Developer Note

Macintosh 630 DOS Compatible Computers

Macintosh LC 630 DOS Compatible Computer
Macintosh Quadra 630 DOS Compatible Computer
Macintosh Performa 640 DOS Compatible Computer

Developer Note

Developer Press

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

Simultaneously published in the United States and Canada.

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

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

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

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

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

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

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

Contents

Figures and Tables

About This Note

This developer note describes the Macintosh 630 DOS compatible computer, a Macintosh computer with a built-in 486-type microprocessor. This developer note describes the DOS compatibility features of this computer and the way DOS software can communicate with Mac OS software.

Note
This developer note applies to the Macintosh LC 630 DOS Compatible, the Macintosh Quadra 630 DOS Compatible, and the Macintosh Performa 640 DOS Compatible computers. ◆

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

Contents of This Note

This developer note has three chapters.

- Chapter 1, "Introduction," presents a summary of the features of the Macintosh 630 DOS Compatible computer and a brief description of the way it operates.
- Chapter 2, "Hardware Design," describes the design of the DOS Compatibility Card and the interface devices that allow DOS programs to operate in a Macintosh 630 DOS Compatible computer.
- Chapter 3, "The PC Interface Driver," describes the system software that allows DOS programs to communicate with Mac OS programs on the Macintosh 630 DOS Compatible computer.

Supplementary Documents

For installation and operating instructions, refer to the user's manual that accompanies the product.

For information about the unmodified Macintosh LC 630 and Macintosh Quadra 630 computers, refer to *Macintosh Developer Note Number 10,* APDA catalog number R0568LL/A. Developer notes for the individual Macintosh models are also published electronically in the quarterly Reference Library Edition of the Developer CD series, available through APDA.

For information about the Cx486DX2 microprocessor, refer to *Cx486DX/DX2 3 and 5 Volt Microprocessors* published by Cyrix Corporation.

For a general description of big-endian and little-endian byte addressing, please refer to Appendix A, "Overview of PowerPC Technology," in *Macintosh Developer Note Number 8,* APDA catalog number R0566LL/A.

Developers may also need copies of the appropriate Apple reference books. You should have the relevant books of the *Inside Macintos*h series, particularly *Inside Macintosh: Processes.*

Obtaining Information From APDA

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

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

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

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

Conventions and Abbreviations

This developer note uses the following typographical conventions and abbreviations.

Typographical Conventions

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

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

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

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

Standard Abbreviations

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

A	amperes	mA	milliamperes
dB	decibels	µA	microamperes
GB	gigabytes	MB	megabytes
Hz	hertz	MHz	megahertz
in.	inches	mm	millimeters
k	1000	ms	milliseconds
K	1024	µs	microseconds
KB	kilobytes	ns	nanoseconds
kg	kilograms	Ω	ohms
kHz	kilohertz	sec.	seconds
kΩ	kilohms	V	volts
lb.	pounds	W	watts

Other abbreviations used in this note include

$*n*	hexadecimal value *n*
ADB	Apple Desktop Bus
API	application program interface
A/V	audiovisual
BIOS	basic input/output system
CAS	column address strobe (a memory control signal)

CGA	Color Graphics Adapter
CLUT	color lookup table
codec	coder/decoder
CPU	central processing unit
DAC	digital-to-analog converter
DC	direct current
DMA	direct memory access
DOS	disk operating system
DRAM	dynamic RAM
EGA	Enhanced Graphics Adapter
FIFO	first in, first out
FM	frequency modulation
GND	ground
IC	integrated circuit
I/O	input and output
IRQ	interrupt request
ISA	Industry Standard Architecture
Mac OS	Macintosh Operating System
MDA	Monochrome Display Adapter
n.c.	no connection
OS	operating system
PC	personal computer
PDS	processor-direct slot
PRAM	parameter random-access memory
RAM	random-access memory
RAS	row address strobe
RGB	red-green-blue, a video signal format with separate red, green, and blue color components
ROM	read-only memory
RTC	real-time clock
SCSI	Small Computer System Interface
SIMM	Single Inline Memory Module
SVGA	super video graphics adapter
TCP/IP	Transport Control Protocol/Interface Program
UART	universal asynchronous receiver-transmitter
VCO	voltage-controlled oscillator
VGA	video graphics adapter

CHAPTER 1

Introduction

The Macintosh 630 DOS Compatible computers are modified Macintosh LC 630, Macintosh Quadra 630, and Macintosh Performa 640 computers with additional hardware that provides IBM-compatible PC functionality. The added hardware includes a 486-type microprocessor and interface devices that allow it to use the I/O capabilities of the host computer. The Macintosh 630 DOS Compatible computer provides a cost-effective system with DOS performance equivalent to a stand-alone PC.

Features

The Macintosh 630 DOS Compatible computer is a Macintosh 630–series computer with the following additional hardware components:

- the main compatibility card, installed in the Macintosh 630 computer's 68040 socket
- the sound expansion card, installed on the main compatibility card
- the game adapter card, installed in the Macintosh 630 computer's I/O expansion slot

Collectively, those components make up the DOS compatibility subsystem.

Note
A Macintosh 630 DOS Compatible computer retains all the features of a Macintosh 630-series computer, including the ability to accept optional video, tuner, and communications cards. ◆

The main card replaces the host's 68040 microprocessor, which is installed in a socket on the card. The game adapter card is installed in the host computer's I/O expansion slot (PDS) and provides a 15-pin connector for a joystick. In addition, a 16-pin ribbon cable carries the audio and video signals between the main card and the host computer.

The following list is a summary of the features of the DOS compatibility subsystem. Each of these features is described later in this developer note.

- ***Processor.*** The main compatibility card has a Cx486DX2 or 80486DX2 microprocessor operating at a clock speed of 66 MHz.
- ***Expansion RAM.*** The main compatibility card accepts one standard 72-pin DRAM SIMM containing either 2, 4, 8, 16, or 32 MB. Recommended DRAM speed is 80 ns or less.
- ***Shared RAM.*** The DOS compatibility subsystem can use part of the DRAM in the Macintosh host computer. The user can select a memory size of 2, 4, 8, 16, or 32 MB, provided the Macintosh computer has enough memory installed.
- ***Direct memory access.*** A DMA channel supports I/O transfers when memory is installed on the main compatibility card; when using shared memory, DMA is provided through the Macintosh system.
- ***Video support.*** A VGA video system on the main compatibility card supports Macintosh monitors from 13-inch through 20-inch size and all available VGA monitors.
- ***Sound card.*** The DOS compatibility subsystem provides standard PC sound output through a sound expansion card that produces 16-bit sound output compatible with

Sound Blaster cards. Sounds are played through the host computer's sound output jack and built-in speaker.

- ***Serial ports.*** The DOS compatibility subsystem uses the host computer's two serial ports by way of serial port interfaces emulated in hardware.
- ***Parallel port.*** The DOS compatibility subsystem has access to a printer on the host computer by way of a parallel port interface emulated in hardware.
- ***Floppy disk.*** The DOS compatibility subsystem uses the host computer's 3.5-inch internal floppy drive.
- ***Hard disk.*** The DOS compatibility subsystem has access to the host computer's internal hard drive and external SCSI devices.
- ***Keyboard and mouse.*** The DOS compatibility subsystem uses the host computer's keyboard and mouse through hardware emulation.
- ***Joystick.*** The game adapter card includes a DB-15 connector that supports a standard PC-style joystick.

The DOS compatibility subsystem in the Macintosh 630 DOS Compatible computer provides performance and features comparable with midrange 80486DX computers currently available. Table 1-1 compares the features of the Macintosh 630 DOS Compatible computer with a midrange PC computer.

Table 1-1 Comparison with a midrange PC

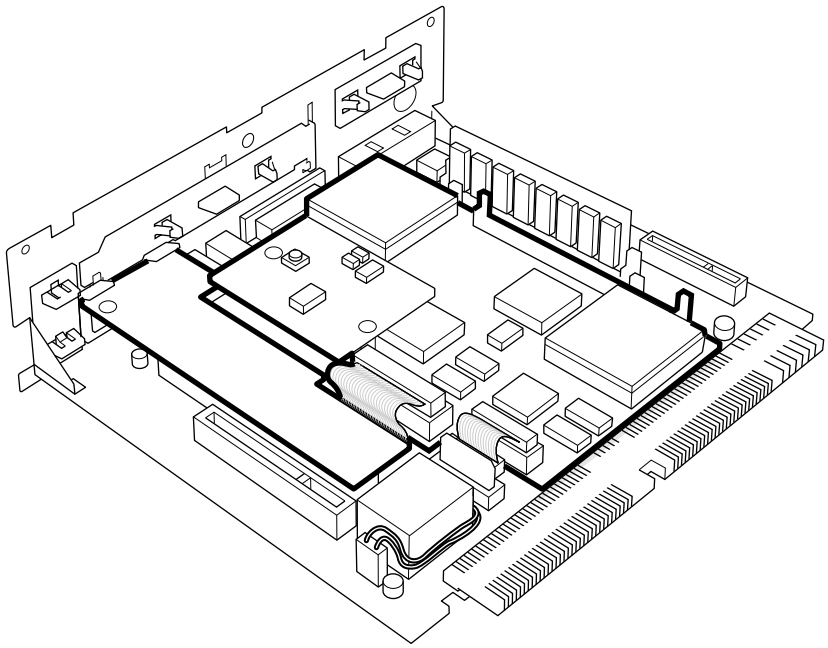
Feature	Macintosh 630 DOS Compatible computer	Midrange PC computer
Processor	66 MHz Cx486DX2 or 80486DX2	Same
Network support	IPX and TCP/IP available	Optional
Onboard RAM	None	4 MB
Expansion RAM	1 SIMM (up to 32 MB)	8 SIMMs (up to 64 MB)
Video support	VGA, EGA, CGA, MDA	Same
Video RAM	512 KB DRAM	Same
Sound card	Sound out only	Optional
Serial ports	2 (COM1 and COM2)	Same
Parallel port	1 (emulated, XT/AT compatible)	1
Keyboard	AT compatible	Same
Mouse	PS/2 compatible	Same
Floppy disk	3.5-inch	3.5-inch and 5.25-inch
AT expansion	None	3 slots
External SCSI	Yes	No

Notice that the Macintosh 630 DOS Compatible computer has greater sound and networking capabilities than a midrange PC. In addition, the Macintosh 630 DOS Compatible computer provides external SCSI expansion (for hard drives and removable-media devices only).

How the DOS Compatibility Subsystem Works

The DOS compatibility subsystem occupies both the 68040 microprocessor socket and the I/O expansion slot of the host Macintosh 630–series computer. Figure 1-1 shows the interior of a Macintosh 630 DOS Compatible computer; the heavy outlines identify the DOS compatibility subsystem.

Figure 1-1 The DOS compatibility subsystem

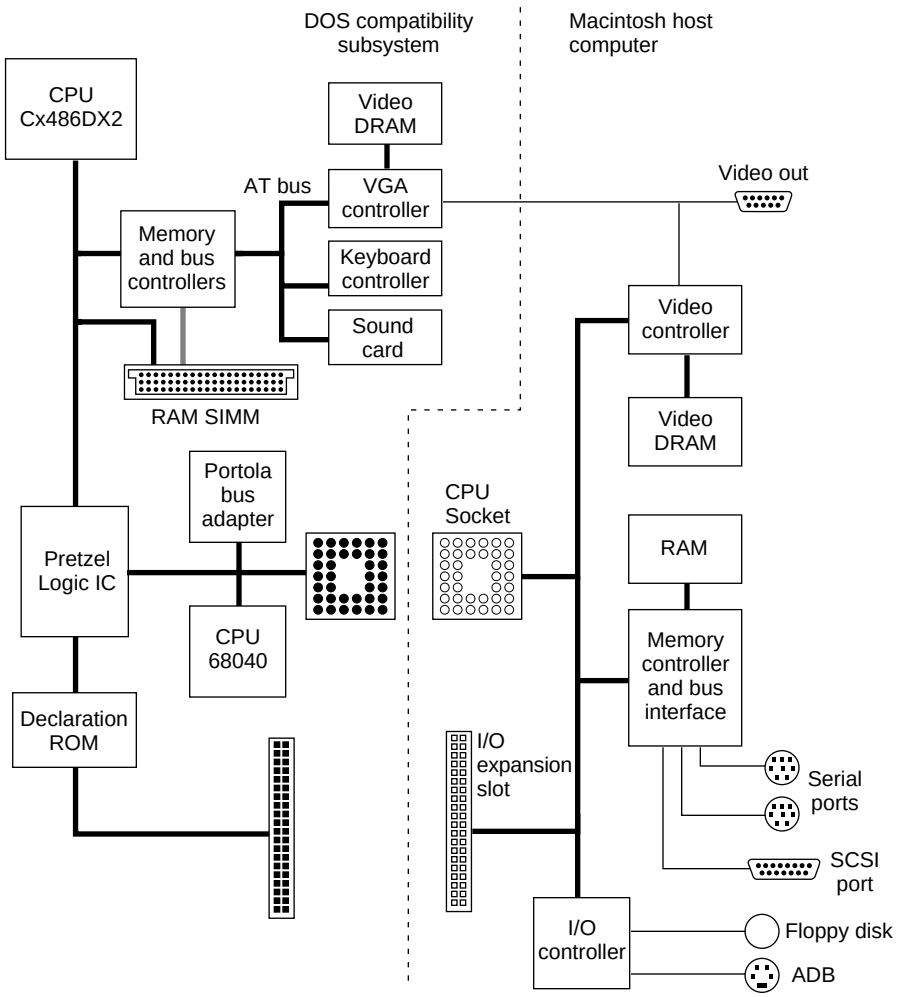


Note
The main logic board in the Macintosh 630 DOS Compatible computer has two RAM SIMM slots and an audio and video connector, making it different from the logic board in other Macintosh 630–series computers. ◆

Outline of Operation

Figure 1-2 shows a simplified block diagram of the DOS compatibility subsystem installed in a Macintosh 630–series computer.

Figure 1-2 Simplified block diagram



The diagram shows some of the hardware devices on the main compatibility card: the memory controller and DRAM SIMM, and the VGA controller and video RAM. It also shows the Pretzel Logic IC, which acts as a bus converter between the compatibility card and the Macintosh computer and provides the interface to Macintosh devices that emulate PC devices. Chapter 2, “Hardware Design,” gives more information about the devices on the DOS compatibility subsystem and the way they operate in conjunction with the Macintosh host computer.

I/O Capabilities

The DOS compatibility subsystem uses I/O devices built into or connected to the Macintosh host computer. This section describes the I/O capabilities; for more information on their operation, see "I/O Components" on page 26.

Floppy Disk

The DOS compatibility subsystem has access to the Macintosh host computer's 3.5-inch internal floppy drive, which can read and write DOS-formatted floppy disks. When RAM SIMM is installed on the main compatibility card, I/O data transfers use the DMA channel. When the DOS subsystem is using shared memory, I/O data transfers are handled by the disk drivers in the Macintosh Operating System (Mac OS).

Hard Disk

The DOS compatibility subsystem has access to the host computer's internal IDE hard drive and external SCSI devices. I/O data transfers use the DMA channel when RAM SIMM is installed on the main compatibility card. When using shared memory, I/O data transfers are handled by the disk drivers in the Mac OS.

Serial Ports

The DOS compatibility subsystem has access to the serial ports on the Macintosh host computer. To provide software compatibility, an IC on the main compatibility card emulates the registers of the standard serial port ICs found in most PC/AT computers. For more information on register emulation, see "Serial Port Support" on page 27.

An adapter cable is necessary to connect a PC serial device to a Macintosh serial port. Table 1-2 shows the signals on the 9-pin connector on the Macintosh serial ports and the corresponding connections on the 25-pin and 9-pin connectors used with a PC serial port.

Table 1-2 Corresponding serial-port signals

Pin number on Macintosh serial port	RS-422 signal name	Pin number on 25-pin PC serial port	Pin number on 9-pin PC serial port	RS-232 signal name
1	HSKo	20	4	DTR
2	HSKi	5, 8	8, 1	CTS, DCD
3	TXD–	2	3	TXD
4	GND	7	5	GND
5	RXD–	3	2	RXD
6	TXD+	n.c.		none

continued

Table 1-2 Corresponding serial-port signals (continued)

Pin number on Macintosh serial port	RS-422 signal name	Pin number on 25-pin PC serial port	Pin number on 9-pin PC serial port	RS-232 signal name
7	GPi	n.c.		none
8	RXD+	7	5	GND
9	+5V	n.c.		none

The Macintosh serial ports are RS-422 ports and do not support all the RS-232 signals. In particular, the Carrier Detect (CD), Data Set Ready (DSR), Request To Send (RTS), and Ring Indicator (RI) signals are not available. Not all RS-232 devices will work using the RS-422 protocol.

Note
The 9-pin sockets on the Macintosh serial ports accept either 9-pin or 8-pin connectors. ◆

Parallel Printer Port

A custom IC on the main compatibility card emulates a compatible parallel port interface and enables the driver software to send printer data to a printer through the Macintosh host computer. The printer may be connected directly to the Macintosh computer's serial port or it may be on a network and selected by means of the Chooser. The IC provides register compatibility only; for more information, see "Printer Port Support" on page 27.

Keyboard and Mouse

The main compatibility card includes hardware that emulates a PC keyboard and mouse using inputs from the keyboard and mouse on the Macintosh host computer. The software protocols for the keyboard and mouse are the same as on a standard PC.

Note
The DOS compatibility subsystem can work with another user input device, such as a trackball, but the device must be connected to the Macintosh host computer by way of the ADB port. ◆

The PC Setup control panel allows the user to define a key command (hot key) to switch operation of the user interface devices (the keyboard, the mouse, and the monitor) between the DOS compatibility subsystem and the Macintosh host computer. The key command consists of the Command key (⌘) and at least one other key. Chapter 5 in the user's manual gives instructions for setting the key command.

Sound

Sound is generated on the DOS compatibility subsystem either by the 8254 interval timer on the main compatibility card or by the sound expansion card. The 8254 interval timer is responsible for the standard system beep (square wave output) and sound effects. The sound expansion card provides 16-bit stereo sound output only and is software compatible with the Sound Blaster register model.

Sounds generated by the DOS compatibility subsystem are routed to the host computer's main logic board where they are mixed with sounds from other sources in the system.

Video Monitor

The DOS compatibility subsystem shares the video monitor used with the Macintosh host computer. The monitors that can be shared are

- Apple Color Plus 14-inch Display
- Macintosh Color Display (14-inch)
- Apple Multiple Scan 15 Display (15-inch)
- VGA (640 by 480 pixels)
- SVGA (800 by 600 pixels)

Note
The DOS compatibility subsystem does not support all the available resolutions on a Macintosh multiple scan monitor. ◆

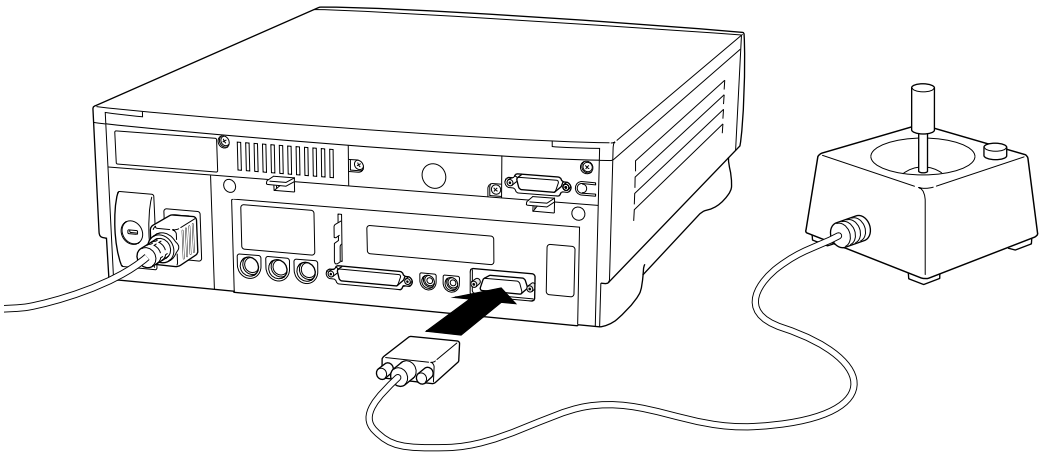
Video signals generated by the DOS compatibility subsystem are routed to the host computer's motherboard where they are mixed with the computer's video signals and sent to the video monitor. System software turns off the video from one video source when the other is selected. See "Sharing a Monitor" on page 22.

The host computer detects the type of video monitor at startup time by interrogating the monitor sense lines. For more information about the monitor sense lines, see the section "Video Components" on page 21.

Game Controller Port

The game controller port is a DB-15 connector on the game adapter card. It is accessible at the I/O expansion slot at the rear of the computer and is used to connect a PC/AT compatible game controller (joystick). Figure 1-3 shows the back of a Macintosh 630 DOS Compatible computer with the game controller installed. The game controller can be used only with programs running on the PC.

Figure 1-3 Installing the joystick



CHAPTER 2

Hardware Design

The DOS compatibility subsystem contains three printed circuits cards: the main compatibility card, the game adapter card, and the sound expansion card. The main compatibility card contains the processor and memory components, the video display components, and the I/O components. The game adapter card contains the slot declaration ROM and the game controller port. The sound expansion card contains the sound generation ICs. The individual ICs are

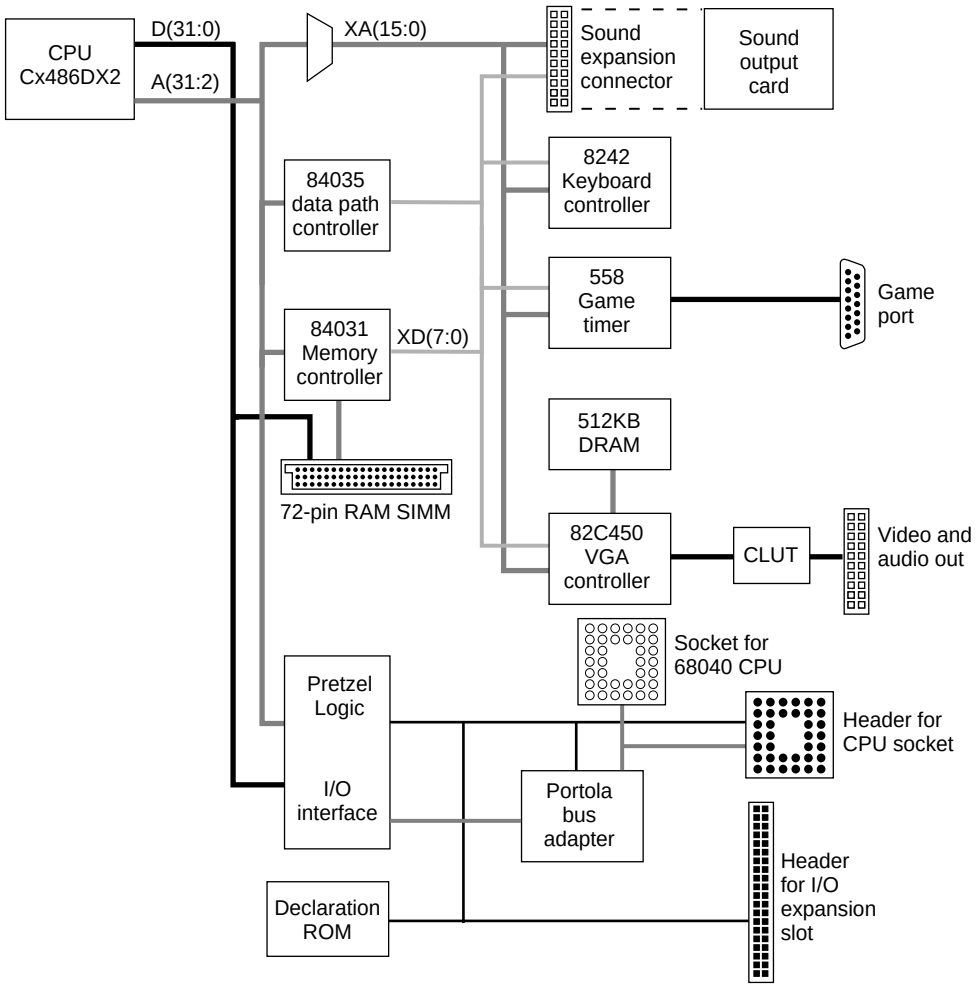
- processor and memory components
  - Cx486DX2 or 80486DX2 microprocessor
  - 84031 memory controller
  - 84035 data path controller
  - Portola bus adapter IC
  - 68040 microprocessor (from main logic board)
- video display components
  - 82C450 VGA controller
  - MU9C9760 SynDAC (video DAC, CLUT, and clock synthesizer)
- I/O components
  - 8242 keyboard and mouse controller
  - Pretzel Logic I/O interface controller
- game adapter card
  - slot declaration ROM
- sound expansion card
  - CT2501 sound system IC (combination bus interface, codec, and audio mixer)
  - YMF262 digitally controlled FM synthesizer
  - YAC512 audio DAC for the YMF262

All the ICs in the DOS compatibility subsystem are commercially available parts except the Pretzel Logic IC and the Portola IC, which are Apple custom parts. The individual ICs are described later.

The block diagram in Figure 2-1 shows the main components of the main compatibility card and the game adapter card.

The main compatibility card has a high-speed processor bus linking the 80486 microprocessor to the RAM SIMM by way of the 84031 memory controller. The main compatibility card also has an I/O bus: the XD (data) bus. The XD bus is used for devices on the card: the keyboard controller, the game controller, the VGA controller, and the sound card. The 84031 memory controller acts as the I/O bus controller that isolates the XD bus from the processor bus. The XD bus is 8 bits wide and operates synchronously with the processor bus at a fraction of its speed. The 84035 data path controller provides additional PC/AT-compatible I/O ports that are accessible through the I/O controller.

Figure 2-1 Detailed block diagram



Processor and Memory Components

The processor and memory components includes the Cx486DX2 or 80486DX2 microprocessor and the control devices for the onboard memory: the 84031 memory controller and the 84035 data path controller.

Cx486DX2 Microprocessor

The DOS compatibility subsystem has a Cx486DX2 or 80486DX2 microprocessor running at 66 MHz (33 MHz processor bus clock). The microprocessor supports 32-bit data paths and 32-bit addresses; that allows up to 4 GB of physically addressable memory.

Some of the key features of the Cx486DX2 are listed below. Please refer to *Cx486DX/DX2 3 and 5 Volt Microprocessors* from Cyrix Corporation for further information.

- full 32-bit addressing architecture with 32-bit data interface
- internal 8 KB unified instruction and data cache
- internal cache operation in either write-back or write-through mode
- instruction prefetch mechanism during idle bus activity
- internal FPU that is faster than the FPU in a standard 80486DX
- internal memory management unit supporting both memory segmentation and paging
- internal write buffer (1 longword deep) to support posted writes
- dynamic bus sizing to support 8-bit and 16-bit peripherals
- support for synchronous 16-byte block reads
- backward compatible with existing 80x86 code

The Cx486DX2 IC in the DOS compatibility subsystem is in a 168-pin ceramic PGA package. With a clock speed of 66 MHz, this package requires a heat sink.

PC System Bus and Devices

The PC system bus is defined as the unbuffered microprocessor pins that are required to support slave and alternate bus masters. This bus operates synchronously at the same clock speed as the processor bus clock (33 MHz). The bus supports burst reads and compelled writes (due to the write-through cache). The key devices attached to this bus are the memory and bus controller IC and the Pretzel Logic bus interface IC.

Cache Snooping

The cache in the 80486 microprocessor supports bus snooping to track activity on the bus that alters the memory represented in the internal cache. In the DOS compatibility subsystem, even though the sound card operates as an alternate bus master, the snoop control lines are deactivated.

The memory space reserved for the PC (whether local or shared memory) cannot be cached or modified by the Mac OS, so it presents no coherency issues.

The interface provides no hooks to support bus snooping in either the PC environment or the Macintosh environment.

Byte Order

Big-endian and *little-endian* are two ways of defining the order in which bytes are addressed. *Big-endian* means that the most significant byte corresponds to the lowest address and the least significant byte corresponds to the highest address. *Little-endian* means that the most significant byte corresponds to the highest address and the least significant byte corresponds to the lowest address.

The 680x0 microprocessors use big-endian byte addressing and the 80x86 microprocessors use little-endian byte addressing. This disparity poses a problem for the DOS compatibility subsystem because its 80486-type microprocessor is dependent on the Mac OS to load applications and data from peripheral devices. When the Mac OS loads PC data from floppy disk, it stores that data at addresses that match the big-endian convention. To allow the PC to function properly, it must be able to read the data the same way as the Mac OS; that is, the transfer must be address invariant. To make that possible with the disparity in addressing modes, the interface IC (Pretzel Logic) performs a byte swapping operation.

Byte swapping is performed for all PC data resident on the Macintosh host computer, that is, for both shared memory data and DMA (I/O) data. The interface IC also swaps the bytes of data in one of the message mailbox data registers. The other data register does not provide for byte swapping and thus provides data invariance.

For a general description of big-endian and little-endian byte addressing, please refer to Appendix A, "Overview of PowerPC Technology," in *Macintosh Developer Note Number 8*.

Misaligned Transfers

Data misalignment occurs when the DOS compatibility subsystem is configured for shared memory. The problem arises because of differences in the lengths of data transfers on the two types of microprocessors.

All memory read and write operations in the Macintosh environment are longword (4 byte) aligned: the low-order 2 bits of the address are zeros. Each time the 80486 performs a 1-, 2-, 3-, or 4-byte memory read operation, the Macintosh host computer performs a 4-byte access. The full 32 bits of data are presented on the PC side and the 80486 accepts the required byte lanes. When the 80486 requests multiple bytes of data from a nonaligned address (that is, when the data extends across a longword address), the 80486 splits the access into two separate transfers.

When the 80486 performs a misaligned write operation, the interface IC (Pretzel Logic) first checks to see if the transfer is an aligned transfer on the Macintosh host computer. If it is, the transfer is allowed to proceed. If the write is misaligned with respect to the host computer (for example, a 3-byte transfer, or a 2-byte transfer that does not fall on a word boundary), the interface IC forces the 80486 to break the transfer into several single-byte operations. This ensures that misaligned transfers on the PC side get mapped to the proper addresses in the host computer's memory.

Table 2-1 on page 16 shows the byte order of the different transfer sizes supported by the 68040 and 80486 microprocessors.

Table 2-1 Microprocessor transfer comparison

Transfer size	Bytes enabled on a 68040 microprocessor	Bytes enabled on a 80486 microprocessor
1 byte	3	0
	2	1
	1	2
	0	3
2 bytes	3, 2	0, 1
	1, 0	1, 2
		2, 3
3 bytes	Not supported	1, 2, 3
		0, 1, 2
4 bytes	3, 2, 1, 0	0, 1, 2, 3

Interrupts

The 84031 and 84035 ICs, described in later sections, are responsible for generating all interrupt requests to the Cx486DX2 microprocessor. The 84035 data path controller IC generates the maskable interrupt resulting from the various IRQ sources. For interrupt functions, the 84035 is equivalent to two cascaded 8259 interrupt controllers (PIC) as found in the original PC/AT computer. Table 2-2 shows the interrupt definitions for the DOS compatibility subsystem.

Table 2-2 Definitions of PC interrupts

Interrupt number	Description
0	Interval timer
1	Keyboard
2	PIC 2
3*	COM2 port*
4*	COM1 port*
5	Sound expansion card
6*	Message mailbox*
7*	Parallel port 1*
8	Real-time clock
12	Mouse

NOTE Asterisk (*) indicates interrupt requests with source in the interface (Pretzel Logic) IC.

The source of the Macintosh interrupt (SLOT_E signal) is the Pretzel Logic IC (described on page 26). With the exception of transfers in which the Pretzel Logic IC becomes bus master, all service between the PC side and the Macintosh host computer is interrupt driven.

The master interrupt status register in the Pretzel Logic IC contains the state of all interrupt sources on the card. Each of these interrupt sources can be individually masked by an accompanying master interrupt enable register. Additionally, higher resolution into the cause of the interrupt can be determined by use of the secondary interrupt status registers for COM1 and COM2 ports, keyboard and mouse port, and DMA channel.

The interrupt and status registers in the Pretzel Logic IC are accessible from the Macintosh environment only. From the PC environment, the registers for the COM1 and COM2 ports and for the printer port match their standard definitions.

Bus Arbitration

The PC system bus supports the Cx486DX2 microprocessor (as bus master) and the two 8-bit DMA channels on the sound expansion connector. Sound DMA cycles use the DMA controllers in the 84035 data path controller IC, but hard disk and floppy disk DMA cycles between the PC and Macintosh memory or peripherals do not. Instead, the disk DMA cycles require the processor to poll the DMA status register and perform I/O reads and writes to the DMA data register in the Pretzel Logic IC.

On the PC system bus, the 84031 memory controller IC and the Pretzel Logic IC respond as slave devices.

The HOLD signal to the Cx486DX2 microprocessor is formed by the logical OR of the DMA controller's output with the autoconfiguration control output. The HOLD signal is used by the DMA controller to hold off the processor for DMA transfer. It's also used at startup time to tristate the processor address bus and allow the Pretzel Logic IC to autoconfigure.

Because there is no way of signaling a bus error to the Cx486DX2 microprocessor, no bus timers exist on the PC side to monitor the PC system bus activity and terminate faulty cycles. For an address outside the decoded range, the 84031 bus controller signals completion and operation continues.

A bus error on the PC system bus will cause the PC to hang. When that happens, the Macintosh environment is not affected, so it can be used to restart the PC, either by the Ctl-Alt-Del key sequence if the PC keyboard is still responding or by the Cmd-Ctl-Alt-Del key sequence if not.

The 84031 memory controller IC acts as the master of the XD(ISA) bus on the PC side. The 8242 keyboard and mouse controller and the 82C450 VGA controller respond only as slave devices on this bus.

The Macintosh system bus on the Macintosh Quadra 630 computer can support three bus masters. Table 2-3 summarizes the priorities assigned to the fixed arbitration devices.

Table 2-3 Arbitration priorities

Priority	Device
Highest	DRAM refresh
	Network DMA
Lowest	Pretzel Logic and the 68040

A secondary DMA arbitration circuit in the Portola IC arbitrates between the 68040 and the Pretzel Logic DMA transfers to the host computer's memory and I/O devices. When performing DMA cycles to the PC, the Pretzel Logic IC becomes a 68040 bus master.

The DOS compatibility subsystem relies on the host computer to maintain a watchdog timer for the I/O expansion slot. This timer is necessary to prevent the host computer from hanging while waiting for a response from the Pretzel Logic IC.

Expansion

The DOS compatibility subsystem does not provide any way to add ISA or EISA expansion boards. The local ISA bus (XD) is closed and supports only the 8242 keyboard and mouse controller, the 80450 VGA controller, the 558 game timer, and the sound expansion card. The COM1, COM2, and LPT1 peripherals usually found on the AT-ISA bus are directly accessible from the Pretzel Logic IC through the processor system bus.

A 50-pin connector on the main compatibility card provides access to a subset of the ISA signals for the sound expansion card.

84031 Memory Controller

The 84031 memory controller IC performs the following system-level functions:

- DRAM control
- ROM control
- system clock generation
- ISA bus control
- VL (local) bus arbitration

DRAM Control

The DRAM on the card is directly interfaced to the local data bus. The /RAS, /CAS, /DWE, and MA lines are driven directly from the 84031 memory controller IC without external buffers.

The DRAM controller in the 84031 supports page mode operation. For memory read operations, the page hit cycles are either 3-2-2-2 or 4-2-2-2 bursts. For write operations, the page hits are 1-wait-state accesses. Both read and write operations are designed for DRAM devices with 80 ns access time and have RAS-CAS delays of two T states.

The main compatibility card has a slot for one 32-bit-wide SIMM that supports up to two banks of DRAM (for double-sided modules). No system DRAM is soldered on the card. A single-sided SIMM can hold 1 MB, 4 MB, or 16 MB using 1, 4, or 16 Mbit DRAM devices, respectively. Double-sided SIMM modules can hold double those amounts of memory.

The DOS compatibility subsystem does not require a DRAM SIMM with parity.

The presence of a DRAM SIMM on the card is sensed by the Pretzel Logic IC at startup and stored in a register in the IC. Upon reading this register bit, the startup software determines the size of the memory and programs the 84031's configuration registers with starting and ending addresses for each bank. If a DRAM SIMM is not present, shared memory is assumed and the software disables all local DRAM banks in the 84031.

BIOS Control

The DOS compatibility subsystem has no ROM except for the declaration ROM common to all Macintosh expansion cards. The basic input/output system (BIOS) is stored in the host computer's RAM and accessed by way of the shared memory channel in the Pretzel Logic IC.

Note
The BIOS and the BIOS extensions in the host computer's memory are always accessed by way of the shared memory interface, regardless of whether a DRAM SIMM is installed on the card. ◆

At reset the Cx486DX2 microprocessor issues the starting reset-vector address from within the address range of the BIOS image in the upper 64 KB of shared system memory. The Pretzel Logic IC remaps this address range down to the lower 1 MB region where the BIOS actually resides. The Pretzel Logic IC also performs the address translation between the BIOS addresses on the PC side and the corresponding addresses in shared memory on the Macintosh host computer.

Clock Generation

The 84031 memory controller IC receives a 2X clock and generates a low-skew 1X and 2X clock for the system and the Cx486DX2 processor. In addition, it divides down the 2X clock to generate the BUSCLK signal for the ISA bus.

ISA Bus Control

The 84031 memory controller IC handles all accesses to the ISA bus by the Cx486DX2. In addition, the memory controller performs data buffering to form the XD bus for local peripherals such as the keyboard, joystick, and VGA controllers. The memory controller also provides support for local bus slaves such as the Pretzel Logic IC.

84035 Data Path Controller

The 84035 data path controller IC performs the following system-level functions:

- system reset
- interrupt control
- speaker drive

In addition, the data path controller IC contains the PC/AT-compatible DMA channels and the system arbitration logic for DMA masters and local bus masters. Those functions are needed by the sound expansion card.

Clocks

The data path controller IC receives a 14.31818 MHz clock signal and divides it by 12 to form the 1.19 MHz clock used by the 8254 timers. In addition, the data path controller receives a 32.768 kHz clock signal for the internal real-time clock. All CPU related functions are based on the 1X clock generated by the memory controller IC.

System Reset

The data path controller IC generates the reset signals for the DOS compatibility subsystem. The data path controller generates the SYSRESET and CPURESET signals based on the /PWRGOOD signal from the Pretzel Logic IC. The CPURESET signal is also affected by soft reset requests received over the control link from the memory controller IC.

The /PWRGOOD signal controls several other signals. It disables all outputs and gates off all inputs to the 84035 except for the /PWRSTB signal (PRAM, RTC power), the 14 MHz clock (14.31818 MHz input), the 32 kHz clock (32.768 kHz input), and the /PWRGOOD signal itself. When the /PWRGOOD signal goes high, the outputs are enabled and the SYSRESET and CPURESET signals are driven high. The data path controller holds the SYSRESET and CPURESET signals high for 8 million cycles of the SCLK clock to ensure proper startup of the 14.31818 MHz oscillator and to allow time for the VCO in the Cx486DX2 to stabilize.

The SYSRESET and CPURESET signals are generated as follows: The SYSRESET signal is generated based on the /PWRGOOD signal alone. The CPURESET signal is generated based on /PWRGOOD but is also generated for soft resets. Soft resets can occur due to a keyboard controller reset, a CPU shutdown cycle, or the transition of bit 0 of port 92 in the 84035 from 0 to 1.

Keyboard reset and shutdown are sent to the 84035 data path controller through the control link from the 84031 memory controller, which decodes shutdown cycles and receives keyboard reset from the 8242 keyboard and mouse controller.

The 84035 data path controller IC generates the /A20M signal to the Cx486DX2 microprocessor. The 84035 generates the /A20M signal by ORing together the GATEA20 signal from the keyboard controller and bit 1 of port 92 in the 84035. The keyboard controller's GATEA20 information comes from the 84031 memory controller through the control link.

Interrupt Control

The 84035 data path controller IC contains two 8259-compatible interrupt controllers. The interrupt numbers are listed in Table 2-2 on page 16.

Portola Bus Adapter IC

The Portola bus adapter is a custom IC that provides some signal modification and bus arbitration between the Pretzel Logic IC and the 68040 bus. The main functions of the Portola bus adapter are:

- generating the device select for the declaration ROM
- generating the handshake signals for the PDS
- remapping the slave registers in the Pretzel Logic IC to an unused portion of the host computer's memory
- providing bus arbitration between the Pretzel Logic IC and the 68040

Burst Transfers

The Macintosh host computer and the DOS compatibility subsystem perform burst transfers in similar ways. The Pretzel Logic IC supports burst memory transfers of 16-byte length (4 longwords). Those transfers are translated to MOVE16 transfers on the 68040 microprocessor.

Video Components

The DOS compatibility subsystem includes a complete video system to support PC video. The video components consist of a DRAM-based frame buffer and a VGA controller with an integrated color lookup table (CLUT), triple digital-to-analog converter (DAC), and clock generator.

Sharing a Monitor

Video output from the DOS compatibility subsystem is displayed on a monitor shared with the host Macintosh computer. A ribbon cable carries the video signals from the main compatibility card to the main logic board in the host computer.

The user can switch the monitor (along with the keyboard and mouse) from one computer subsystem to the other by typing a programmable command key sequence (hot key). When the user switches the monitor to the Mac OS, the software sets a bit in port A of the interface IC. This bit is connected directly to the blanking input of the SynDAC IC (described on page 25) and causes the PC's video to be blanked (held at 0.0 V). The port A bit also controls a multiplexer between the PC and Macintosh sync lines so that the video signal from the Macintosh host computer is sent to the monitor.

When the user switches the interface to the PC, the software on the Macintosh host computer writes black RGB values into all entries of the CLUT and sets the DC offset register in the DAC to make the black and blank levels equal (0.0 V). The port A bit is then switched so that the SynDAC unblanks the PC's video signal and the video multiplexer sends the PC's video signal to the monitor.

Monitors Supported

The main compatibility card has 512 KB of DRAM soldered on that provides all the standard VGA modes and some extended SVGA modes. No video DRAM expansion is provided because none is needed to meet full VGA compatibility. The VGA controller supports the 14-inch and 15-inch RGB Apple monitors as well as the standard VGA monitors. Table 2-4 summarizes the monitor sizes and display modes supported by the DOS compatibility subsystem.

Table 2-4 Monitors and display modes

Monitor size	Modes supported
Apple 14-inch	All VGA (modes 0–7, D–13h); SVGA 640 by 480 pixels (79h)
VGA	All VGA (modes 0–7, D–13h); SVGA 640 by 480 pixels (79h)
SVGA	All VGA (modes 0–7, D–13h); SVGA 640 by 480 pixels (79h) and 800 by 600 pixels (6Ah, 70h)
Apple multiple-scan 15-inch	All VGA (modes 0–7, D–13h); SVGA 640 by 480 pixels (79h) and 800 by 600 pixels (6Ah, 70h)

Monitor Sense Lines

The host computer detects the monitor type by way of sense lines in the video cable. Information about the monitor type is made available to the VGA driver so that it can program the card's video control registers appropriately.

Video Timing

Table 2-5 and Figure 2-2 on page 24 define the video monitors and timings supported by the Macintosh host computer that are also supported by DOS compatibility subsystem. For Macintosh monitors that are fixed frequency (the 14-inch and 16-inch monitors), the VGA controller on the card needs to be configured for this horizontal and vertical retrace rate.

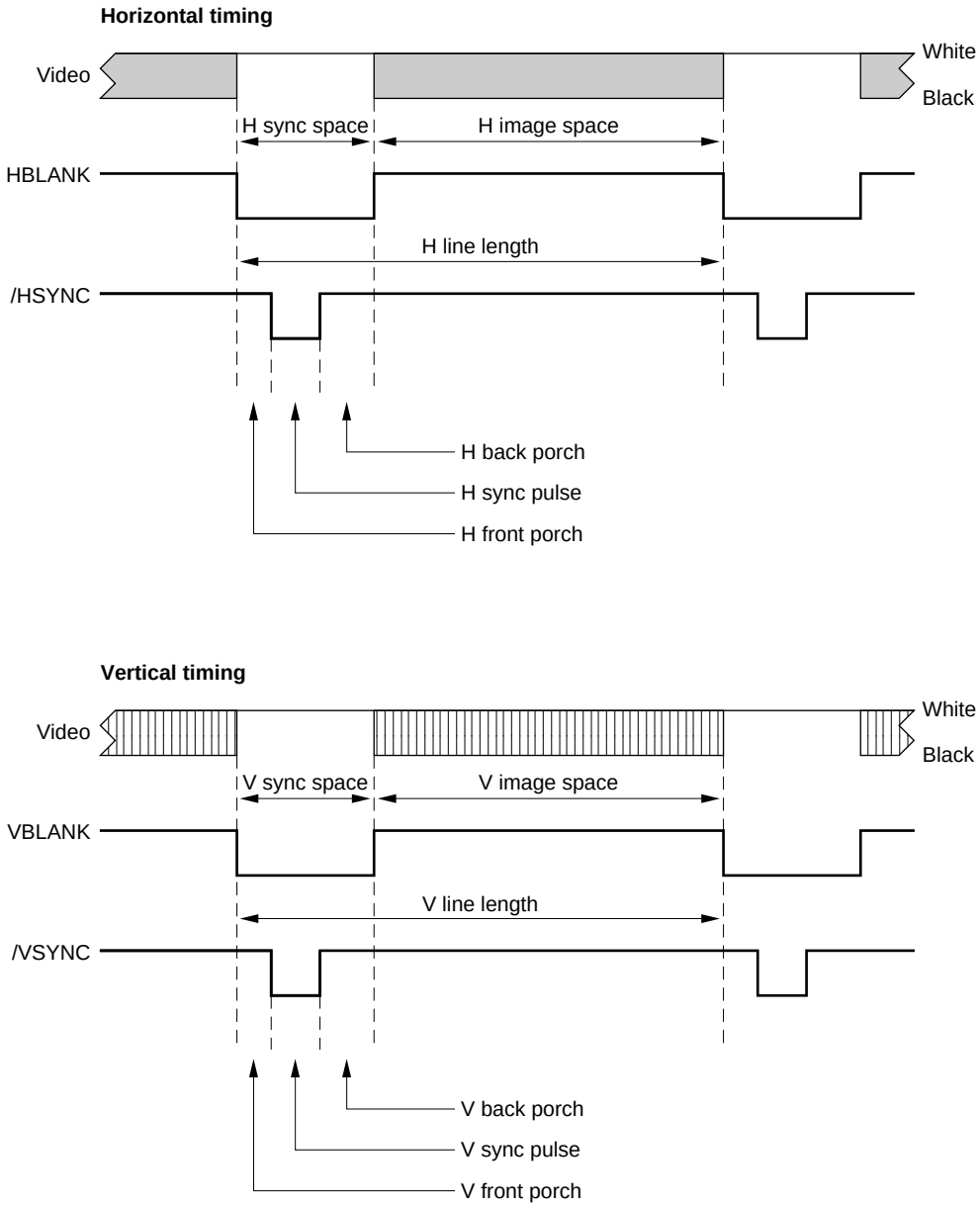
Table 2-5 VIdeo timing parameters for supported monitors

Parameter	14-inch RGB	16-inch RGB	VGA	SVGA
Display size (pixels)	640 by 480	832 by 624	640 by 480	800 by 600
Pixel clock	30.24 MHz	57.28 MHz	25.18 MHz	36.00 MHz
Pixel time	33.07 ns	17.46 ns	39.72 ns	27.78 ns
Line rate	35.00 kHz	49.73 kHz	31.47 kHz	35.16 kHz
Line time	28.57 µs (864 pixels)	20.11 µs (1152 pixels)	31.78 µs (800 pixels)	28.44 µs (1024 pixels)
Horizontal active video	640 pixels	832 pixels	640 pixels	800 pixels
Horizontal blanking	224 pixels	320 pixels	160 pixels	224 pixels
Horizontal front porch	64 pixels	32 pixels	16 pixels	16 pixels
Horizontal sync pulse	64 pixels	64 pixels	96 pixels	112 pixels
Horizontal back porch	96 pixels	224 pixels	48 pixels	96 pixels
Frame rate	66.72 Hz	74.55 Hz	59.94 Hz	52.71 Hz
Frame time	15.01 ms (525 lines)	13.41 ms (667 lines)	16.68 ms (525 lines)	18.97 ms (667 lines)
Vertical active video	480 lines	624 lines	480 lines	600 lines
Vertical blanking	45 lines	43 lines	45 lines	28 lines
Vertical front porch	3 lines	1 line	10 lines	1 line
Vertical sync pulse	3 lines	3 lines	2 lines	4 lines
Vertical back porch	39 lines	39 lines	33 lines	23 lines

Note

The DOS compatibility subsystem can operate with a 17-inch (or larger) monitor. With a large monitor, the user can open the Monitors control panel and set the display to either 640 by 480 pixels or 832 by 624 pixels. ◆

Figure 2-2 Video timing parameters



To accommodate the various VGA and SVGA modes on the Macintosh monitors, the video controller must have its timing parameters changed by the BIOS. To do that, the Macintosh software reads the video sense lines and loads the appropriate values for the video BIOS before starting up the PC. (Remember that system and video BIOS reside in Macintosh system memory and are modifiable by the software.)

IMPORTANT

With a shared monitor, modifying the video parameters by writing directly to the video control registers of the VGA controller can cause loss of video synchronization. ▲

Video Components

Two ICs provide the video support for the PC:

- 82C450 VGA controller
- MU9C9760 SynDAC

82C450 VGA Controller

The 82C450 is an integrated VGA video controller that is backward compatible with EGA, CGA, and MDA video modes. With the card's 512 KB of video DRAM (four 256K-by-4 DRAM ICs), the SynDAC supports all standard VGA modes as well as 800 by 600 pixels at 4 bits per pixel (noninterlaced), 640 by 480 pixels at 8 bits per pixel, and 132-column text mode.

The video controller is connected to the system through the ISA bus (the XD bus on the main compatibility card).

MU9C9760 SynDAC

Most of the video logic on the main compatibility card in provided by a device called the SynDAC: an IC (MU9C9760) that combines a lookup table, a triple video DAC, and a dual clock synthesizer. The SynDAC IC drives the video output line directly and is compatible with the Brooktree BT475 CLUT/DAC IC. The SynDAC IC provides 256 colors from a palette of 256K colors. The SynDAC IC also provides an internal pixel clock with eight programmable frequencies.

I/O Components

The I/O components in the DOS compatibility subsystem are the Pretzel Logic IC and the 8242 keyboard and mouse controller.

Pretzel Logic I/O Controller IC

The main I/O component in the DOS compatibility subsystem is the Pretzel Logic IC. It acts as a bus converter between the PC processor bus and the Macintosh processor bus. The Pretzel Logic IC integrates many of the I/O functions required to support the PC and also helps support the communication between the PC and the Macintosh host computer. The Pretzel Logic IC has the following features:

- two DMA channels (one for shared memory and one for disk I/O)
- address translation logic for 32-bit addresses and block sizes up to 64 MB
- two serial ports (16C450 compatible)
- one Centronics parallel printer port
- keyboard and mouse controller (8047 compatible)
- a64-bit message mailbox with a 32-bit command port
- power-on reset logic
- autoconfiguration logic

The Pretzel Logic IC functions as a slave device on the PC system bus. On the Macintosh system bus, the Pretzel Logic IC functions both as a slave and as an alternate bus master.

To the Macintosh host computer, the DOS compatibility subsystem appears as an I/O expansion card capable of generating slot $E interrupts to the 68040 and either responding as a system bus slave or becoming a system bus master. The Pretzel Logic IC communicates with the host computer as a bus master when the PC is performing floppy disk or hard disk accesses or when sharing Macintosh memory. The Pretzel Logic IC responds as a system bus slave on the Macintosh host computer during interrupt acknowledge cycles, keyboard and mouse accesses, and message mailbox accesses.

DMA Channels

When the DOS compatibility subsystem is configured to operate in shared memory mode (that is, when no SIMM is installed), the Pretzel Logic IC uses one of its DMA channels for access to memory in the Macintosh host computer. The DMA channel incorporates separate FIFOs for read and write operations; each FIFO is four longwords deep. The write FIFO allows the Cx486DX2 to post up to four longword writes before forcing the processor to wait.

The second DMA channel is used to perform I/O data transfers between Macintosh peripherals and PC memory. This I/O DMA channel is used when a DRAM SIMM is installed on the main compatibility card.

Address Translation

The address translation register provides 32-bit address translation between the PC and the host computer. This feature supports block sizes of 2 to 64 MB and allows the PC memory to be relocated anywhere within the unreserved memory area on the Macintosh host computer.

Note
The Pretzel Logic IC is not involved with address decoding for the declaration ROM; that decoding is provided by the Portola bus adapter. ◆

Serial Port Support

To support serial ports, the Pretzel Logic IC contains two identical sets of UART emulation registers. These registers emulate the hardware of the standard 16C450 serial port ICs found in many PC/AT computers. When the PC accesses these registers, interrupts are generated in the Macintosh host computer that cause the serial driver in the Mac OS to route the data to the Macintosh serial ports.

The Macintosh serial ports are RS-422 ports and do not support all RS-232 signals. In particular, the Carrier Detect (CD), Data Set Ready (DSR), Request To Send (RTS), and Ring Indicator (RI) signals are not available. Table 1-2 (in Chapter 1) shows the corresponding signals on the two types of serial ports.

Note
Not all RS-232 devices work properly using the RS-422 protocol. ◆

Printer Port Support

The Pretzel Logic IC implements all the registers of the standard Centronics parallel port found on a PC. When the PC accesses these registers, interrupts are generated in the Macintosh host computer that cause the driver software in the Mac OS to send data to a print spooler file. The spooler file is then sent to whatever printer is selected by the user in the Macintosh environment.

Note
The parallel port interface does not control printer hardware signals and does not support bidirectional data transfer. ◆

Keyboard and Mouse Emulation

The Pretzel Logic IC emulates in hardware the PC's keyboard and mouse. The 8242 keyboard and mouse controller is configured to support a PS/2 mouse making the protocol identical for the keyboard and mouse. The Pretzel Logic IC generates the appropriate serial clock protocol and serial bit stream to communicate with the 8242.

Message Mailbox

The message-passing interface in the Pretzel Logic IC supports simple interrupt-driven communication between the PC and the Macintosh host computer. The message-passing interface contains two data registers and one command register. One of the data registers incorporates byte swapping to allow address-invariant data to be moved between the two systems. The interface uses a semaphore mechanism of arbitration and grants to control the direction of the message passing. See "Passing Messages" beginning on page 45 for a description of the software API for message passing.

Power-on Reset

The Pretzel Logic IC contains the reset logic that allows the Macintosh host computer to start up the PC. Reset of the PC is controlled through the /PWRGOOD signal to the 84035 data path controller IC. Power for the PRAM on the PC is provided by the Macintosh computer, so the PRAM is not invalidated when the PC is reset. When the host computer is turned off, the PRAM becomes invalid; the next time the computer is turned on, software on the Macintosh side reloads the PRAM on the PC side before the PC system BIOS is executed.

Soft reset of the PC by way of the keyboard (Ctl-Alt-Del keys) is handled by the 8242 keyboard controller once the proper key code is sent by the Pretzel Logic IC through the keyboard port.

Autoconfiguration

The Pretzel Logic IC performs autoconfiguration each time the PC is reset. The following configurations are sensed and set upon reset:

- Presence of local DRAM (SIMM installed on the card)
- Card ID (001)

The card ID for the DOS compatibility subsystem in the Macintosh 630 DOS Compatible Computer is 001.

Game Adapter Card

The game adapter card contains the declaration ROM and the game controller port. The game adapter card occupies the I/O expansion slot.

The game controller port is a DB-15 connector for connecting a standard PC-style game controller (joystick). The game controller port occupies the I/O expansion opening in the back of the computer.

The declaration ROM is similar to the standard declaration ROM used for Macintosh expansion cards. The device used for the declaration ROM is a 32 KB ROM IC with an access time of 150 ns. The address decoding and the device select signal for the declaration ROM are provided by the Portola bus adapter IC.

Sound Expansion Card

The sound expansion card is plugged into the main compatibility card through a connector that provides a subset of the unbuffered XD bus.

The sound expansion card provides MPC level 1 and level 2 sound output capability. The card does not provide sound input capability; instead, the Macintosh host computer provides sound input and record features. The sound expansion card is compatible with the Sound Blaster register set and uses the standard ISA bus interface and 8-bit DMA channel.

The sound expansion card is designed around three ICs:

- CT2501 sound system IC
- YMF262 FM synthesizer IC
- TAC512 DAC IC

CT2501 Sound System IC

The CT2501 is a single IC that incorporates all the functions of a 16-bit PC sound system except FM synthesis and output filtering. The CT2501 sound system IC, also known as the Vibra 16, includes the following features:

- an 8-bit ISA bus interface including DMA support and interrupt generation
- FIFO buffers and control logic for digital audio playback and format conversion for the DAC
- a 16-bit stereo codec

- a Sound Blaster–compatible mixer with AGC
- a control interface for the FM synthesizer IC

The CT2501 sound system IC allows analog mixing of audio from the PC and from the FM synthesizer IC. The audio signal from the sound card is then mixed with the square wave sounds generated on the main compatibility card. The resulting sound signal is sent to the Macintosh host computer where it is mixed with the Macintosh computer's sound signals and sent to the sound outputs.

YMF262 FM Synthesizer IC

The YMF262 IC, a type I3 (OPL3) device, uses FM synthesis to generate sounds. The YMF262 IC includes the following features:

- 24 operators configurable in four-operator mode for 6 channels
- 36 operators configurable in two-operator mode for either 18 channels or 15 channels with 5 rhythm channels
- 8 selectable FM source waveforms
- 4 channels of sound output
- hardware vibrato and tremolo effects
- 2 programmable timers capable of generating interrupt requests

The YMF262 IC interfaces directly to the 8-bit ISA data and address bus; the CT2501 IC provides the chip select signal.

YAC512 Sound DAC IC

The YAC512 IC is a two-channel, 16-bit digital-to-analog converter that interfaces with the YMF262 FM synthesizer IC to provide analog sound output signals.

Subsystem Connectors

The DOS compatibility subsystem is connected to the host computer's main logic board by three connectors:

- the 68040 microprocessor socket
- the I/O expansion slot (PDS)
- the audio and video connector

The 68040 Microprocessor Socket

Most of the connections between the DOS compatibility subsystem and the host computer are made by way of the host computer's 68040 microprocessor socket. Most of

the signals on the card's 68040 socket are connected directly to the corresponding pins on the 68040 header on the card. A few of the signals from the 68040 are qualified by the Portola bus adapter and then sent on to the pins of the 68040 header.

The I/O Expansion Slot

Only 33 of the signals on the I/O expansion slot are used by the DOS compatibility subsystem. Those signals include

- the /PDS.DSACK0 handshaking signal, which is generated by the Portola bus adapter
- the address lines and the upper byte of the data bus, which are connected to the declaration ROM in the DOS compatibility subsystem

Note
The I/O expansion slot in the Macintosh 630–series computers is not a true PDS (processor-direct slot) because it is not connected directly to the computer's main processor. ◆

Table 2-6 shows the signals connected to the I/O expansion slot.

Table 2-6 Signals connected to the I/O expansion slot

Pin number	Signal name	Pin number	Signal name
A-2	/SLOTIRQ	B-10	D27
A-3	/PDS.AS	B-11	D24
A-9	D31	B-25	A2
A-10	D28	B-26	A12
A-11	D25	B-27	A13
A-18	A1	B-28	A8
A-25	A4	B-32	GND
A-26	A6	C-4	/PDS.SACK0
A-27	A11	C-9	D29
A-28	A9	C-10	D26
A-32	+12V	C-17	A0
B-3	+5V	C-25	A3
B-4	+5V	C-26	A5
B-6	GND	C-27	A7
B-7	CLK16M	C-28	A10
B-8	GND	C-32	-5V
B-9	D30		

Audio and Video Connector

A ribbon cable carries the audio and video signals from the main compatibility card to the main logic board in the host computer. Table 2-7 gives the signal assignments on the ribbon cable's 16-pin connector.

Table 2-7 Signals on the audio and video connector

Pin number	Signal	Pin number	Signal
1	Sound out R	2	Sound GND
3	Sound out L	4	GND
5	Video out red	6	GND
7	Video out green	8	GND
9	Video out blue	10	GND
11	CSYNC	12	GND
13	HSYNC	14	GND
15	VSYNC	16	RGB_Select

Note

The audio and video connector is a feature of the Macintosh 630 DOS Compatible Computer that is not present on other Macintosh 630–series computers. ◆

CHAPTER 3

The PC Interface Driver

The PC Interface driver provides communication and control between the Macintosh Operating System (Mac OS) and the DOS compatibility subsystem. Programs running on the Mac OS can use the driver to configure and control the card. Programs in both environments can use the driver to exchange messages; see the section "Passing Messages" beginning on page 45.

Initializing the Driver

The PC interface driver is named `.Symbiosis`. Before you can use the driver, your application must initialize it by calling the `open` routine. Both opening and closing the driver are performed only from programs running on the Mac OS.

Open

When you call the `open` routine, it allocates and initializes the driver's memory, installs the interrupt handler, and makes patches to the system needed by the driver. The `open` routine initializes all devices to the null device and puts the PC into the reset state.

The `open` routine fails if the driver cannot allocate enough memory or if it cannot find the DOS compatibility subsystem.

Close

When you call the `close` routine, it releases all memory allocated to the PC Interface driver, removes the driver's interrupt handler, removes any patches installed by the `open` routine, and puts the PC into the reset state.

Configuring the PC

A program running on the Mac OS can use the PC Interface driver to configure the PC on the DOS compatibility subsystem. You can use calls to the driver to perform the following operations:

- setting the memory available to the PC
- configuring the disk drives available to the PC
- setting and reading the status of the network driver
- configuring the communications port
- configuring the parallel port
- defining the key combination that deactivates the PC

The routines that perform those configuration tasks are defined here.

rsSetMemoryConfig

You can use the `rsSetMemoryConfig` control call to make memory on the Macintosh computer available for the PC. The calling program first allocates the memory and sets it locked and contiguous. The control call sets the base address and length of the memory.

This call is needed only when no RAM SIMM is installed for the PC. The calling program can determine whether a RAM SIMM is installed by calling the `rsPCStatus` status routine (described below).

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsSetMemoryConfig`
→	`csParam+0`	`long`	Logical base address of PC memory
→	`csParam+4`	`long`	Physical base address of PC memory
→	`csParam+6`	`long`	Length of PC memory

rsSetDriveConfig

You can use the `rsSetDriveConfig` control call to configure each of the PC's fixed disk drives (A:, B:, C:, and D:) as a floppy drive, Macintosh file, or SCSI partition, or as having no corresponding drive.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsSetDriveConfig`
→	`csParam+0`	`long`	Pointer to `RSFixedDriveConfig`

The `csParam` contains a pointer to an `RSFixedDriveConfig` data structure.

```
typedef  struct{
   short type;       // Type of device this drive is
   short vRefNum;    // Volume refNum or SCSI ID
   long  dirID;      // Directory ID or starting sector number on hard drive
   long  fileNamePtr;// Filename or number of sectors on hard drive
} RSFixedDriveConfig[4], *RSFixedDriveConfigPtr;
```

`RSFixedDriveConfig[0]` contains the configuration for drive A:, `RSFixedDriveConfig[1]` contains the configuration for drive B:, `RSFixedDriveConfig[2]`contains the configuration for drive C:, and `RSFixedDriveConfig[3]` contains the configuration for drive D:.

The `type` field specifies what type the drive is configured as (`rsFloppyDrive`, `rsFileDrive`, `rsPartitionDrive`, or `rsNULLDrive`).

If the value of `type` is `rsNULLDrive`, the corresponding drive does not exist to the PC and no other fields need to be filled in.

If the value of `type` is `rsFloppyDrive`, the corresponding drive on the PC is connected to one of the Macintosh computer's floppy drives.

If the value of `type` is `rsFileDrive`, the corresponding drive is connected to a Macintosh file system file. The `vRefNum` field contains the volume the file is on, `dirID` contains the directory ID of the file, and `fileNamePtr` contains a pointer to the file name. The driver opens and closes the file as needed.

If the value of `type` is `rsPartitionDrive`, the corresponding drive is connected to a SCSI drive partition. The `vRefNum` field contains the SCSI ID, `dirID` contains the starting sector number of the partition, and `fileNamePtr` contains the number of sectors in the partition.

If the value of `type` is set to `rsIgnore`, the configuration of the corresponding drive is not changed.

The program on the Macintosh computer should call `rsSetDriveConfig` at least once before starting the PC. The routine can also be called after the PC has been started to change the drive configuration. In that case, the new drive configuration does not take effect until the PC is restarted.

rsGetNetDriveConfig

You can use the `rsGetNetDriveConfig` status call to obtain drive configuration data. This call returns a pointer to an array of 22 `RSNetDriveConfig` data structures, one for each drive letter from *E* through *Z*.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsGetNetDriveConfig`
←	`csParam+0`	`long`	Pointer to `RSNetDriveConfig`

```
typedef struct {
   char  status;  // 0 = unused, -1 = in use, 1 = cannot be used
   char  changed; // Used by the driver, do not use
   short vRefNum; // Reference number of volume containing shared drive
   long  dirID;   // Directory ID
} RSNetDriveConfig[26], *RSNetDriveConfigPtr;
```

The `RSNetDriveConfig` data structure contains the current configuration for folder sharing for each PC drive letter. If the PC has its `LASTDRIVE` parameter set to less than Z or if other block device drivers are loaded on the PC, not all drive letters will be available. The data structures for drives that are not available have their status parameters set to 1 by the PC Interface driver.

The caller can use the returned pointer to modify an entry in the `RSNetDriveConfig` data structure and then call the `rsSetNetDriveConfig` control call.

rsSetNetDriveConfig

You can use the `rsSetNetDriveConfig` control call to establish links between Macintosh directories and PC drive letters.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsSetNetDriveConfig`
→	`csParam+0`	`word`	Entry number of `RSNetDriveConfig (0=E)`

This call simply notifies the PC Interface driver that an entry in the `RSNetDriveConfig` data structure has been modified.

rsSetComPortConfig

You can use the `rsSetComPortConfig` control call to set the configurations of the two communication ports (COM1 and COM2) on the PC. Each communication port can have a virtual connection to either the modem port, the printer port, a communication tool box port, a spool file, or the null device.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsSetComPortConfig`
→	`csParam+0`	`long`	Pointer to `RSComConfig`

A pointer to an `RSComConfig` data structure is passed in the `csParam` field.

```
typedef  struct{
   short    type;       // Port type (rsModemComPort, rsPrinterComPort, etc.)
   short    vRefNum;    // Volume reference number for serial spool file
   long     dirID;      // Directory ID
   long     fileNamePtr;// Pointer to the filename
} RSComConfig[2], *RSComConfigPtr;
```

`RSComConfig[0]` contains the configuration for COM1 and `RSComConfig[1]` contains the configuration for COM2. The `type` field specifies what type of connection to make (either `rsNULLComPort`, `rsModemComPort`, `rsPrinterComPort`, `rsSpoolComPort`, `rsComToolBoxComPort`, or `rsIgnore`). The value of the `vRefNum` parameter is the volume reference number, `dirID` is the directory ID, and `fileNamePtr` is the pointer to the name of the spool file.

When a PC port is connected to the null device, any output from the PC is ignored.

When a PC port is connected to the modem or printer port, the PC controls the port by means of the UART emulation register in the DOS compatibility subsystem. For example, when the PC sets the baud rate divisor in the UART emulation register, the PC Interface driver intercepts the operation and translates the action to a control call to the driver for the modem or printer port.

When a PC port is connected to a spool file, all output from the PC is captured and written to the specified file. The driver opens and closes the file as needed.

The `rsSetComPortConfig` routine should be called at least once before the PC is started up. It can also be called after the PC has been started; in that case, the change in configuration takes effect immediately.

If the `type` field is set to `reIgnore`, the port's configuration does not change.

rsSetParallelPortConfig

You can use the `rsSetParallelPortConfig` function to set the configuration of the parallel port emulation. A pointer to an `RSParallelConfig` data structure is passed in `csParam`.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsSetParallelPortConfig`
→	`csParam`	`long`	Pointer to `RSParallelConfig`

```
typedef  struct{
   short eojTimeOut; // End of job after n seconds of no data
   short vRefNum;    // Volume RefNum
   long  spoolDirID; // RefNum for spool directory
} RSParallelConfig, *RSParallelConfigPtr;
```

The `spoolDirID` field is the ID of the directory where the spool files will be stored. The `vRefNum` field contains the reference number of the volume that contains the directory. The `eojTimeOut` field specifies the number of seconds the parallel port must be inactive before the driver will force an end of job. If this field is set to 0, the driver does not force the end of job based on time.

When a print job has been completed, the driver notifies the application by means of the `rsSetNotificationProc` procedure (defined on page 44). The driver also notifies the application if it has trouble saving the spool data.

rsSetDeactivateKey

You can use the `rsSetDeactivateKey` control call to set the deactivate key along with its modifiers and a user-defined task. When the PC has control of the keyboard, the driver monitors the keyboard input data for the deactivate key combination and calls the user-defined task when that key combination occurs.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsSetDeactivateKey`
→	`csParam+0`	`long`	Pointer to user-defined task
→	`csParam+4`	`word`	Modifiers
→	`csParam+6`	`word`	The deactivate key

Upon return, the parameter block is set as follows:

←	`csParam+0`	`long`	Pointer to the previous user-defined task
←	`csParam+4`	`word`	The previous modifiers
←	`csParam+6`	`word`	The previous deactivate key

The user-defined task is called during `NeedTime` after the deactivate key and modifiers are pressed. If the user-defined task is null, no task is called. The modifiers are specified as they appear in the `KeyMap+6`. The value of the deactivate key is the Macintosh key code of the desired key.

Control and Status Calls

A program running on the Mac OS can use the PC Interface driver to make control and status calls to the PC running on the DOS compatibility subsystem. You can perform the following functions:

- getting the status of the PC
- booting (starting) the PC
- resetting the PC
- enabling and disabling the video display of the PC
- enabling and disabling disk mounting on the PC
- activating and deactivating keyboard operation by the PC
- activating and deactivating mouse tracking by the PC
- terminating print spooling from the PC

rsPCStatus

You can use the `rsPCStatus` status call to get information about the state of the PC hardware. This call returns the current state of the PC.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsPCStatus`
←	`csParam+4`	`long`	The status word

Table 3-1 shows the meanings of the bits in the status word.

Table 3-1 Bits in the PC status word

Bit	Meaning
0	1 = PC is running (`rsBooted`)
1	1 = VGA screen is enabled (`rsVGAEnabled`)
2	1 = keyboard is enabled (`rsKeyboardEnabled`)
3	1 = mouse is enabled (`rsMouseEnabled`)
4	1 = disk mounting is enabled (`rsDiskMountEnabled`)
5	1 = shared memory is enabled (`rsSharedEnabled`)
6	1 = DMA is enabled (`rsDMAEnabled`)
7	1 = video cable is enabled (`rsCableInstalled`)
8	1 = modem port is used by COM1
9	1 = printer port is used by COM1
10	1 = modem port is used by COM2
11	1 = printer port is used by COM2
24–27	0000–1111 = video identification
28–31	0000–1111 = type of expansion card (0001 = this card)

rsBootPC

You can use the `rsBootPC` control call to start up the PC. This call resets the PC's processor and boots the PC's system BIOS. If the PC is already running, this call resets it.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsBootPC`

The calling program must set up the PC's configuration before booting the PC. You can use the following control calls (defined previously) to set the configuration:

- `rsSetMemoryConfig`
- `rsSetDriveConfig`
- `rsSetComPortConfig`
- `rsSetParallelConfig`

rsResetPC

You can use the `rsResetPC` control call to put the PC into a reset state. This call stops the PC from running; any programs or data in the PC's memory are lost. The calling program must use the `rsBootPC` control call to start the PC running again.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsResetPC`

rsEnableVideo

You can use the `rsEnableVideo` control call to enable the VGA display output. You use the call when switching the video monitor from the Mac OS to the PC.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsEnableVideo`

rsDisableVideo

You can use the `rsDisableVideo` control call to disable the VGA display output when the Macintosh video output is selected.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsDisableVideo`

rsMountDisks

You can use the `rsMountDisks` call to enable the mounting and unmounting of PC disks. After the call has been made, the PC Interface driver monitors all disk-insertion events, looking for possible PC formatted disks. If the inserted disk is not a Macintosh formatted disk, it is considered a PC disk and is made available to the PC if the PC is active. The mounting and unmounting of the PC disks happens automatically; the `rsMountDisks` call merely enables the process.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsMountDisks`

rsDontMountDisks

You can use the `rsDontMountDisks` control call to stop the PC Interface driver from monitoring disk-insertion events. If the PC Interface driver has already mounted a PC disk before you make this call, the PC disk remains in the drive and available to the PC.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsDontMountDisks`

rsActivateKB

You can use the `rsActivateKB` control call to direct the data from the computer's keyboard to the PC side. All keys except the Command key are trapped; key codes are translated and transmitted to the PC.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsActivateKB`

rsDeactivateKB

You can use the `rsDeactivateKB` control call to stop the transmission of keyboard data to the PC and direct the keyboard data to the Mac OS.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsDeactivateKB`

rsBeginMouseTracking

You can use the `rsBeginMouseTracking` control call to cause the mouse movements and button presses to be directed to the PC. This call also causes the driver to hide the Macintosh cursor.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsBeginMouseTracking`

rsEndMouseTracking

You can use the `rsEndMouseTracking` control calls to cause the mouse movements and button presses to be directed to the Mac OS. This call also causes the driver to show the Macintosh cursor.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsEndMouseTracking`

rsEndPrintJob

You can use the `rsEndPrintJob` control call to end the current print job and close the spool file (if any). Any subsequent data from the PC to the parallel port starts a new spool file.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsEndPrintJob`

Detecting Errors

Programs on the Mac OS can use the next two procedures to detect error conditions or other special events on the PC.

rsSetNotificationProc

You can use the `rsSetNotificationProc` control call to install a user-defined procedure that is called whenever a special event happens within the driver. The procedure can be called at interrupt time; it is responsible for deferring handling of the event until noninterrupt time.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsSetNotificationProc`
→	`csParam+0`	`long`	Pointer to the notification procedure
→	`csParam+4`	`long`	A1Param value

Upon return, the parameters are set as follows:

←	`csParam+0`	`long`	Pointer to the previous notification procedure
←	`csParam+4`	`long`	Previous `A1Param` value

The caller passes a pointer to the user-defined procedure and a parameter to be passed to that procedure in A1. The control call returns the previous values. Calling `rsSetNotificationProc` with a `NULL` pointer disables the notification procedure.

When the user-defined procedure is called, the D0.w register contains the event and A1 contains the `A1Param` value. The procedure can use registers D0–D2 and A0–A1.

The events are

`rsPrintSpoolErr` = problem opening or writing to a print spool file

`rsCOM1SpoolErr` = problem opening or writing to the COM1 spool file

`rsCOM2SpoolErr` = problem opening or writing to the COM2 spool file

`rsDiskFileErr` = problem reading the disk file

rsLastError

You can use the `rsLastError` status call to obtain the last nonzero error code returned by the driver.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsLastError`
←	`csParam+4`	`long`	Pointer to the last error routine

Passing Messages

Programs on the Mac OS and the PC can send messages to each other by calling the PC Interface driver. Programs can also install a receive procedure for receiving messages. When the PC Interface driver receives a message intended for your program, the driver calls your receive procedure. Your procedure decides whether or not to accept the message's data and, if so, where to store the data.

Message Conventions

Before communications can take place, a program on the Mac OS and a program on the PC must have the same definitions of the messages they transfer. A message consists of a 16-bit command, two 32-bit parameters, and up to 64 KB of data. The parameters and the data can consist of any data in any format. The command must be a unique value recognized by the programs on the Mac OS and the PC that are sending and receiving messages. The programs on both the PC and the Mac OS must request command numbers from the PC Interface driver before sending messages.

Macintosh Interface

Programs on the Mac OS communicate with the PC Interface driver through driver calls. Your program should first open the driver using the `open` call and then use the control calls defined in the next section to register, send, and receive messages.

PC Interface

Programs on the PC communicate with the PC Interface driver through a software interrupt interface. The program loads registers with appropriate values, including a function selector in register AH, and calls the PC Interface driver with an `INT 5Fh` call. PC programs can determine whether the PC Interface driver interface is available by calling `INT 5Fh` with register AH = 0. If the PC Interface driver is installed, it returns `0A5h` in register AH and the highest implemented function code (currently 4) in register AL.

Registering Messages

For a program on the Mac OS to send messages to a program on the PC, both programs must register their messages with the PC Interface driver. This is done by calling the driver with a 32-bit selector defined in both programs and a count of the number of messages to be used by the programs. The PC Interface driver allocates a range of messages for that selector and returns the base command number to the caller. The PC Interface driver makes sure that both the PC program and the Macintosh program registering messages under the same selector will receive the same base command number.

On the Mac OS

To register your messages from a Macintosh program, make an `rsRegisterMessage` control call with the message selector in `csParam+0` and the number of message commands to allocate in `csParam+4`.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsRegisterMessage`
↔	`csParam+0`	`long`	32-bit message selector
→	`csParam+4`	`long`	Number of message commands to allocate

The PC Interface driver returns the base command number in `csParam+0`. If the PC Interface driver cannot allocate the messages, an error code is returned in `ioResult`.

On the PC

To register your messages from a PC program, load the 32-bit selector into register EBX and the message count in register CX; then call `INT 5Fh` with AH = 4. The PC Interface driver returns the base command number in register BX. Register AH contains an error code if the messages could not be allocated.

Sending a Message

To send a message, you must pass a message parameter block (`MsgPBlk`) to the PC Interface driver. The `rsSendMessage` routine is always asynchronous; it simply queues the message parameter block and returns to the caller. The `msgResult` field is set to 1 (busy) until the message has been sent.

After the message has been sent, the `msgResult` field is set to 0 (no error) or –3 (`MsgTimeout`). The `msgActCount` field contains the number of bytes actually sent. If you have specified a completion routine, it is then called.

On the Mac OS

The `MsgPBlk` data structure for programs on the Mac OS has the following format:

```
MsgPBlk         RECORD   0
msgQLink        DS.l     1    ; Next queue element
msgQType        DS.w     1    ; Queue flags
msgCmd          DS.w     1    ; The message type or command
msgParam1       DS.l     1    ; Message parameter 1
msgParam2       DS.l     1    ; Message parameter 2
msgBuffer       DS.l     1    ; Pointer to the message data buffer
msgReqCount     DS.l     1    ; Requested data length
msgActCount     DS.l     1    ; Actual data length
msgCompletion   DS.l     1    ; Pointer to completion routine or
                                NULL
msgResult       DS.w     1    ; The result of any message operation
msgFlags        DS.w     1    ; Message Flags (Swap and Shared)
                                Set to zero!
msgUserData     DS.l     1    ; For the callers use
MsgPBlkSize     Equ      *    ; Size of record
                ENDR
```

To send a message, build a `MsgPBlk` and then pass the pointer to the `MsgPBlk` to the PC Interface driver in an `rsSendMessage` control call.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsSendMessage`
→	`csParam+0`	`long`	Pointer to `MsgPBlk`

Your completion routine is called at Deferred time and can use registers D0–D2 and A0–A1. You must save all other registers. Upon return, A0 contains a pointer to the `MsgPBlk` structure.

On the PC

The `MsgPBlk` data structure on the PC has the following format. Please note that the sizes of some of the fields are different from the Mac OS equivalent.

```
MsgPBlk         STRUCT
link            DWORD    ?    ; Link to next queue element
msgCmd          WORD     ?    ; The message command or type
msgParam1       DWORD    ?    ; Param 1
msgParam2       DWORD    ?    ; Param 2
msgBuffer       DWORD    ?    ; Pointer to the data buffer
msgReqCount     DWORD    ?    ; Length of the data
msgActCount     DWORD    ?    ; # of bytes actually transferred
msgCompletion   DWORD    ?    ; Pointer to the completion routine
msgResult       BYTE     ?    ; The error code after complete or 1
msgFlags        BYTE     ?    ; Msg flags (Shared and Swapped)
                                 set to zero!
msgUserData     DWORD    ?    ; For the caller's use
msgVXD          DWORD    ?    ; Reserved for driver use
MsgPBlk         ENDS
```

To send a message on the PC, build a `MsgPBlk` structure and call the PC Interface driver with AH = 1 (`rsSendMessage`) and ES:BX = the pointer to the `MsgPBlk` structure. When you execute an `INT 5Fh`, the message is queued, `msgResult` is set to 1 (busy), and control returns to your program.

Your completion routine is called with a FAR call and it should return with an `RETF`. Also, your routine may use registers AX, BX, CX,DX, DI,SI, ES, and DS. When your completion routine is called, ES:BX is a pointer to the `MsgPBlk` structure.

Installing a Message Handler

Before you can receive messages, you must install a message handler. The PC Interface driver calls the message handler when the driver receives a message with a command value greater than or equal to `recCmdBase` and less than `recCmdBase` + `recCmdCount` in the `MsgRecElem` data structure. The driver passes the message's 16-bit command and the two 32-bit parameters to your message handler.

The message handler examines the command and parameters and determines whether there is any data to be received. If there is, the handler passes back a pointer to a `MsgPBlk`. The PC Interface driver then receives the data and puts it into the buffer pointed to by `msgBuffer`. The driver then updates `msgActCount` with the number of bytes of data received and sets `msgResult` to 0 (no error), –1 (`MsgOverrun`), –2 (`MsgUnderrun`), or –3 (`MsgTimeout`). The driver then calls your completion routine, if there is one.

A message handler is described by a `MsgRecElem` record. The `recProc` field points to the handler procedure; the values of `recBaseCmd` and `recCmdCount` are the values allocated by `rsRegisterMessage`.

IMPORTANT

Before your program terminates, you must remove your message handler so that the PC Interface driver will not call it after you are gone. See the section "Removing a Message Handler" on page 50. ▲

On the Mac OS

The `MsgRecElem` data structure for programs on the Mac OS has the following format:

```
MsgRecElem     RECORD   0
recQLink       DS.l     1    ; Next queue element
recQType       DS.w     1    ; Queue flags
recFlags       DS.w     1    ; Not used...yet...set to zero
recProc        DS.l     1    ; Pointer to the receive procedure
recCmdBase     DS.w     1    ; First command received by this
                                procedure
recCmdCount    DS.w     1    ; Number of commands allocated for
                                this procedure
recUserData    DS.l     1    ; For caller's use (could be A5...)
MsgRecElemSize Equ      *
               ENDR
```

To install a message handler on the Mac OS, build a `MsgRecElem` record and pass a pointer to it in a control call to the PC Interface driver.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsInstallMsgHandler`
→	`csParam+0`	`long`	Pointer to `MsgRecElem`

When your message handler procedure is called, D0.w contains the message command, D1.l contains the `msgParam1` value, D2.l contains the `msgParam2` value, and A1 contains a pointer to the `MsgRecElem` record. Your routine must pass back a pointer to a `MsgPBlk` structure in A0 if you wish to receive the message data; otherwise, return 0 in A0. The handler procedure is called at interrupt time with interrupts masked at the slot interrupt level. It can use registers D0–D2 and A0–A1.

The completion routine for the `MsgPBlk` returned by the receive procedure is called at deferred time and can use registers D0–D2 and A0–A1. You must save all other registers. Upon return, A0 contains a pointer to the `MsgPBlk` structure.

On the PC

For a program on the PC, the `MsgRecElem` data structure has the following format:

```
MsgRecElem      STRUCT
Link            DWORD    ?    ; Pointer to next link
Code            DWORD    ?    ; Pointer to the code for this link
cmdBase         WORD     ?    ; Base message number for this
                                procedure
cmdCount        WORD     ?    ; Number of message numbers for this
                                procedure
userData        DWORD    ?    ; For caller's use
msgVXD          DWORD    ?    ; Reserved for driver use
MsgRecElem      ENDS
```

To install a message handler on the PC, build a `MsgRecElem` record and call `INT 5Fh` with AH = 2 and ES:BX containing a pointer to the `MsgRecElem` structure.

When your message handler is called, AX contains the message command, ECX contains `msgParam1`, EDX contains `msgParam2`, and ES:DI contain a pointer to the `MsgRecElem` record. Your program must pass a pointer to a `MsgPBlk` structure in ES:BX if you wish to receive the message data; otherwise, it must return 0 in BX. The handler is called at interrupt time with interrupts turned off. It can use registers AX, BX, CX,DX, DI, SI, ES, and DS.

The completion routine for the `MsgPBlk` structure returned by the receive procedure is called at interrupt time and can use registers AX, BX, CX,DX, DI, SI, ES, and DS. You must save all other registers. Also, ES:BX contain a pointer to the `MsgPBlk` structure.

Removing a Message Handler

Message handlers can be called until they are removed. Before your program terminates, you must remove the handler so that the PC Interface driver will not call it after your program is gone.

On the Mac OS

To remove a message handler on the Mac OS, your program makes an appropriate control call to the PC Interface driver and passes it a pointer to the handler.

Parameter block

→	`ioCompletion`	`long`	Pointer to the completion routine
←	`ioResult`	`word`	
→	`ioRefNum`	`word`	
→	`csCode`	`word`	Equals `rsRemoveMsgHandler`
→	`csParam+0`	`long`	Pointer to `MsgRecElem`

On the PC

To remove a message handler on the PC, your program makes a call to `INT 5Fh` with AH = 3 and with a pointer to the `MsgRecElem` record in registers ES:BX.

Header File for PC Interface

Here is a sample header file for access to the PC interface driver.

```
/*
   File: PCCardCalls.h

   Contains:This file contains the data structures and equates needed to
            call the PC Card driver on the Macintosh side.

   Copyright:© 1994 by Apple Computer, Inc., all rights reserved.

*/


#ifndef __PCCARDCALLS__
#define __PCCARDCALLS__

/
*-------------------------------------------------------------------------------
;                        Other Header Files
;-------------------------------------------------------------------------------*/

#ifndef __TYPES__
#include <Types.h>
#endif

#ifndef __OSUTILS__
#include <OSUtils.h>
#endif

#ifndef __EVENTS__
#include <Events.h>
#endif
```

```
/
*-------------------------------------------------------------------------
;                       Misc. Equates
;-------------------------------------------------------------------------*/

#defineRSDriverName"\p.RoyalScam"// The name of the driver

/
*-------------------------------------------------------------------------
;                 Error Codes Returned from Control/Status Calls
;-------------------------------------------------------------------------*/

enum{
   rsFirstErr        = 0x7000,   // first error code
   rsAlreadyBooted   = 0x7001,   // PC is already booted, can't complete
                                    operation
   rsLastErr         = 0x7001    // last error code
};

/
*-------------------------------------------------------------------------
;                 Notifications Codes Sent to Notification Proc
;-------------------------------------------------------------------------*/

enum{
   rsPrintSpoolErr         = 0x7F00,   // Having trouble opening/writting the
                                          print spool file
   rsPrintSpoolFileReady   = 0x7F01,   // at least 1 spool file is ready to
                                          be printed
   rsCOM1SpoolErr          = 0x7F01,   // Having trouble opening/writting the
                                          serial spool file
   rsCOM2SpoolErr          = 0x7F02    // Having trouble opening/writting the
                                          serial spool file
};

/
*-------------------------------------------------------------------------
;                   Control/Status ParamBlock Record
;-------------------------------------------------------------------------*/

typedefstruct {   // PC Card driver Control/Status ParamBlock Record
   QElemPtr    qLink;         // queue link in header
   short       qType;         // type byte for safety check
   short       ioTrap;        // FS: the Trap
   Ptr         ioCmdAddr;     // FS: address to dispatch to
```

```
   ProcPtr     ioCompletion;  // completion routine addr (0 for synch calls)
   OSErr       ioResult;      // result code
   StringPtr   ioNamePtr;     // ptr to Vol:FileName string
   short       ioVRefNum;  // volume refnum (DrvNum for Eject and MountVol)
   short       ioCRefNum;     // refNum for I/O operation
   short       csCode;        // The operation code
   void *      csPtr;         // pointer to proceedure or data
   long        csData;        // data
   long        csData2;       // data
} RSParamBlockRec, *RSParamBlockRecPtr;

/
*-------------------------------------------------------------------------
;                          PC Control
;-------------------------------------------------------------------------*/

// Control Codes

enum{
   rsBootPC          = 701,   // Boots the PC
   rsResetPC         = 702,   // Resets the PC
   rsWriteCMOS       = 703,   // Writes the CMOS RAM values
   rsReadCMOS        = 704,   // Reads current CMOS RAM values
   rsEnableVideo     = 705,   // Enables the VGA output
   rsDisableVideo    = 706,   // Disables the VGA output
   rsSetMemoryConfig = 707,   // Sets the shared memory config
   rsHaltPC          = 708,   // Stops the PC
   rsResumePC        = 709    // Allows PC to continue
};

// Status Codes

enum{
   rsPCStatus  = 701,      // Returns driver state information
   rsLastError = 702       // Returns the last non-zero error
};

// PC Status Masks

enum{
   rsBooted             = 1,     // Mask for boot state
   rsVGAEnabled         = 2,     // Mask for VGA output state
   rsKeyboardEnabled    = 4,     // Mask for keyboard state
   rsMouseEnabled       = 8,     // Mask for mouse tracking state
   rsDiskMountEnabled   = 16,    // Mask for Disk Mounting state
```

```
   rsSharedEnabled      = 32,    // Mask for Shared Memory Enabled
   rsDMAEnabled         = 64,    // Mask for DMA Enabled
   rsCableInstalled     = 128,   // Mask for video Cable Installed
   rsModemUsedByCom1    = 256,   // Mask for modem use by com1
   rsPrinterUsedByCom1  = 512,   // Mask for printer use by com1
   rsModemUsedByCom2    = 1024,  // Mask for modem use by com2
   rsPrinterUsedByCom2  = 2048,  // Mask for printer use by com2
   rsSoundEnabled       = 4096,  // Mask for Sound Enabled
   rsBIOSModified       = 8192   // Mask for Bios modified by Driver
};

/
*----------------------------------------------------------------------------
;                          Keyboard
;----------------------------------------------------------------------------*/

// Control Codes

enum{
   rsActivateKB         = 102,   // Tells the driver keyboard is now active
   rsDeactivateKB       = 103,   // Tells the driver keyboard is no longer
                                    active
   rsSetDeactivateKey   = 104,   // Sets the key that causes the deactivate
                                    routine to be called
   rsSetKeyMap1         = 105,   // Sets the scan code map #1
   rsSetKeyMap2         = 106,   // Sets the scan code map #2
   rsSetKeyMap3         = 107,   // Sets the scan code map #3
   rsSetKeyMap          = 108    // Sets the Mac to PC map
};

// Data Structures

typedef  char RSKeyMap[128];     // KeyMap data structure
typedef  char *RSKeyMapPtr;

typedefstruct {            // Scan Code Map data structure
   char  length;           // Length of the scan code (# of bytes)
   char  code[6];          // Scan code
} RSScanCodeMap[128], *RSScanCodeMapPtr, **RSScanCodeMapHdl;
```

```
/
*---------------------------------------------------------------------------
;                          Mouse
;---------------------------------------------------------------------------*/

// Control Codes

enum{
   rsSetMouseButtonKey     = 200,      // Sets which key to use as the
                                          second mouse button
   rsBeginMouseTracking    = 201,      // Tells the driver to track the
                                          mouse movement
   rsEndMouseTracking      = 202       // Releases control of the mouse
};

/
*---------------------------------------------------------------------------
;                          Serial IO
;---------------------------------------------------------------------------*/

// Com port Indexs

enum{
   rsCom1 = 0,    // Index for Com1
   rsCom2 = 1     // Index for Com2
};

// Device Types

enum{
   rsNULLComPort        = 0,  // Com port is connected to NULL (bit bucket)
   rsModemComPort       = 1,  // Com port is connected to Modem port
   rsPrinterComPort     = 2,  // Com port is connected to Printer port
   rsSpoolComPort       = 3,  // Com port data dumped into a file
   rsComToolBoxComPort  = 4,  // Com port is connected to Com Tool Box port
   rsIgnore             = -1  // Do not change this port
};

// Control Codes

enum{
   rsSetComPortConfig = 300   // Sets the connection and flow control for
                                 a port
};
```

```
// Parameter Block data structures

typedefstruct{
   short    type;          // The port type (rsModemComPort,
                              rsPrinterComPort, etc..)
   short    vRefNum;       // Volume reference number for serial spool file
   long     dirID;         // Directory ID for the file
   long     fileNamePtr;   // Pointer to the file name
} RSComConfig[2], *RSComConfigPtr;

/
*---------------------------------------------------------------------------
;                          Parallel IO
;---------------------------------------------------------------------------*/

// Control Calls

enum{
   rsSetParallelConfig  = 400,// Sets the configuration for the parallel port
   rsEndPrintJob        = 401 // forces an end to the current print job
};

// Parameter Block Data Structures

typedef  struct{
   short    eojTimeOut;    // Signal End of job after n seconds of no data
   short    vRefNum;       // Volume RefNum of the Mac Volume the dir is on
   long     spoolDirID;    // RefNum for spool directory
} RSParallelConfig, *RSParallelConfigPtr;

/
*---------------------------------------------------------------------------
;                          Fixed Drive IO
;---------------------------------------------------------------------------*/

// Fixed Drive Types

enum{
   rsNullDrive       = 0,  // No drive available
   rsFloppyDrive     = 1,  // Drive is a super drive
   rsFileDrive       = 2,  // Drive is a FS file
   rsPartitionDrive  = 3   // Drive is a partition defined elsewhere
   rsIgnore          = -1  // Don't change this drive
};
```

```
// Fixed Drive Array Index Numbers

enum{
   rsDriveA = 0,  // Floppy Drive A:
   rsDriveB = 1,  // Floppy Drive B:
   rsDriveC = 2,  // Hard Drive C:
   rsDriveD = 3   // Hard Drive D:
};

// Control Codes

enum{
   rsSetDriveConfig  = 500,   // Initialize the Drive Configuration
   rsMountDisks      = 501,   // Mount any disk that is inserted
   rsDontMountDisks  = 502    // Do not mount any disk that is inserted
};

// Fixed Disk Data Structures

typedef  struct{
   short    type;             // what type of device this drive is
   short    vRefNum;          // Volume refNum or SCSI ID
   long     dirID;            // Directory ID or starting sector number
                                 for SCSI
   long     fileNamePtr;      // Ptr to file name or # of sectors if SCSI
} RSFixedDriveConfig[4], *RSFixedDriveConfigPtr;

/
*-------------------------------------------------------------------------
;                          Network Disk IO
;-------------------------------------------------------------------------*/

// Control Codes

enum{
   rsSetNetDriveConfig = 600  // Set Net Drive config
};

// Status Codes

enum{
   rsGetNetDriveConfig = 650  // Get Drive Letter Info
};
```

```
// Data Structure

typedef  struct{
   char     status;     // 0 = unusable, - = inuse, + = can be used
   char     changed;    // Used by driver!
   short    vRefNum;    // which mac volume (0 = no net drive)
   long     dirID;      // the Directory ID
} RSNetDriveConfig[26], *RSNetDriveConfigPtr;

/
*---------------------------------------------------------------------------
;                           Messaging
;---------------------------------------------------------------------------*/

// Message control codes

enum{
   rsSendMessage        = 800,   // Send a message
   rsInstallMsgHandler  = 801,   // Install a message handler
   rsRemoveMsgHandler   = 802,   // Remove message handler
   rsRegisterMessage    = 803    // Register message type
};

// Message Results (in msgResult field of MsgPBlk)

enum {
   msgNoError  = 0,     // No error, completed
   msgOverrun  = -1,    // More data was available
   msgUnderrun = -2,    // Less data was available
   msgTimeout  = -3     // Timeout error
};

typedef struct MsgPBlk {
   struct MsgPBlk*   msgQLink;   // Pointer to next queue element
   short    msgQType;            // Queue Flags
   short    msgCmd;              // The message type or command
   long     msgParam1;           // Message parameter 1
   long     msgParam2;           // Message parameter 2
   void*    msgBuffer;           // Ptr to the message data buffer
   long     msgReqCount;         // Requested data length
   long     msgActCount;         // Actual data length
   ProcPtr  msgCompletion;`      // Ptr to completion routine or NULL
   short    msgResult;           // The result of message operation
   short    msgFlags;            // Message flags (swap and shared)
   long     msgUserData;         // For use by caller (a5, etc…)
} MsgPBlk, *MsgPBlkPtr;
```

```
typedef struct MsgRecElem {
   struct MsgRecElem*   recQLink;   // Next queue element
   short    recQType;               // queue flags
   short    recFlags;               // Not used...Yet...Set to zero
   ProcPtr  recProc;                // Ptr to the receive proceedure
   short    recCmdBase;             // first command received by this proc
   short    recCmdCount;            // # of commands allocated for this proc
   long     recUserData;            // For caller's use (could be A5...)
} MsgRecElem, *MsgRecElemPtr;

/
*-----------------------------------------------------------------------------
;                           Sound
;-----------------------------------------------------------------------------*/

enum{
   rsEnableSound  = 1000,        // Enable sound emulation
   rsDisableSound = 1001         // Disable sound emulation
};

/
*-----------------------------------------------------------------------------
;                           Notification Proc
;-----------------------------------------------------------------------------*/

// Control Codes

enum{
   rsSetNotificationProc = 900   // Sets the address of the Notification
                                    proceedure
};

/
*-----------------------------------------------------------------------------
;                           Event Notification
;-----------------------------------------------------------------------------*/

// Control Codes

enum{
   rsEventInstall = 1100,     // Install and event handler
   rsEventRemove  = 1101,     // Remove an event handler
   rsEventNotify  = 1102      // Notify event chain of an event
};
```

```
// Status Codes

enum{
   rsEventSample  = 1100         // Sample data on an event
};

/
*-------------------------------------------------------------------------
;                          NotifyUPP definition
;-------------------------------------------------------------------------*/
/*
   pcCardNotificationProcs cannot be written in or called from a high-level
   language without the help of mixed mode or assembly glue because they
   use the following parameter-passing convention:

   typedef pascal void (*PCCardNotifyProcPtr)(short event, long a1Param);

      In:
         event    D0.W
         a1Param  A1.L

      Out:
         none
*/

enum  {
   uppPCCardNotifyProcInfo= kRegisterBased
         |REGISTER_ROUTINE_PARAMETER(1,kRegisterD0,kTwoByteCode)
         |REGISTER_ROUTINE_PARAMETER(2,kRegisterA1,kFourByteCode)
};

#if USESROUTINEDESCRIPTORS
typedef pascal void (*PCCardNotifyProcPtr)(short event, long a1Param);

typedef UniversalProcPtr PCCardNotifyUPP;

#define CallPCCardNotifyProc(userRoutine, event, a1Param)  \
   CallUniversalProc((UniversalProcPtr)(userRoutine),
   uppPCCardNotifyProcInfo, (event, a1Param))

#define NewPCCardNotifyProc(userRoutine)  \
   (PCCardNotifyUPP) NewRoutineDescriptor((ProcPtr)(userRoutine),  \
   uppPCCardNotifyProcInfo, GetCurrentISA())

#else
typedef ProcPtr PCCardNotifyUPP;
```

```
#define NewPCCardNotifyProc(userRoutine)  \
   (PCCardNotifyUPP)((userRoutine))

#endif

/
*-------------------------------------------------------------------------
;                    function prototypes
;-------------------------------------------------------------------------*/
OSErr OpenPCCardDriver(short *refNum);
OSErr CallrsBootPCSync(short refNum);
OSErr CallrsResetPCSync(short refNum);
OSErr CallrsPCStatusSync(short refNum, long *status);
OSErr CallrsEnableVideoSync(short refNum);
OSErr CallrsDisableVideoSync(short refNum);
OSErr CallrsMountDisksSync(short refNum);
OSErr CallrsDontMountDisksSync(short refNum);
OSErr CallrsActivateKBSync(short refNum);
OSErr CallrsDeactivateKBSync(short refNum);
OSErr CallrsBeginMouseTrackingSync(short refNum);
OSErr CallrsEndMouseTrackingSync(short refNum);
OSErr CallrsEndPrintJobSync(short refNum);
OSErr CallrsSetNotifcationProcSync(short refNum, PCCardNotifyUPP
                                    *pcCardNotifyUPPPtr, long *a1ParamPtr);
OSErr CallrsSetMemoryConfigSync(short refNum, long logBaseAddr,
                                long physBaseAddr, long memlen);

#endif// __PCCARDCALLS__
```

Index

Numerals

A

B

C

D

F

G

H

I

J

K

L

M

O

P

R

S

T

V

X

Y

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

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

WRITER
Allen Watson III

DEVELOPMENTAL EDITOR
Jeanne Woodward

ILLUSTRATORS
Deb Dennis and Shawn Morningstar

Special thanks to Richard Kubota, Tom Llewellyn, and Jim Stockdale.