your software can determine whether to make a disk access or postpone it until a time when the disk is already spinnings.

```
pascal Boolean SBSafeToAccessStartupDisk();
```

The `SBSafeToAccessStartupDisk` routine returns a Boolean value of `true` if the disk is spinning and `false` if it is not.

SBOpenModuleResourceFile

You can use the `SBOpenModuleResourceFile` routine to open a module resource file.

```
pascal short SBOpenModuleResourceFile(OSType fileCreator);
```

The `SBOpenModuleResourceFile` routine opens the resource fork of the module file whose creator is `fileCreator`, and return the file's reference number as its result. If the file cannot be found or opened, `SBOpenModuleResourceFile` returns a result of –1.

`SBOpenModuleResourceFile` provides a means for a module to load in large or infrequently used resources that it doesn't usually need, but that it requires for a particular operation.

SBLoadPreferences

You can use the `SBLoadPreferences` routine to load a resource from a preferences file.

```
pascal OSErr SBLoadPreferences(ConstStr255Param prefsResourceName,
                               Handle *preferences);
```

The `SBLoadPreferences` routine loads a resource containing a module's configuration information from the control strip's preferences file. The `PrefsResourceName` parameter points to a Pascal string containing the name of the resource. The `Preferences` parameter points to a variable that will hold a handle to the resource read from the file. The handle does not need to be preallocated.

If either `prefsResourceName` or `preferences` contains a nil pointer, `SBLoadPreferences` does nothing and returns a result of `paramErr`. If the resource is successfully loaded, it returns a result of 0. `SBLoadPreferences` can also return other Memory Manager and Resource Manager errors if it fails during some part of the process.

SBSavePreferences

You can use the SBSavePreferences routine to save a resource to a preferences file.

```
pascal OSErr SBSavePreferences(ConstStr255Param prefsResourceName,
                               Handle preferences);
```

The SBSavePreferences routine saves a resource containing a module's configuration information to the control strip's preferences file. The PrefsResourceName parameter points to a Pascal string containing the name of the resource. The preferences parameter contains a handle to a block of data which will be written to the file.

If either prefsResourceName or preferences has a nil value, SBSavePreferences does nothing and returns a result of paramErr. If the resource is successfully saved, SBSavePreferences returns a result of 0. SBSavePreferences can also return other Memory Manager and Resource Manager errors if it fails during some part of the process.

SBGetDetachedIndString

You can use the SBGetDetachedIndString routine to get a string from a detached resource.

```
pascal void SBGetDetachedIndString(StringPtr theString,
                                   Handle stringList,
                                   short whichString);
```

The SBGetDetachedIndString routine is the detached resource version of GetIndString. The parameter theString points to a Pascal string; stringList is a handle to a detached 'STR#' resource; and whichString is the index (1–n) into the array of Pascal strings contained in the detached resource. SBGetDetachedIndString will copy the string whose index is whichString into the space pointed to by theString. If whichString is out of range, SBGetDetachedIndString will return a zero-length string.

SBGetDetachIconSuite

You can use the SBGetDetachIconSuite routine to set up a detached icon suite.

```
pascal OSErr SBGetDetachIconSuite(Handle *theIconSuite,
                                  short theResID,
                                  unsigned long selector);
```

The `SBGetDetachIconSuite` routine creates a new icon suite, loads all of the requested icons, and then detaches the icons. The parameter `theIconSuite` points to the location where the handle to the icon suite will be stored; the parameter `theResID` is the resource ID of the icons that make up the icon suite; and the parameter `selector` tells which icons should be loaded into the suite. The `selector` parameter should typically contain one (or a combination of) the following values:

`svAllLargeData`	`0x000000FF`	load large 32-by-32-pixel icons (`'ICN#'`, `'icl4'`, `'icl8'`)
`svAllSmallData`	`0x0000FF00`	load small 16-by-16-pixel icons (`'ics#'`, `'ics4'`, `'ics8'`)
`svAllMiniData`	`0x00FF0000`	load mini 12-by-12-pixel icons (`'icm#'`, `'icm4'`, `'icm8'`)

These values may be OR-ed together to load combinations of icon sizes. `SBGetDetachIconSuite` returns an appropriate error code if it's unsuccessful, or 0 if if was able to load the icon suite. Note that if none of the icons comprising the icon suite could be found, the call returns the error `resNotFound`.

IMPORTANT

You should call `SBGetDetachIconSuite` only when the module's resource file is open, which is typically the case during a module's initialization call. ▲

SBTrackpopupMenu

You can use the `SBTrackpopupMenu` routine to manage a pop-up menu.

```
pascal short SBTrackpopupMenu(const Rect *moduleRect,
                              MenuHandle theMenu);
```

The `SBTrackpopupMenu` routine handles setting up and displaying a pop-up menu associated with a module. The module should pass a pointer to its display rectangle and a handle to the menu to use. The menu will be displayed just above the module's display rectangle, allowing the user to view the current configuration or to change the settings. `SBTrackpopupMenu` returns which menu item was selected, or 0 if no item was selected because the user moved the cursor outside the menu's bounds.

IMPORTANT

Menus are displayed in the control strip's font, so don't use `the CheckItem()` routine to mark menu items, because a checkmark is supported only in the system font. Use the `SetItemMark()` routine instead and pass it a bullet (•). ▲

SBTrackSlider

You can use the `SBTrackSlider` routine to display and set an arbitrary parameter.

```
pascal short SBTrackSlider(const Rect *moduleRect,
                            short ticksOnSlider,
                            short initialValue);
```

The `SBTrackSlider` routine displays an unlabeled slider above the module's display rectangle. You can use the slider for displaying and setting the state of an arbitrary parameter. The parameter `ModuleRect` contains a pointer to the module's display rectangle; `ticksOnSlider` is the upper bounds of the value returned by the slider; and `initialValue` is the starting position (0 to `ticksOnSlider`–1). When the user releases the mouse button, `SBTrackSlider` returns the final position.

SBShowHelpString

You can use the `SBShowHelpString` routine to display a help balloon.

```
pascal OSErr SBShowHelpString(const Rect *moduleRect,
                               StringPtr helpString);
```

The `SBShowHelpString` routine displays a module's help balloon. The module passes a pointer to its display rectangle and a pointer to a Pascal string, and the routine displays the balloon if possible. If the help string has a length of 0 or the Help Manager is unable to display a balloon, an error result is returned. If `SBShowHelpString` successfully displays the help balloon, it returns a result of 0.

SBGetBarGraphWidth

You can use the `SBGetBarGraphWidth` routine to find out the how wide a bar graph drawn by `SBDrawBarGraph` (described next) will be so that a module can calculate its display width.

```
pascal short SBGetBarGraphWidth(short barCount);
```

The `SBGetBarGraphWidth` routine returns the width of a bar graph containing `barCount` segments. If `barCount` has a value less than 0, the `SBGetBarGraphWidth` routine returns a width of 0.

SBDrawBarGraph

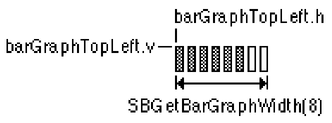
You can use the `SBDrawBarGraph` routine to draw a bar graph.

```
pascal void SBDrawBarGraph(short level, short barCount,
                           short direction,
                           Point barGraphTopLeft);
```

The `SBDrawBarGraph` routine draws a bar graph containing the number of segments specified by the `barCount` parameter in a module's display area. If the value of barCount is less than or equal to 0, `SBDrawBarGraph` does nothing.

The bar graph is drawn relative to the location specified by `barGraphTopLeft`. Figure 3-7 shows the way the point barGraphTopLeft determines the position of the bar graph.

Figure 3-7 Positioning a bar graph



The `level` parameter determines how many segments are highlighted. The value of `level` should be in the range of 0 to `barCount`–1. If the value of `level` is less than 0, no segments in the bar graph are highlighted; if `level` is greater than or equal to `barCount`, all segments in the bar graph are highlighted.

The direction parameter specifies which way the bar graph will be drawn to show a larger level. It should be one of the following values:

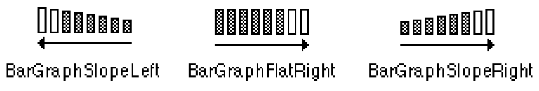
```
#define BarGraphSlopeLeft   -1   // max end of sloping graph is on the left

#define BarGraphFlatRight   0    // max end of flat graph is on the right

#define BarGraphSlopeRight  1    // max end of sloping graph is on the right
```

Figure 3-8 shows the resulting bar graph for each direction value. The arrows indicate which way an increasing level value is displayed. For sloped versions of the bar graph, the number of segments specified by the `barCount` value may not be larger than 8. If a larger `barCount` value is passed, `SBDrawBarGraph` draws nothing.

Figure 3-8 Direction of a bar graph



SBModalDialogInContext

You can use the `SBModalDialogInContext` in place of `ModalDialog` routine to keep background applications from getting run while your modal dialog window is visible.

```
pascal void SBModalDialogInContext(ModalFilterProcPtr filterProc,
                                   short *itemHit);
```

The `SBModalDialogInContext` routine is a special version of `ModalDialog` that doesn't allow background applications to get time while a modal dialog window is visible. You should use `SBModalDialogInContext` when you don't want any context switching to occur.

Gestalt Selectors

The control strip software installs two `Gestalt` selectors to return information to the outside world. One selector returns software attributes, and the other returns the software version.

gestaltControlStripAttr

The selector `gestaltControlStripAttr` (`'sdev'`) returns 32 bits describing the software attributes of this version of the control strip. Currently only the following bit is defined:

`gestaltControlStripExists`	0	1=control strip is installed

gestaltControlStripVersion

The selector `gestaltControlStripVersion` (`'sdvr'`) returns the version of control strip software that is installed. The format of the returned version is the same as that of the numeric part of a `'vers'` resource, that is:

Bits 31-24	Major part of the version, in BCD
Bits 23-20	Minor part of the version, in BCD
Bits 19-16	Bug release version, in BCD
Bits 15- 8	Release stage: 80=final 60=beta 40=alpha 20=development
Bits 7- 0	Revision level of nonreleased version, in binary

Thus, if the software version were 1.5.3b25, the `gestaltControlStripVersion` selector would return $01536019.

C H A P T E R 4

Power Manager Interface

This chapter describes the new application programming interface (API) to the Power Manager control software. Developers who provide expanded control panel software for the PowerBook Duo 280 and 280c computers will no longer need access to the Power Manager's internal data structures.

About the Power Manager Interface

Developers have written control panel software for previous PowerBook models that gives the user more control over the power management settings than is provided in the PowerBook control panel. Because that software reads and writes directly to the Power Manager's private data structure and parameter RAM, the software needs to be updated any time Apple Computer makes a change to the internal operation of the Power Manager.

System software for the PowerBook 520 and 540 computers and for future PowerBook models includes interface routines for program access to the Power Manager functions, so it is no longer necessary for applications to deal directly with the Power Manager's data structures. The new routines provide access to most of the Power Manager's parameters. Some functions will be reserved because of their overall effect on the system. The interface is extensible; it will probably grow over time to acccommodate new kinds of functions.

Things That May Change

By using the Power Manager interface, developers can isolate themselves from future changes to the internal operation of the Power Manager software.

IMPORTANT
Apple Computer reserves the right to change the internal operation of the Power Manager software. Developers should not make their applications depend on the Power Manager's internal data structures or parameter RAM. ▲

Starting with the PowerBook 520 and 540 models, developers should not depend on the Power Manager's internal data structures staying the same. In particular, developers should beware of the following assumptions regarding different PowerBook models:

- assuming that timeout values such as the hard disk spindown time reside at the same locations in parameter RAM
- assuming that the power cycling process works the same way or uses the same parameters
- assuming that direct commands to the Power Manager microcontroller are supported on all models

Checking for Routines

Before calling any of the Power Manager interface routines, it's always a good idea to call the Gestalt Manager to see if it they're present on the computer. The Gestalt Manager is described in *Inside Macintosh: Overview.*

A new bit has been added to the `gestaltPowerMgrAttr` selector:

```
#define gestaltPMgrDispatchExists 4
```

If that bit is set to 1, then the routines are present.

Because more routines may be added in the future, one of the new routines simply returns the number of routines that are implemented. The following code fragment determines both that the routines in general exist and that at least the hard disk spindown routine exists.

```
long   pmgrAttributes;
Boolean routinesExist;

routinesExist = false;
if (! Gestalt(gestaltPowerMgrAttr, &pmgrAttributes))
 if (pmgrAttributes & (1<<gestaltPMgrDispatchExists))
  if (PMSelectorCount() >= 7)
   routinesExist = true;
```

▲ **WARNING**
If you call a routine that does not exist, the call to the public Power Manager trap (if the trap exists) will return an error code, which your program could misinterpret as data. ▲

Power Manager Interface Routines

This section tells you how to call the interface routines for the Power Manager software. The interface routines are listed here in the order of their routine selector values, as shown in Table 4-1 on page 48.

Assembly-language note:
All the routines share a single trap, `_PowerMgrDispatch` ($A09E). The trap is register based: parameters are passed in register D0 and sometimes also in A0. A routine selector value passed in the low word of register D0 determines which routine is executed. ◆

Table 4-1 Interface routines and their selector values

	Selector value	
Routine name	decimal	hexadecimal
`PMSelectorCount`	0	$00
`PMFeatures`	1	$01
`GetSleepTimeout`	2	$02
`SetSleepTimeout`	3	$03
`GetHardDiskTimeout`	4	$04
`SetHardDiskTimeout`	5	$05
`HardDiskPowered`	6	$06
`SpinDownHardDisk`	7	$07
`IsSpindownDisabled`	8	$08
`SetSpindownDisable`	9	$09
`HardDiskQInstall`	10	$0A
`HardDiskQRemove`	11	$0B
`GetScaledBatteryInfo`	12	$0C
`AutoSleepControl`	13	$0D
`GetIntModemInfo`	14	$0E
`SetIntModemState`	15	$0F
`MaximumProcessorSpeed`	16	$10
`CurrentProcessorSpeed`	17	$11
`FullProcessorSpeed`	18	$12
`SetProcessorSpeed`	19	$13
`GetSCSIDiskModeAddress`	20	$14
`SetSCSIDiskModeAddress`	21	$15
`GetWakeupTimer`	22	$16
`SetWakeupTimer`	23	$17
`IsProcessorCyclingEnabled`	24	$18
`EnableProcessorCycling`	25	$19
`BatteryCount`	26	$1A
`GetBatteryVoltage`	27	$1B
`GetBatteryTimes`	28	$1C

PMSelectorCount

You can use the `PMSelectorCount` routine to determine which routines are implemented.

```
short PMSelectorCount();
```

DESCRIPTION

The `PMSelectorCount` routine returns the number of routine selectors present. Any routine whose selector value is greater than the returned value is not implemented.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `PMSelectorCount` is 0 ($00) in the low word of register D0. The number of selectors is returned in the low word of register D0.

PMFeatures

You can use the `PMFeatures` routine to find out which features of the Power Manager are implemented.

```
unsigned long PMFeatures();
```

DESCRIPTION

The `PMFeatures` routine returns a 32-bit field describing hardware and software features associated with the Power Manager on a particular machine. If a bit value is 1, that feature is supported or available; if the bit value is 0, that feature is not available. Unused bits are reserved by Apple for future expansion.

Field descriptions

Bit name	Bit number	Description
`hasWakeupTimer`	0	The wakeup timer is supported.
`hasSharedModemPort`	1	The hardware forces exclusive access to either SCC port A or the internal modem. (If this bit is not set, then typically port A and the internal modem may be used simultaneously by means of the Communications Toolbox.)
`hasProcessorCycling`	2	Processor cycling is supported; that is, when the computer is idle, the processor power will be cycled to reduce the power usage.

`mustProcessorCycle`	3	The processor cycling feature must be left on (turn it off at your own risk).
`hasReducedSpeed`	4	Processor can be started up at a reduced speed in order to extend battery life.
`dynamicSpeedChange`	5	Processor speed can be switched dynamically between its full and reduced speed at any time, rather than only at startup time.
`hasSCSIDiskMode`	6	The SCSI disk mode is supported.
`canGetBatteryTime`	7	The computer can provide an estimate of the battery time remaining.
`canWakeupOnRing`	8	The computer supports waking up from the sleep state when an internal modem is installed and the modem detects a ring.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `PMFeatures` is 1 ($01) in the low word of register D0. The 32-bit field of supported features is returned in register D0.

GetSleepTimeout

You can use the `GetSleepTimeout` routine to find out how long the computer will wait before going to sleep.

```
unsigned char GetSleepTimeout();
```

DESCRIPTION

The `GetSleepTimeout` routine returns the amount of time that the computer will wait after the last user activity before going to sleep. The value of `GetSleepTimeout` is expressed as the number of 15-second intervals that the computer will wait before going to sleep.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `GetSleepTimeout` is 2 ($02) in the low word of register D0. The sleep timeout value is returned in the low word of register D0.

SetSleepTimeout

You can use the `SetSleepTimeout` routine to set how long the computer will wait before going to sleep.

```
void SetSleepTimeout(unsigned char timeout);
```

DESCRIPTION

The `SetSleepTimeout` routine sets the amount of time the computer will wait after the last user activity before going to sleep. The value of `SetSleepTimeout` is expressed as the number of 15-second intervals making up the desired time. If a value of 0 is passed in, the routine sets the `timeout` value to the default value (currently equivalent to 8 minutes).

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `SetSleepTimeout` is 3 ($03) in the low word of register D0. The sleep timeout value to set is passed in the high word of register D0.

GetHardDiskTimeout

You can use the `GetHardDiskTimeout` routine to find out how long the computer will wait before turning off power to the internal hard disk.

```
unsigned char GetHardDiskTimeout();
```

DESCRIPTION

The `GetHardDiskTimeout` routine returns the amount of time the computer will wait after the last use of a SCSI device before turning off power to the internal hard disk. The value of `GetHardDiskTimeout` is expressed as the number of 15-second intervals the computer will wait before turning off power to the internal hard disk.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `GetHardDiskTimeout` is 4 ($04) in the low word of register D0. The hard disk timeout value is returned in the low word of register D0.

SetHardDiskTimeout

You can use the `SetHardDiskTimeout` routine to set how long the computer will wait before turning off power to the internal hard disk.

```
void SetHardDiskTimeout(unsigned char timeout);
```

DESCRIPTION

The `SetHardDiskTimeout` routine sets how long the computer will wait after the last use of a SCSI device before turning off power to the internal hard disk. The value of `SetHardDiskTimeout` is expressed as the number of 15-second intervals the computer will wait before turning off power to the internal hard disk. If a value of 0 is passed in, the routine sets the `timeout` value to the default value (currently equivalent to 4 minutes).

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `SetHardDiskTimeout` is 5 ($05) in the low word of register D0. The hard disk timeout value to set is passed in the high word of register D0.

HardDiskPowered

You can use the `HardDiskPowered` routine to find out whether the internal hard disk is on.

```
Boolean HardDiskPowered();
```

DESCRIPTION

The `HardDiskPowered` routine returns a Boolean value indicating whether or not the internal hard disk is powered up. A value of `true` means that the hard disk is on, and a value of `false` means that the hard disk is off.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `HardDiskPowered` is 6 ($06) in the low word of register D0. The Boolean result is returned in the low word of register D0.

SpinDownHardDisk

You can use the `SpinDownHardDisk` routine to force the hard disk to spin down.

```
void SpinDownHardDisk();
```

DESCRIPTION

The `SpinDownHardDisk` routine immediately forces the hard disk to spin down and power off if it was previously spinning. Calling `SpinDownHardDisk` will not spin down the hard disk if spindown is disabled by calling `SetSpindownDisable` (defined later in this section).

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `SpinDownHardDisk` is 7 ($07) in the low word of register D0.

IsSpindownDisabled

You can use the `IsSpindownDisabled` routine to find out whether hard disk spindown is enabled.

```
Boolean IsSpindownDisabled();
```

DESCRIPTION

The `IsSpindownDisabled` routine returns a Boolean `true` if hard disk spindown is disabled, or `false` if spindown is enabled.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `IsSpindownDisabled` is 8 ($08) in the low word of register D0. The Boolean result is passed in the low byte of register D0.

SetSpindownDisable

You can use the `SetSpindownDisable` routine to disable hard disk spindown.

```
void SetSpindownDisable(Boolean setDisable);
```

DESCRIPTION

The `SetSpindownDisable` routine enables or disables hard disk spindown, depending on the value of `setDisable`. If the value of `setDisable` is `true`, hard disk spindown will be disabled; if the value is `false`, spindown will be enabled.

Disabling hard disk spindown affects the `SpinDownHardDisk` routine, defined earlier, as well as the normal spindown that occurs after a period of hard disk inactivity.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `SetSpindownDisable` is 9 ($09) in the low word of register D0. The Boolean value to set is passed in the high word of register D0.

HardDiskQInstall

You can use the `HardDiskQInstall` routine to notify your software when power to the internal hard disk is about to be turned off.

```
OSErr HardDiskQInstall(HDQueueElement *theElement);
```

DESCRIPTION

The `HardDiskQInstall` routine installs an element into the hard disk power down queue to provide notification to your software when the internal hard disk is about to be powered off. For example, this feature might be used by the driver for an external battery-powered hard disk. When power to the internal hard disk is turned off, the external hard disk could be turned off as well.

The structure of `HDQueueElement` is as follows.

```
typedef pascal void (*HDSpindownProc)(HDQueueElement *theElement);

struct HDQueueElement {
  Ptr             hdQLink;       /* pointer to next queue element */
  short           hdQType;       /* queue element type (must be HDQType) */
  short           hdFlags;       /* miscellaneous flags (reserved) */
  HDSpindownProc hdProc;         /* pointer to routine to call */
  long      hdUser;              /* user-defined (variable storage, etc.) */
} HDQueueElement;
```

When power to the internal hard disk is about to be turned off, the software calls the routine pointed to by the `hdProc` field so that it can do any special processing. The routine will be passed a pointer to its queue element so that, for example, the routine can reference its variables.

Before calling `HardDiskQInstall`, the calling program must set the `hdQType` field to

```
#define HDPwrQType 'HD'      /* queue element type */
```

or the queue element won't be added to the queue and `HardDiskQInstall` will return an error.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `HardDiskQInstall` is 10 ($0A) in the low word of register D0. The pointer to the `HDQueue` element is passed in register A0. The result code is returned in the low word of register D0.

HardDiskQRemove

You can use the `HardDiskQRemove` routine to discontinue notification of your software when power to the internal hard disk is about to be turned off.

```
OSErr HardDiskQRemove(HDQueueElement *theElement);
```

DESCRIPTION

The `HardDiskQRemove` routine removes a queue element installed by `HardDiskQInstall`. If the `hdQType` field of the queue element is not set to `HDPwrQType`, `HardDiskQRemove` simply returns an error.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `HardDiskQRemove` is 11 ($0B) in the low word of register D0. The pointer to the `HDQueue` element is passed in register A0. The result code is returned in the low word of register D0.

GetScaledBatteryInfo

You can use the `GetScaledBatteryInfo` routine to find out the condition of the battery or batteries .

```
void GetScaledBatteryInfo(short whichBattery, BatteryInfo *theInfo);
```

DESCRIPTION

The `GetScaledBatteryInfo` routine provides a generic means of returning information about the battery or batteries in the system. Instead of returning a voltage value, the routine returns the battery level as a fraction of the total possible voltage.

Note
New battery technologies such as NiCad (nickel cadmium) and nickel metal hydride (NiMH) have replaced the sealed lead acid batteries of the original Macintosh Portable. The algorithm for determining the battery voltage that is documented in the Power Manager chapter of *Inside Macintosh,* Volume VI, is no longer correct for all PowerBook models. ◆

The value of `whichBattery` determines whether `GetScaledBatteryInfo` returns information about a particular battery or about the total battery level. The value of `GetScaledBatteryInfo` should be in the range of 0 to `BatteryCount()`. If the value of `whichBattery` is 0, `GetScaledBatteryInfo` returns a summation of all the batteries, that is, the effective battery level of the whole system. If the value of `whichBattery` is out of range, or the selected battery is not installed, `GetScaledBatteryInfo` will return a result of 0 in all fields. Here is a summary of the effects of the `whichBattery` parameter:

Value of `whichBattery`	Information returned
0	Total battery level for all batteries
From 1 to `BatteryCount`()	Battery level for the selected battery
Less than 0 or greater than `BatteryCount`	0 in all fields of `theInfo`

The `GetScaledBatteryInfo` routine returns information about the battery in the following data structure:

```
typedef struct BatteryInfo {
  unsigned char flags;             /* misc flags (see below) */
  unsigned char   warningLevel;    /* scaled warning level (0-255) */
  char            reserved;        /* reserved for internal use */
  unsigned char   batteryLevel;    /* scaled battery level (0-255) */
} BatteryInfo;
```

The `flags` character contains several bits that describe the battery and charger state. If a bit value is 1, that feature is available or is operating; if the bit value is 0, that feature is not operating. Unused bits are reserved by Apple for future expansion.

Field descriptions

Bit name	Bit number	Description
`batteryInstalled`	7	A battery is installed.
`batteryCharging`	6	The battery is charging.
`chargerConnected`	5	The charger is connected.

The value of `warningLevel` is the battery level at which the first low battery warning message will appear. The routine returns a value of 0 in some cases when it's not appropriate to return the warning level.

The value of `batteryLevel` is the current level of the battery. A value of 0 represents the voltage at which the Power Manager will force the computer into sleep mode; a value of 255 represents the highest possible voltage.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `GetScaledBatteryInfo` is 12 ($0C) in the low word of register D0. The `BatteryInfo` data are returned in the low word of register D0 as follows:

Bits 31–24	Flags
Bits 23–16	Warning level
Bits 15–8	Reserved
Bits 7–0	Battery level

AutoSleepControl

You can use the `AutoSleepControl` routine to turn the automatic sleep feature on and off.

```
void AutoSleepControl(Boolean enableSleep);
```

DESCRIPTION

The `AutoSleepControl` routine enables or disables the automatic sleep feature that causes the computer to go into sleep mode after a preset period of time. When `enableSleep` is set to `true`, the automatic sleep feature is enabled (this is the normal state). When `enableSleep` is set to `false`, the computer will not go into the sleep mode unless it is forced to either by some user action—for example, by the user's selecting Sleep from the Special menu of the Finder—or in a low battery situation.

IMPORTANT

Calling `AutoSleepControl` with `enableSleep` set to `false` multiple times increments the auto sleep disable level so that it requires the same number of calls to `AutoSleepControl` with `enableSleep` set to `true` to reenable the auto sleep feature. If more than one piece of software makes this call, auto sleep may not be reenabled when you think it should be. ▲

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `AutoSleepControl` is 13 ($0D) in the low word of register D0. The Boolean value is passed in the high word of register D0.

GetIntModemInfo

You can use the `GetIntModemInfo` routine to find out information about the internal modem.

```
unsigned long GetIntModemInfo();
```

DESCRIPTION

The `GetIntModemInfo` routine returns a 32-bit field containing information that describes the features and state of the internal modem. It can be called whether or not a modem is installed and will return the correct information.

If a bit is set, that feature or state is supported or selected; if the bit is cleared, that feature is not supported or selected. Undefined bits are reserved by Apple for future expansion.

Field descriptions

Bit name	Bit number	Description
`hasInternalModem`	0	An internal modem is installed.
`intModemRingDetect`	1	The modem has detected a ring on the telephone line.
`intModemOffHook`	2	The internal modem has taken the telephone line off hook (that is, you can hear the dial tone or modem carrier).
`intModemRingWakeEnb`	3	The computer will come out of sleep mode if the modem detects a ring on the telephone line and the computer supports this feature (see the `canWakeupOnRing` bit in `PMFeatures`).
`extModemSelected`	4	The external modem is selected (if this bit is set, then the modem port will be connected to port A of the SCC; if the modem port is not shared by the internal modem and the SCC, then this bit can be ignored).

Bits 15–31 contain the modem type, which will take on one of the following values:

–1	Modem is installed but type not recognized.
0	No modem is installed.
1	Modem is a serial modem.
2	Modem is a PowerBook Duo–style Express Modem.
3	Modem is a PowerBook 160/180–style Express Modem.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `GetIntModemInfo` is 14 ($0E) in the low word of register D0. The bit field to set is passed in the high word of register D0.

SetIntModemState

You can use the `SetIntModemState` routine to set some parts of the state of the internal modem.

```
void SetIntModemState(short theState);
```

DESCRIPTION

The `SetIntModemState` routine configures some of the internal modem's state information. Currently the only items that can be changed are the internal/external modem selection and the wakeup-on-ring feature.

To change an item of state information, the calling program sets the corresponding bit in `theState`. In other words, to change the internal/external modem setting, set bit 4 of `theState` to 1. To select the internal modem, bit 15 should be set to 0; to select the external modem, bit 15 should be set to 1.Using this method, the bits may be set or cleared independently, but they may not be set to different states at the same time.

Note
In some PowerBook computers, there is a hardware switch to connect either port A of the SCC or the internal modem to the modem port. The two are physically separated, but software emulates the serial port interface for those applications that don't use the Communications Toolbox. You can check the `hasSharedModemPort` bit returned by `PMFeatures` to determine which way the computer is set up. ◆

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `SetIntModemState` is 15 ($0F) in the low word of register D0. The bit field is returned in register D0.

MaximumProcessorSpeed

You can use the `MaximumProcessorSpeed` routine to find out the maximum speed of the computer's microprocessor.

```
short MaximumProcessorSpeed();
```

DESCRIPTION

The `MaximumProcessorSpeed` routine returns the maximum clock speed of the computer's microprocessor, in MHz.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `MaximumProcessorSpeed` is 16 ($10) in the low word of register D0. The processor speed value is returned in the low word of register D0.

CurrentProcessorSpeed

You can use the `CurrentProcessorSpeed` routine to find out the current clock speed of the microprocessor.

```
short CurrentProcessorSpeed();
```

DESCRIPTION

The `CurrentProcessorSpeed` routine returns the current clock speed of the computer's microprocessor, in MHz. The value returned will be different from the maximum processor speed if the computer has been configured to run with a reduced processor speed to conserve power.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `CurrentProcessorSpeed` is 17 ($11) in the low word of register D0. The processor speed value is returned in the low word of register D0.

FullProcessorSpeed

You can use the `FullProcessorSpeed` routine to find out whether the computer will run at full speed the next time it restarts.

```
Boolean FullProcessorSpeed();
```

DESCRIPTION

The `FullProcessorSpeed` routine returns a Boolean value of `true` if, on the next restart, the computer will start up at its maximum processor speed; it returns `false` if the computer will start up at its reduced processor speed.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `FullProcessorSpeed` is 18 ($12) in the low word of register D0. The Boolean result is returned in the low byte of register D0.

SetProcessorSpeed

You can use the `SetProcessorSpeed` routine to set the clock speed the microprocessor will use the next time it is restarted.

```
Boolean SetProcessorSpeed(Boolean fullSpeed);
```

DESCRIPTION

The `SetProcessorSpeed` routine sets the processor speed that the computer will use the next time it is restarted. If the value of `fullSpeed` is set to `true`, the processor will start up at its full speed (the speed returned by `MaximumProcessorSpeed`, described on page 59). If the value of `fullSpeed` is set to `false`, the processor will start up at its reduced speed.

For PowerBook models that support changing the processor speed dynamically, the processor speed will also be changed. If the speed is actually changed, `SetProcessorSpeed` will return `true`; if the speed isn't changed, it will return `false`.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `SetProcessorSpeed` is 19 ($13) in the low word of register D0. The Boolean value to set is passed in the high word of register D0. The Boolean result is returned in register D0.

GetSCSIDiskModeAddress

You can use the `GetSCSIDiskModeAddress` routine to find out the SCSI ID the computer uses in SCSI disk mode.

```
short GetSCSIDiskModeAddress();
```

DESCRIPTION

The `GetSCSIDiskModeAddress` routine returns the SCSI ID that the computer uses when it is started up in SCSI disk mode. The returned value is in the range 1 to 6.

Note
When the computer is in SCSI disk mode, the computer appears as a hard disk to another computer. ◆

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `GetSCSIDiskModeAddress` is 20 ($14) in the low word of register D0. The SCSI ID is returned in the low word of register D0.

SetSCSIDiskModeAddress

You can use the `SetSCSIDiskModeAddress` routine to set the SCSI ID for the computer to use in SCSI disk mode.

```
void SetSCSIDiskModeAddress(short scsiAddress);
```

DESCRIPTION

The `SetSCSIDiskModeAddress` routine sets the SCSI ID that the computer will use if it is started up in SCSI disk mode.

The value of `scsiAddress` must be in the range of 1 to 6. If any other value is given, the software sets the SCSI ID for SCSI disk mode to 2.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `SetSCSIDiskModeAddress` is 21 ($15) in the low word of register D0. The SCSI ID to set is passed in the high word of register D0.

GetWakeupTimer

You can use the `GetWakeupTimer` routine to find out when the computer will wake up from sleep mode.

```
void GetWakeupTimer(WakeupTime *theTime);
```

DESCRIPTION

The `GetWakeupTimer` routine returns the time when the computer will wake up from sleep mode.

If the PowerBook model doesn't support the wakeup timer, `GetWakeupTimer` returns a value of 0. The time and the enable flag are returned in the following structure:

```
typedef struct WakeupTime {
 unsigned long wakeTime;    /* wakeup time (same format as the time) */
 char      wakeEnabled;  /* 1=enable timer, 0=disable timer */
} WakeupTime;
```

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `GetWakeupTimer` is 22 ($16) in the low word of register D0. The pointer to `WakeupTime` is passed in register A0.

SetWakeupTimer

You can use the `SetWakeupTimer` routine to set the time when the computer will wake up from sleep mode.

```
void SetWakeupTimer(WakeupTime *theTime);
```

DESCRIPTION

The `SetWakeupTimer` routine sets the time when the computer will wake up from sleep mode and enables or disables the timer. On a PowerBook model that doesn't support the wakeup timer, `SetWakeupTimer` does nothing.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `SetWakeupTimer` is 23 ($17) in the low word of register D0. The pointer to `WakeupTime` is passed in register A0.

IsProcessorCyclingEnabled

You can use the `IsProcessorCyclingEnabled` routine to find out whether processor cycling is enabled.

```
Boolean IsProcessorCyclingEnabled();
```

DESCRIPTION

The `IsProcessorCyclingEnabled` routine returns a Boolean value of `true` if processor cycling is currently enabled, or `false` if it is disabled.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `IsProcessorCyclingEnabled` is 24 ($18) in the low word of register D0. The Boolean result is returned in register D0.

EnableProcessorCycling

You can use the `EnableProcessorCycling` routine to turn the processor cycling feature on and off.

```
void EnableProcessorCycling(Boolean enable);
```

DESCRIPTION

The `EnableProcessorCycling` routine enables processor cycling if a value of `true` is passed in, and disables it if `false` is passed.

▲ **WARNING**
You should follow the advice of the `mustProcessorCycle` bit in the feature flags when turning processor cycling off. Turning processor cycling off when it's not recommended can result in hardware failures due to overheating. ▲

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `EnableProcessorCycling` is 25 ($19) in the low word of register D0. The Boolean value to set is passed in the high word of register D0.

BatteryCount

You can use the `BatteryCount` routine to find out how many batteries the computer supports.

```
short BatteryCount();
```

DESCRIPTION

The `BatteryCount` routine returns the number of batteries that are supported internally by the computer. The value of `BatteryCount` returned may not be the same as the number of batteries currently installed.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `BatteryCount` is 26 ($1A) in the low word of register D0. The number of batteries supported is returned in the low word of register D0.

GetBatteryVoltage

You can use the `GetBatteryVoltage` routine to find out the battery voltage.

```
Fixed GetBatteryVoltage(short whichBattery);
```

DESCRIPTION

The `GetBatteryVoltage` routine returns the battery voltage as a fixed-point number.

The value of `whichBattery` should be in the range 0 to `BatteryCount()`–1. If the value of `whichBattery` is out of range, or the selected battery is not installed, `GetBatteryVoltage` will return a result of 0.0 volts.

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `GetBatteryVoltage` is 27 ($1B) in the low word of register D0. The battery number is passed in the high word of register D0. The 32-bit value of the battery voltage is returned in register D0.

GetBatteryTimes

You can use the `GetBatteryTimes` routine to find out about how much battery time remains.

```
void GetBatteryTimes (short whichBattery, BatteryTimeRec *theTimes);
```

DESCRIPTION

The `GetBatteryTimes` routine returns information about the time remaining on the computer's battery or batteries. The information returned has the following data structure:

```
typedef struct BatteryTimeRec {
  unsigned long expectedBatteryTime;/* estimated time remaining */
  unsigned long minimumBatteryTime;/* minimum time remaining */
  unsigned long maximumBatteryTime;/* maximum time remaining */
  unsigned long timeUntilCharged;/* time until full charge */
} BatteryTimeRec;
```

The time values are in seconds. The value of `expectedBatteryTime` is the estimated time remaining based on current usage patterns. The values of `minimumBatteryTime` and `maximumBatteryTime` are worst-case and best-case estimates, respectively. The value of `timeUntilCharged` is the time that remains until the battery or batteries are fully charged.

The value of `whichBattery` determines whether `GetBatteryTimes` returns the time information about a particular battery or the total time for all batteries. The value of `GetScaledBatteryInfo` should be in the range of 0 to `BatteryCount()`. If the value of `whichBattery` is 0, `GetBatteryTimes` returns a total time for all the batteries, that is, the effective battery time for the whole system. If the value of `whichBattery` is out of range, or the selected battery is not installed, `GetBatteryTimes` will return a result of 0 in all fields. Here is a summary of the effects of the `whichBattery` parameter:

Value of `whichBattery`	**Information returned**
0	Total battery time for all batteries
From 1 to `BatteryCount( )`	Battery time for the selected battery
Less than 0 or greater than `BatteryCount`	0 in all fields of `theTimes`

ASSEMBLY-LANGUAGE INFORMATION

The trap is `_PowerMgrDispatch` ($A09E). The selector value for `GetBatteryTimes` is 28 ($1C) in the low word of register D0. The pointer to `BatteryTimeRec` is passed in register A0.

Header File for Power Manager Dispatch

Here is a sample header file for access to the Power Manager.

```
/*************************************************************************************

   file:  PowerMgrDispatch.h

   contains:  header file for access to the Power Manager

   Copyright © 1992-1993 by Apple Computer, Inc. All rights reserved.

*************************************************************************************/

#ifndef __PowerMgrDispatch__

#define __PowerMgrDispatch__

#ifndef __TYPES__

#include <Types.h>

#endif


#ifndef gestaltPMgrDispatchExists

#define gestaltPMgrDispatchExists      4   /* gestaltPowerMgrAttr bit:
                                            1=PowerMgrDispatch exists */

#endif


/* bits in bitfield returned by PMFeatures */

#define hasWakeupTimer          0   /* 1=wakeup timer is supported  */

#define hasSharedModemPort      1   /* 1=modem port shared by SCC and internal modem */

#define hasProcessorCycling     2   /* 1=processor cycling is supported */

#define mustProcessorCycle      3   /* 1=processor cycling should not be turned off */

#define hasReducedSpeed         4   /* 1=processor can be started up at reduced speed */

#define dynamicSpeedChange      5   /* 1=processor speed can be switched dynamically */

#define hasSCSIDiskMode         6   /* 1=SCSI disk mode is supported */

#define canGetBatteryTime       7   /* 1=battery time can be calculated */

#define canWakeupOnRing     8   /* 1=can wake up when the modem detects a ring  */
```

```
/* bits in bitfield returned by GetIntModemInfo and set by SetIntModemState */
#define hasInternalModem     0   /* 1=internal modem installed */
#define intModemRingDetect   1   /* 1=internal modem has detected a ring */
#define intModemOffHook      2   /* 1=internal modem is off hook */
#define intModemRingWakeEnb  3   /* 1=wake up on ring is enabled */
#define extModemSelected     4   /* 1=external modem selected */
#define modemSetBit          15  /* 1=set bit, 0=clear bit (SetIntModemState) */


/* information returned by GetScaledBatteryInfo */
struct BatteryInfo {
   unsigned charflags;              /* misc flags (see below) */
   unsigned charwarningLevel;       /* scaled warning level (0-255) */
   char   reserved;                 /* reserved for internal use */
   unsigned charbatteryLevel;       /* scaled battery level (0-255) */
};


typedef struct BatteryInfo BatteryInfo;
/* bits in BatteryInfo.flags */
#define batteryInstalled     7      /* 1=battery is currently connected */
#define batteryCharging      6      /* 1=battery is being charged */
#define chargerConnected     5      /* 1=charger is connected to the PowerBook */
                                    /* (this doesn't mean the charger is plugged in) */


/* hard disk spindown notification queue element */
typedef struct HDQueueElement HDQueueElement;


typedef pascal void (*HDSpindownProc)(HDQueueElement *theElement);
```

```
struct HDQueueElement {
   Ptr            hdQLink;      /* pointer to next queue element */
   short          hdQType;      /* queue element type (must be HDQType) */
   short          hdFlags;      /* miscellaneous flags */
   HDSpindownProc hdProc;       /* pointer to routine to call */
   long           hdUser;       /* user-defined (variable storage, etc.) */
};

#define HDPwrQType'HD'          /* queue element type */
/* wakeup time record */
typedef struct WakeupTime {
   unsigned long    wakeTime;     /* wakeup time (same format as current time) */
   char             wakeEnabled;  /* 1=enable wakeup timer, 0=disable wakeup timer */
} WakeupTime;

/* battery time information (in seconds) */
typedef struct BatteryTimeRec {
   unsigned long    expectedBatteryTime;     /* estimated battery time remaining */
   unsigned long    minimumBatteryTime;      /* minimum battery time remaining */
   unsigned long    maximumBatteryTime;      /* maximum battery time remaining */
   unsigned long    timeUntilCharged;        /* time until battery is fully charged */
} BatteryTimeRec;

#ifdef __cplusplus
extern "C" {
#endif
```

```
#pragma parameter __D0 PMSelectorCount(__D0)
short PMSelectorCount()
    = {0x7000, 0xA09E};

#pragma parameter __D0 PMFeatures
unsigned long PMFeatures()
    = {0x7001, 0xA09E};

#pragma parameter __D0 GetSleepTimeout
unsigned char GetSleepTimeout()
    = {0x7002, 0xA09E};

#pragma parameter __D0 SetSleepTimeout(__D0)
void SetSleepTimeout(unsigned char timeout)
    = {0x4840, 0x303C, 0x0003, 0xA09E};

#pragma parameter __D0 GetHardDiskTimeout
unsigned char GetHardDiskTimeout()
    = {0x7004, 0xA09E};

#pragma parameter __D0 SetHardDiskTimeout(__D0)
void SetHardDiskTimeout(unsigned char timeout)
    = {0x4840, 0x303C, 0x0005, 0xA09E};

#pragma parameter __D0 HardDiskPowered
Boolean HardDiskPowered()
    = {0x7006, 0xA09E};
```

```
#pragma parameter __D0 SpinDownHardDisk
void SpinDownHardDisk()
    = {0x7007, 0xA09E};

#pragma parameter __D0 IsSpindownDisabled
Boolean IsSpindownDisabled()
    = {0x7008, 0xA09E};

#pragma parameter __D0 SetSpindownDisable(__D0)
void SetSpindownDisable(Boolean setDisable)
    = {0x4840, 0x303C, 0x0009, 0xA09E};

#pragma parameter __D0 HardDiskQInstall(__A0)
OSErr HardDiskQInstall(HDQueueElement *theElement)
    = {0x700A, 0xA09E};

#pragma parameter __D0 HardDiskQRemove(__A0)
OSErr HardDiskQRemove(HDQueueElement *theElement)
    = {0x700B, 0xA09E};

#pragma parameter __D0 GetScaledBatteryInfo(__D0,__A0)
void GetScaledBatteryInfo(short whichBattery, BatteryInfo *theInfo)
    = {0x4840, 0x303C, 0x000C, 0xA09E, 0x2080};

#pragma parameter __D0 AutoSleepControl(__D0)
void AutoSleepControl(Boolean enableSleep)
    = {0x4840, 0x303C, 0x000D, 0xA09E};
```

```
#pragma parameter __D0 GetIntModemInfo(__D0)

unsigned long GetIntModemInfo()

    = {0x700E, 0xA09E};


#pragma parameter __D0 SetIntModemState(__D0)

void SetIntModemState(short theState)

    = {0x4840, 0x303C, 0x000F, 0xA09E};


#pragma parameter __D0 MaximumProcessorSpeed

short MaximumProcessorSpeed()

    = {0x7010, 0xA09E};


#pragma parameter __D0 CurrentProcessorSpeed

short CurrentProcessorSpeed()

    = {0x7011, 0xA09E};


#pragma parameter __D0 FullProcessorSpeed

Boolean FullProcessorSpeed()

    = {0x7012, 0xA09E};


#pragma parameter __D0 SetProcessorSpeed(__D0)

Boolean SetProcessorSpeed(Boolean fullSpeed)

    = {0x4840, 0x303C, 0x0013, 0xA09E};


#pragma parameter __D0 GetSCSIDiskModeAddress

short GetSCSIDiskModeAddress()

    = {0x7014, 0xA09E};
```

```
#pragma parameter __D0 SetSCSIDiskModeAddress(__D0)
void SetSCSIDiskModeAddress(short scsiAddress)
    = {0x4840, 0x303C, 0x0015, 0xA09E};

#pragma parameter __D0 GetWakeupTimer(__A0)
void GetWakeupTimer(WakeupTime *theTime)
    = {0x7016, 0xA09E};

#pragma parameter __D0 SetWakeupTimer(__A0)
void SetWakeupTimer(WakeupTime *theTime)
    = {0x7017, 0xA09E};

#pragma parameter __D0 IsProcessorCyclingEnabled
Boolean IsProcessorCyclingEnabled()
    = {0x7018, 0xA09E};

#pragma parameter __D0 EnableProcessorCycling(__D0)
void EnableProcessorCycling(Boolean enable)
    = {0x4840, 0x303C, 0x0019, 0xA09E};

#pragma parameter __D0 BatteryCount
short BatteryCount()
    = {0x701A, 0xA09E};

#pragma parameter __D0 GetBatteryVoltage(__D0)
Fixed GetBatteryVoltage(short whichBattery)
    = {0x4840, 0x303C, 0x001B, 0xA09E};
```

```
#pragma parameter __D0 GetBatteryTimes(__D0,__A0)

void GetBatteryTimes(BatteryTimeRec *theTimes)

   = {0x4840, 0x303C, 0x001C, 0xA09E};


#ifdef __cplusplus

}

#endif

#endif
```

Color Lookup Table

This appendix contains more information about the color lookup table used in the PowerBook Duo 280 and 280c computers. Table A-1 shows the color values for each index. Index numbers are shown in hexadecimal ($0000) and decimal (0). Red (R), green (G), and blue (B) color values are shown in hexadecimal ($0000).

The first 215 entries are combinations, made up of R, G, and B values of $0000, $3333, $6666, $9999, $CCCC, and $FFFF. You should generally select colors from those 215 entries of the CLUT.

The last 40 entries are assigned to red ramp, green ramp, blue ramp, and gray scale. The values of those last 40 entries can be dithered, either spatially or temporally, to simluate the appearance of intermediate colors. Each colored ramp consists of a single color with values of $0000, $1111, $2222, $4444, $5555, $7777, $8888, $AAAA, $BBBB, $DDDD, and $EEEE. The gray ramp has those same values in all three color channels.

Table A-1 Color lookup table

Index (hexadecimal)	Index (decimal)	R value	G value	B value
$0000	0	$FFFF	$FFFF	$FFFF
$0001	1	$FFFF	$FFFF	$CCCC
$0002	2	$FFFF	$FFFF	$9999
$0003	3	$FFFF	$FFFF	$6666
$0004	4	$FFFF	$FFFF	$3333
$0005	5	$FFFF	$FFFF	$0000
$0006	6	$FFFF	$CCCC	$FFFF
$0007	7	$FFFF	$CCCC	$CCCC
$0008	8	$FFFF	$CCCC	$9999
$0009	9	$FFFF	$CCCC	$6666
$000A	10	$FFFF	$CCCC	$3333
$000B	11	$FFFF	$CCCC	$0000
$000C	12	$FFFF	$9999	$FFFF
$000D	13	$FFFF	$9999	$CCCC
$000E	14	$FFFF	$9999	$9999
$000F	15	$FFFF	$9999	$6666

continued

Table A-1 Color lookup table (continued)

Index (hexadecimal)	Index (decimal)	R value	G value	B value
$0010	16	$FFFF	$9999	$3333
$0011	17	$FFFF	$9999	$0000
$0012	18	$FFFF	$6666	$FFFF
$0013	19	$FFFF	$6666	$CCCC
$0014	20	$FFFF	$6666	$9999
$0015	21	$FFFF	$6666	$6666
$0016	22	$FFFF	$6666	$3333
$0017	23	$FFFF	$6666	$0000
$0018	24	$FFFF	$3333	$FFFF
$0019	25	$FFFF	$3333	$CCCC
$001A	26	$FFFF	$3333	$9999
$001B	27	$FFFF	$3333	$6666
$001C	28	$FFFF	$3333	$3333
$001D	29	$FFFF	$3333	$0000
$001E	30	$FFFF	$0000	$FFFF
$001F	31	$FFFF	$0000	$CCCC
$0020	32	$FFFF	$0000	$9999
$0021	33	$FFFF	$0000	$6666
$0022	34	$FFFF	$0000	$3333
$0023	35	$FFFF	$0000	$0000
$0024	36	$CCCC	$FFFF	$FFFF
$0025	37	$CCCC	$FFFF	$CCCC
$0026	38	$CCCC	$FFFF	$9999
$0027	39	$CCCC	$FFFF	$6666
$0028	40	$CCCC	$FFFF	$3333
$0029	41	$CCCC	$FFFF	$0000
$002A	42	$CCCC	$CCCC	$FFFF
$002B	43	$CCCC	$CCCC	$CCCC
$002C	44	$CCCC	$CCCC	$CCCC
$002D	45	$CCCC	$CCCC	$6666

continued

Table A-1 Color lookup table (continued)

Index (hexadecimal)	Index (decimal)	R value	G value	B value
$002E	46	$CCCC	$CCCC	$3333
$002F	47	$CCCC	$CCCC	$0000
$0030	48	$CCCC	$9999	$FFFF
$0031	49	$CCCC	$9999	$CCCC
$0032	50	$CCCC	$9999	$9999
$0033	51	$CCCC	$9999	$6666
$0034	52	$CCCC	$9999	$3333
$0035	53	$CCCC	$9999	$0000
$0036	54	$CCCC	$6666	$FFFF
$0037	55	$CCCC	$6666	$CCCC
$0038	56	$CCCC	$6666	$9999
$0039	57	$CCCC	$6666	$6666
$003A	58	$CCCC	$6666	$3333
$003B	59	$CCCC	$6666	$0000
$003C	60	$CCCC	$3333	$FFFF
$003D	61	$CCCC	$3333	$CCCC
$003E	62	$CCCC	$3333	$9999
$003F	63	$CCCC	$3333	$6666
$0040	64	$CCCC	$3333	$3333
$0041	65	$CCCC	$3333	$0000
$0042	66	$CCCC	$0000	$FFFF
$0043	67	$CCCC	$0000	$CCCC
$0044	68	$CCCC	$0000	$9999
$0045	69	$CCCC	$0000	$6666
$0046	70	$CCCC	$0000	$3333
$0047	71	$CCCC	$0000	$0000
$0048	72	$9999	$FFFF	$FFFF
$0049	73	$9999	$FFFF	$CCCC
$004A	74	$9999	$FFFF	$9999
$004B	75	$9999	$FFFF	$6666

continued

Table A-1 Color lookup table (continued)

Index (hexadecimal)	Index (decimal)	R value	G value	B value
$004C	76	$9999	$FFFF	$3333
$004D	77	$9999	$FFFF	$0000
$004E	78	$9999	$9999	$FFFF
$004F	79	$9999	$CCCC	$CCCC
$0050	80	$9999	$CCCC	$9999
$0051	81	$9999	$CCCC	$6666
$0052	82	$9999	$CCCC	$3333
$0053	83	$9999	$CCCC	$0000
$0054	84	$9999	$9999	$FFFF
$0055	85	$9999	$9999	$CCCC
$0056	86	$9999	$9999	$9999
$0057	87	$9999	$9999	$6666
$0058	88	$9999	$9999	$3333
$0059	89	$9999	$9999	$0000
$005A	90	$9999	$6666	$FFFF
$005B	91	$9999	$6666	$CCCC
$005C	92	$9999	$6666	$9999
$005D	93	$9999	$6666	$6666
$005E	94	$9999	$6666	$3333
$005F	95	$9999	$6666	$0000
$0060	96	$9999	$3333	$FFFF
$0061	97	$9999	$3333	$CCCC
$0062	98	$9999	$3333	$9999
$0063	99	$9999	$3333	$6666
$0064	100	$9999	$3333	$3333
$0065	101	$9999	$3333	$0000
$0066	102	$9999	$0000	$FFFF
$0067	103	$9999	$0000	$CCCC
$0068	104	$9999	$0000	$9999
$0069	105	$9999	$0000	$6666

continued

Table A-1 Color lookup table (continued)

Index (hexadecimal)	Index (decimal)	R value	G value	B value
$006A	106	$9999	$0000	$3333
$006B	107	$9999	$0000	$0000
$006C	108	$6666	$FFFF	$FFFF
$006D	109	$6666	$FFFF	$CCCC
$006E	110	$6666	$FFFF	$9999
$006F	111	$6666	$FFFF	$6666
$0070	112	$6666	$FFFF	$3333
$0071	113	$6666	$FFFF	$0000
$0072	114	$6666	$CCCC	$FFFF
$0073	115	$6666	$CCCC	$CCCC
$0074	116	$6666	$CCCC	$9999
$0075	117	$6666	$CCCC	$6666
$0076	118	$6666	$CCCC	$3333
$0077	119	$6666	$CCCC	$0000
$0078	120	$6666	$9999	$FFFF
$0079	121	$6666	$9999	$CCCC
$007A	122	$6666	$9999	$9999
$007B	123	$6666	$9999	$6666
$007C	124	$6666	$9999	$3333
$007D	125	$6666	$9999	$0000
$007E	126	$6666	$6666	$FFFF
$007F	127	$6666	$6666	$CCCC
$0080	128	$6666	$6666	$9999
$0081	129	$6666	$6666	$6666
$0082	130	$6666	$6666	$3333
$0083	131	$6666	$6666	$0000
$0084	132	$6666	$3333	$FFFF
$0085	133	$6666	$3333	$CCCC
$0086	134	$6666	$3333	$9999
$0087	135	$6666	$3333	$6666

continued

Table A-1 Color lookup table (continued)

Index (hexadecimal)	Index (decimal)	R value	G value	B value
$0088	136	$6666	$3333	$3333
$0089	137	$6666	$3333	$0000
$008A	138	$6666	$0000	$FFFF
$008B	139	$6666	$0000	$CCCC
$008C	140	$6666	$0000	$9999
$008D	141	$6666	$0000	$6666
$008E	142	$6666	$0000	$3333
$008F	143	$6666	$0000	$0000
$0090	144	$3333	$FFFF	$FFFF
$0091	145	$3333	$FFFF	$CCCC
$0092	146	$3333	$FFFF	$9999
$0093	147	$3333	$FFFF	$6666
$0094	148	$3333	$FFFF	$3333
$0095	149	$3333	$FFFF	$0000
$0096	150	$3333	$CCCC	$FFFF
$0097	151	$3333	$CCCC	$CCCC
$0098	152	$3333	$CCCC	$9999
$0099	153	$3333	$CCCC	$6666
$009A	154	$3333	$CCCC	$3333
$009B	155	$3333	$CCCC	$0000
$009C	156	$3333	$9999	$FFFF
$009D	157	$3333	$9999	$CCCC
$009E	158	$3333	$9999	$9999
$009F	159	$3333	$9999	$6666
$00A0	160	$3333	$9999	$3333
$00A1	161	$3333	$9999	$0000
$00A2	162	$3333	$6666	$FFFF
$00A3	163	$3333	$6666	$CCCC
$00A4	164	$3333	$6666	$9999
$00A5	165	$3333	$6666	$6666

continued

Table A-1 Color lookup table (continued)

Index (hexadecimal)	Index (decimal)	R value	G value	B value
$00A6	166	$3333	$6666	$3333
$00A7	167	$3333	$6666	$0000
$00A8	168	$3333	$3333	$FFFF
$00A9	169	$3333	$3333	$CCCC
$00AA	170	$3333	$3333	$9999
$00AB	171	$3333	$3333	$6666
$00AC	172	$3333	$3333	$3333
$00AD	173	$3333	$3333	$0000
$00AE	174	$3333	$0000	$FFFF
$00AF	175	$3333	$0000	$CCCC
$00B0	176	$3333	$0000	$9999
$00B1	177	$3333	$0000	$6666
$00B2	178	$3333	$0000	$3333
$00B3	179	$3333	$0000	$0000
$00B4	180	$0000	$FFFF	$FFFF
$00B5	181	$0000	$FFFF	$CCCC
$00B6	182	$0000	$FFFF	$9999
$00B7	183	$0000	$FFFF	$6666
$00B8	184	$0000	$FFFF	$3333
$00B9	185	$0000	$FFFF	$0000
$00BA	186	$0000	$CCCC	$FFFF
$00BB	187	$0000	$CCCC	$CCCC
$00BC	188	$0000	$CCCC	$9999
$00BD	189	$0000	$CCCC	$6666
$00BE	190	$0000	$CCCC	$3333
$00BF	191	$0000	$CCCC	$0000
$00C0	192	$0000	$9999	$FFFF
$00C1	193	$0000	$9999	$CCCC
$00C2	194	$0000	$9999	$9999
$00C3	195	$0000	$9999	$6666

continued

Table A-1 Color lookup table (continued)

Index (hexadecimal)	Index (decimal)	R value	G value	B value
$00C4	196	$0000	$9999	$3333
$00C5	197	$0000	$9999	$0000
$00C6	198	$0000	$6666	$FFFF
$00C7	199	$0000	$6666	$CCCC
$00C8	200	$0000	$6666	$9999
$00C9	201	$0000	$6666	$6666
$00CA	202	$0000	$6666	$3333
$00CB	203	$0000	$3333	$0000
$00CC	204	$0000	$3333	$FFFF
$00CD	205	$0000	$3333	$CCCC
$00CE	206	$0000	$3333	$9999
$00CF	207	$0000	$3333	$6666
$00D0	208	$0000	$3333	$3333
$00D1	209	$0000	$0000	$0000
$00D2	210	$0000	$0000	$FFFF
$00D3	211	$0000	$0000	$CCCC
$00D4	212	$0000	$0000	$9999
$00D5	213	$0000	$0000	$6666
$00D6	214	$0000	$0000	$3333
$00D7	215	$EEEE	$0000	$0000
$00D8	216	$DDDD	$0000	$0000
$00D9	217	$BBBB	$0000	$0000
$00DA	218	$AAAA	$0000	$0000
$00DB	219	$8888	$0000	$0000
$00DC	220	$7777	$0000	$0000
$00DD	221	$5555	$0000	$0000
$00DE	222	$4444	$0000	$0000
$00DF	223	$2222	$0000	$0000
$00E0	224	$1111	$0000	$0000
$00E1	225	$0000	$EEEE	$0000

continued

Table A-1 Color lookup table (continued)

Index (hexadecimal)	Index (decimal)	R value	G value	B value
$00E2	226	$0000	$0000	$0000
$00E3	227	$0000	$BBBB	$0000
$00E4	228	$0000	$AAAA	$0000
$00E5	229	$0000	$8888	$0000
$00E6	230	$0000	$7777	$0000
$00E7	231	$0000	$5555	$0000
$00E8	232	$0000	$4444	$0000
$00E9	233	$0000	$2222	$0000
$00EA	234	$0000	$1111	$0000
$00EB	235	$0000	$0000	$EEEE
$00EC	236	$0000	$0000	$DDDD
$00ED	237	$0000	$0000	$BBBB
$00EE	238	$0000	$0000	$AAAA
$00EF	239	$0000	$0000	$8888
$00F0	240	$0000	$0000	$7777
$00F1	241	$0000	$0000	$5555
$00F2	242	$0000	$0000	$4444
$00F3	243	$0000	$0000	$2222
$00F4	244	$0000	$0000	$1111
$00F5	245	$EEEE	$EEEE	$EEEE
$00F6	246	$DDDD	$DDDD	$DDDD
$00F7	247	$BBBB	$BBBB	$BBBB
$00F8	248	$AAAA	$AAAA	$AAAA
$00F9	249	$8888	$8888	$8888
$00FA	250	$7777	$7777	$7777
$00FB	251	$5555	$5555	$5555
$00FC	252	$4444	$4444	$4444
$00FD	253	$2222	$2222	$2222
$00FE	254	$1111	$1111	$1111
$00FF	255	$0000	$0000	$0000

Index

This Apple document was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Proof and final pages were created on an Apple LaserWriter Pro 630 printer. Line art was created using Adobe™ Illustrator. 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 III

DEVELOPMENTAL EDITORS
Wendy Krafft, Beverly Zegarski

ILLUSTRATORS
Deborah Dennis, Shawn Morningstar

Special thanks to Steve Christensen, Rhoads Hollowell, Mike Puckett, and Steve Sfarzo.