Chapter 3 Firmware

This chapter describes the firmware portion of the total software environment for the Macintosh Portable computer. Software will include both ROM-stored code (firmware) and disk-stored code (system software). The Macintosh Portable firmware is an outgrowth of that for the Macintosh SE. This chapter describes the changes from the Macintosh SE firmware (the ROM image). The ROM to which we refer is that on the 68HC000 I/O bus. Chapter 4 describes the changes in the contents of the system tools disk. Also, see "Software Developer Guidelines," Chapter 2.

▪

This page is a blank

3.1 Overview

The Macintosh SE software is extensively documented in *Inside Macintosh*, Volume V. The contents of Volume V that apply to the Macintosh SE describe its software in terms of changes from the Macintosh Plus, documented in *Inside Macintosh*, Volume IV. Volume IV, in turn, describes changes from the classic Macintosh as documented in *Inside Macintosh*, Volumes I, II, and III.

Terminology

The Macintosh Portable software comes in two components:

- Firmware—contents of the three ROMs, one for each of the three processors (68000 CPU, power manager processor, and the keyboard processor).
- System software—contents of an 800 KB, 3.5" disk, Version 6.0.3. System software includes the system file, ROM patches, cdevs (control devices), and more.

Reference to *Macintosh Portable ROM* will mean the 68000 processor ROM; discussion of ROM firmware for either of the peripheral processors will be explicitly labeled as such.

The Macintosh Portable ROM is fundamentally the same ROM as for the Macintosh SE; however, there are some important changes and additions. See section 3.3, “Changes to ROM,” for more detail.

See Chapter 4, “Software,” for information on updates to the system tools disk.

3.2 Address map

Figure 3-1 shows the address map of the Macintosh Portable. For comparison, Figure 3-2 shows the address map of the Macintosh SE.

Figure 3-1 The Macintosh Portable Address Map

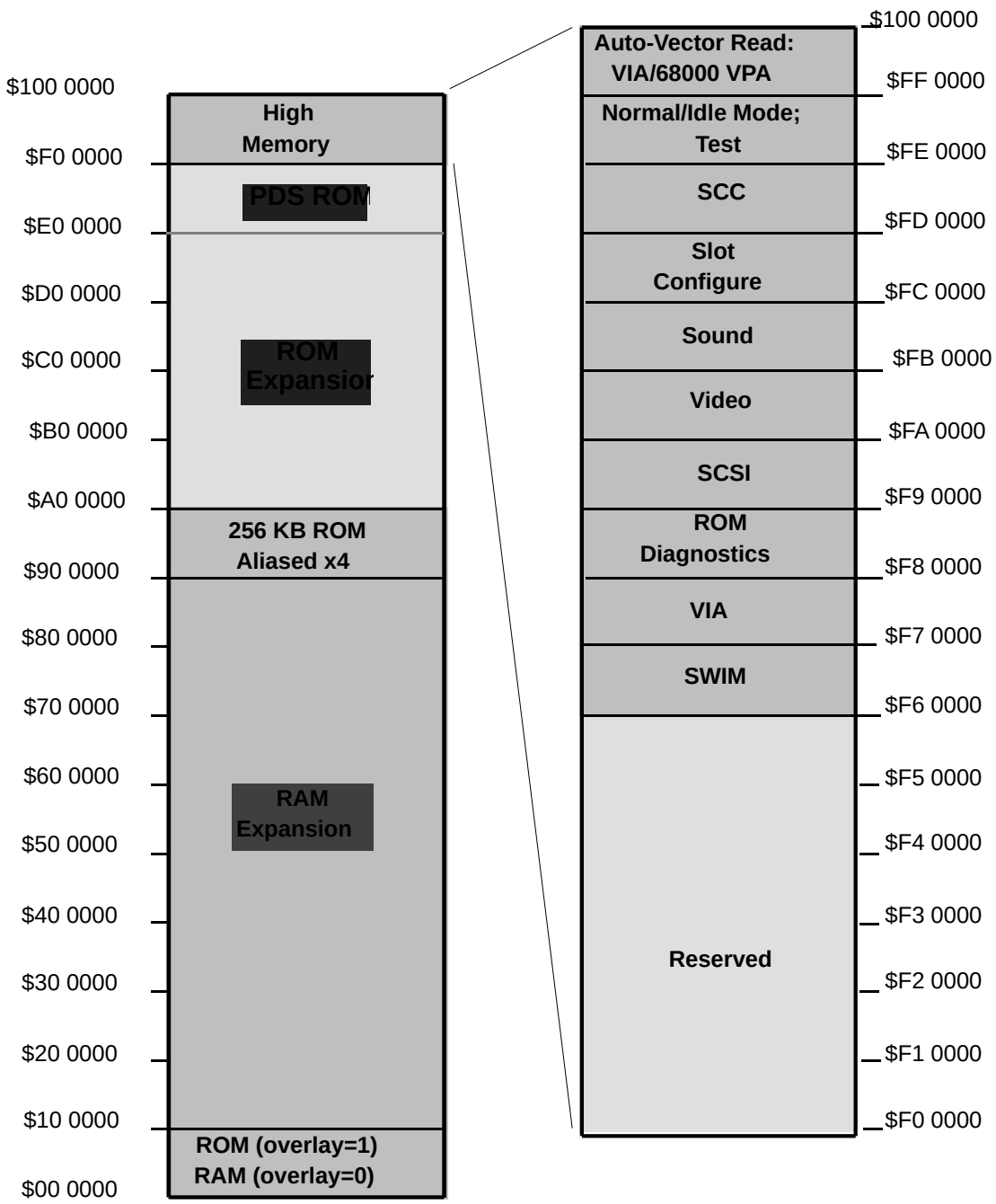
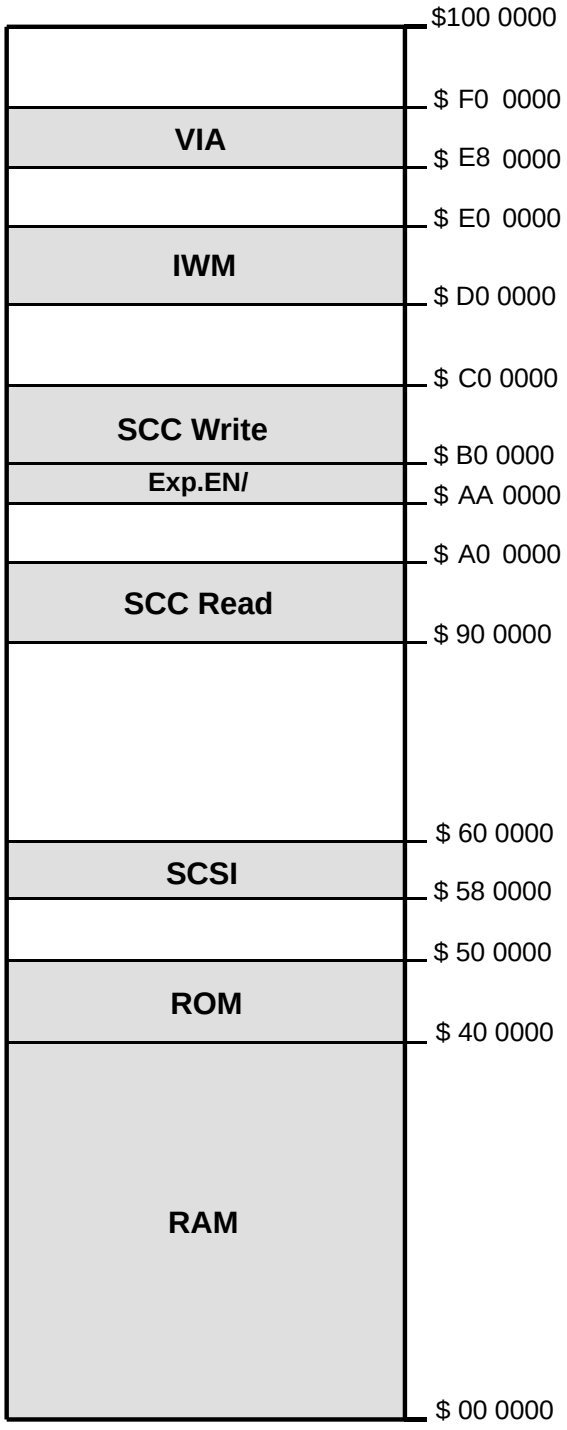


Figure 3-2 The Macintosh SE Address Map



3.3 Changes to ROM

The Macintosh Portable ROM software is based on the 256 KB Macintosh SE ROM; all patches to the Macintosh SE ROM that are contained in disk-stored system software have been incorporated in the ROM image of the Macintosh Portable and no longer need reside in RAM. The Macintosh Portable ROM image is 256 KB in size. It is stored in either masked ROM on the main logic board or (during development) in two 1 megabit ROMs on the ROM expansion board.

Power manager processor

The hardware interface to the power manager processor is through a port on the VIA (see Chapter 6, “The Power Manager,” for details); the software interface is through the A trap mechanism (see *Inside Macintosh* Volume I, page I-88). Drivers have been modified to call the power manager to turn on and off their respective peripheral chips.

■ **Table 3-1** Power Manager Processor

Replaces	Provides
■ Real time clock	-
■	Apple Desktop Bus transceiver -
-	■ Power and clock control for peripheral subsystems
-	■ A computer wake-up timer facility
-	■ LCD screen contrast control
-	■ Control of internal modem connection to serial ports
-	■ Parameter RAM
-	■ Monitoring and control of the battery and charger system

Apple Desktop Bus

Some changes have been made to the code for the ADB state machine to use the power manager as a smart transceiver. Externally, there is no visible change, the same functionality has been provided.

Real-time clock (RTC) and Parameter RAM

As the RTC and parameter RAM functions have been taken over by the power manager, a new interface to the clock is provided. A one-second timer, based on the 60 Hz oscillator used by the power manager processor, is used to generate the real-time clock.

There are two real-time clock functions: one to set and one to read the clock. The clock data is stored as a count of the number of seconds since midnight, 1 January 1904.

Parameter RAM is the storage location for various settings (such as those specified by the user on the Control Panel desk accessory) that need to be preserved during sleep or power off. Only 128 bytes of extended parameter RAM are supported by the power manager. See Chapter 6, "The Power Manager," for further details.

Applications should not use parameter RAM assuming it to be the same as earlier Macintosh models, because it is not. The Macintosh Plus, Macintosh SE, and Macintosh II have 256 bytes as compared to 128 bytes for the Macintosh Portable.

Serial Driver

This driver software is modified for switching power to the SCC chip and the serial driver chips before accessing them.

FDHD, the high-density floppy disk drive

The FDHD™ disk drive is a new 3.5-inch floppy disk drive for the Macintosh II, Macintosh IIx, Macintosh SE/30, and Macintosh Portable computers. FDHD provides the ability to read and write data in both group code recording (GCR) and modified frequency modulation (MFM) formats. This new MFM capability allows Macintosh users to read and write MS-DOS files and, potentially, other files that use the MFM disk formats.

FDHD provides 1400 KB (1.4 MB) of MFM storage. It also continues to support the standard storage capacities in GCR mode: single-sided 400 KB storage capacity, and double-sided 800 KB storage capacity disks.

The FDHD disk drive provides the high-density read/write hardware and read/write circuitry. The new ROM set incorporates the new disk driver. The SWIM chip provides Macintosh Portable with MFM disk read/write capability while maintaining HFS compatibility. The following information is needed to develop application programs that use the new FDHD 1.4 MB floppy disk drive. Also described are the data encoding techniques used in the drive, the disk driver firmware, and how to use both.

Data storage

The theory behind disk drive technology is relatively easy to understand. By making calls to the disk driver, you are causing the disk controller chip (the SWIM chip) to send control signals and data to the disk drive.

Data is recorded on a disk very much as a voice or music is recorded on magnetic tape: Current flow is varied through a read/write head that is placed close to the medium (the disk or tape). Changing the current flow in the read/write head results in a magnetic transition on the disk. It is these magnetic transitions that interest us.

Several techniques are used by computer designers to write data to disk media. All of these techniques share a common technology: encoding data as magnetic transitions on a disk. To represent individual bits, a pattern of ones and zeros is expressed as a series of magnetic transitions. The data format is the combination of magnetic transitions that represents a particular series of bits. FDHD uses two data formats: group code recording (GCR) and modified frequency modulation (MFM). Table 3-2 shows the four possible combinations of file systems and disk formats.

Table 3-2 Possible disk format

Disk formats	File systems
400 KB GCR †	MFS
800 KB GCR †	HFS
720 KB MFM†	MS-DOS
1440 KB MFM *	HFS or MS-DOS

† requires a standard 3.5-inch disk
* requires a high density 3.5-inch disk

GCR format

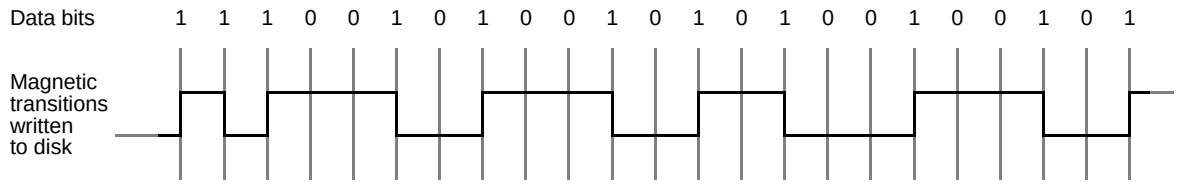
The GCR data format has been used in all Apple floppy disk drives to date, including the Macintosh single-sided and double-sided disk drives.

Data bits are represented by magnetic transitions in the following manner:

- A transition always occurs when a 1 is encountered.
- No transitions occur when a 0 is encountered.

Figure 3-3 shows the relationship between the data bits and the magnetic transitions written to the disk when using the GCR data format.

- **Figure 3-3** GCR data format



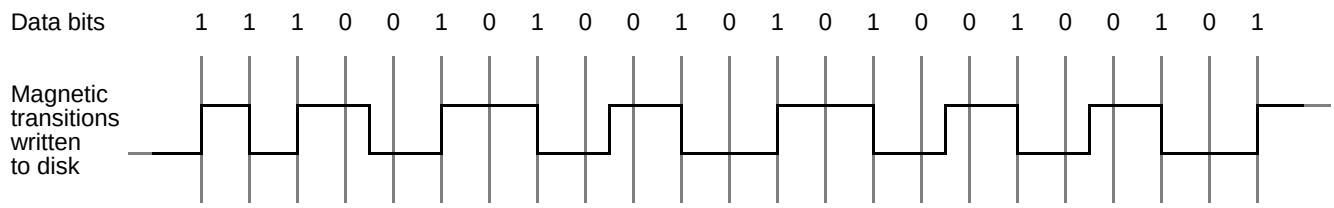
MFM format

Another standard data format is the MFM format, which is used by MS-DOS computers to store data. Data bits are represented by magnetic transitions in the following manner:

- A transition always occurs when a 1 is encountered.
- A transition always occurs when two adjacent 0's are encountered.

Figure 3-4 shows the relationship of data bits and magnetic transitions when using the MFM data format.

Figure 3-4 MFM data format



The disk media

The FDHD disk drive uses a special 3.5-inch 1440 KB disk. The disk has a hole cut in the upper-left corner that identifies it as high-density media. Do not format this disk as a 400 KB or 800 KB GCR disk in any Macintosh disk drive. Doing so will place your data at risk. Use only standard Macintosh single-sided and double-sided disks in single-sided (400 KB) and double-sided (800 KB) drives. When a GCR formatted 400K or 800K HD disk is inserted in a FDHD drive, an eject or initialize dialog box will be displayed.

Figure 3–5 shows media compatibility for different disk drives.

Figure 3-5 Disk media compatibility

Media \ Disk drive	Single-sided	Double-sided	FDHD
Single-sided 400K	Disk is recognized as a standard 3.5-inch disk. Only single-sided (400K) format is possible.	Disk is recognized as a standard 3.5-inch disk. Format this disk only as a single-sided disk. Formatting this disk as a double-sided disk and saving data on it places that data at risk.	Disk is recognized as a standard 3.5-inch disk.
Double-sided 800K	Disk is recognized as a standard 3.5-inch disk. Only single-sided (400K) format is possible.	Disk is recognized as a standard 3.5-inch disk. Either single- or double-sided format is possible.	Disk is recognized as a standard 3.5-inch disk. Use only double-sided disks when formatting in 800K mode.
High-density 1.44MB	Incompatible. Do not use high-density media in single-sided or double-sided disk drives. Use high-density disks only in the FDHD disk drive. Data written to a high-density disk in 400K or 800K disk drives places that data at risk.		Disk is recognized as a high-density 3.5-inch disk. Use high-density disks only in the FDHD disk drive.

The File System

Apple is working on full support in the file system for the FDHD disk drive in both MFM and GCR modes. Today, support for the FDHD is provided in the current version of the Apple File Exchange (version 1.1). Apple File Exchange is a Macintosh application program that allows users to transfer files from MS-DOS disks to the Macintosh, with the option of translating proprietary file formats. For more information on Apple File Exchange, refer to Chapter 7, "Using Apple File Exchange," in the *Macintosh Utilities User's Guide*.

3-13 Developer Notes

High Density Floppy Disk Driver

The Macintosh Portable ROM includes a new disk driver. This driver supports the MFM data format and several other new features. This is a new driver to interface with the SWIM chip and support MFM, as well as GCR, data encoding; the driver also provides power control. The calls to this driver are described in the following several pages.

Control calls perform all of the disk operations except reading data and writing data. The control opcode is passed to the driver in the *csCode* field (byte 26) of the I/O parameter block. Refer to the Device Manager chapter in *Inside Macintosh*, Volume II. Control calls that return information pass it back in the I/O parameter block, beginning with the *csParam* field (byte 28).

Kill I/O (csCode=1)

Kill I/O is called to abort any current I/O request in progress. The driver does not support this control call and always returns a result code of –1.

Verify Disk (csCode=5)

This control call reads every sector from the selected disk to verify that all sectors have been written correctly. If any sector is found to be bad, the call aborts immediately and returns an error code.

Format Disk (csCode=6)

If the selected disk is a floppy disk, the driver writes address headers and data fields for every sector on the disk and (for GCR disks only) does a limited verification of the format by checking that the address field of the first sector on each track can be read. If the selected disk is a Hard Disk 20, the driver doesn't format the media, but instead initializes the data of each sector to be all 0's. If any error occurs (including write-protected media), the formatting is aborted and an error code is returned.

The *csParam* field is used to specify the type of format to be done on *floppy* disks only. In the SWIM and later versions of the driver, this value is an index into a list of possible formats for the given drive/media combination. (See the Return Format List status call under "Status Calls," later in this chapter, for values.)

◆ *Note*: In previous versions of the driver, setting *csParam* to $0001 creates a single-sided disk. Setting *csParam* to a value other than $0001 creates a double-sided disk.

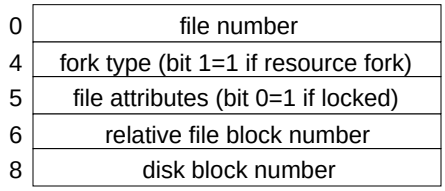
Eject Disk (csCode=7)

This call ejects the disk in the selected drive if that drive supports removable media. Since hard disks are not removable, if a hard disk is ejected, the driver posts a diskInserted event and remounts the drive.

Set Tag Buffer (csCode=8)

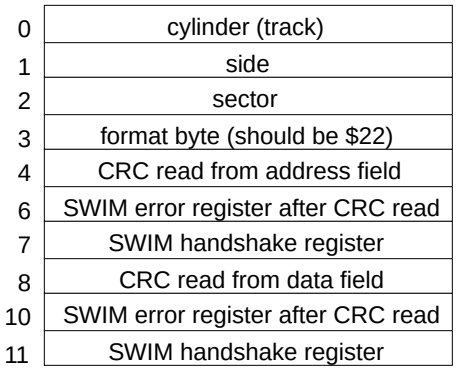
If *csParam* is zero, no separate tag buffer is used. If *csParam* is nonzero, it is assumed to contain a pointer to a buffer into which tag bytes from each block are read or into which they are written on each Prime call. Every time a block is read from the disk, the 12 tag bytes are copied into the file tag buffer at TagData+2 ($2FC), and then are copied into the user's tag buffer. When a block is written, tag bytes are copied into the file tag buffer from the user's tag buffer, and then written to the disk with the rest of the block. The position of a particular block's tag bytes in the user tag buffer is determined by that block's position relative to the first block read/written on the current Prime call. The file tags for GCR disks include information that a scavenging utility can use to rebuild a disk if the directory structure gets trashed. Figure 3–6 shows the tag format.

Figure 3-6 GCR file tag format



MFM disks don't support file tags, so instead of scrapping the whole idea, information about the sector itself is returned. Most of it is read from the disk, but the error register bytes show any error conditions that exist after the address or data field is read. Figure 3–7 shows the format of a sector information block.

Figure 3-7 MFM sector information block



Track Cache Control (csCode=9)

When the track cache is enabled, all of the sectors on the last track accessed during a read request, and those requested by the user, are read into a RAM buffer. On future read requests, if the track is the same as the last track on the last read request, the sector data is read from the cache rather than from disk. Write requests to the driver are passed directly to the disk, and any of the sectors written that are in the cache are marked invalid. To control the cache, 2 bytes are passed at *csParam* to control the cache, located at *csParam* and *csParam*+1. These codes are shown in Table 3–3 and Table 3–4.

Table 3-3 Cache enable codes

csParam	Result
0	Disable the cache.
≠0	Enable the cache.

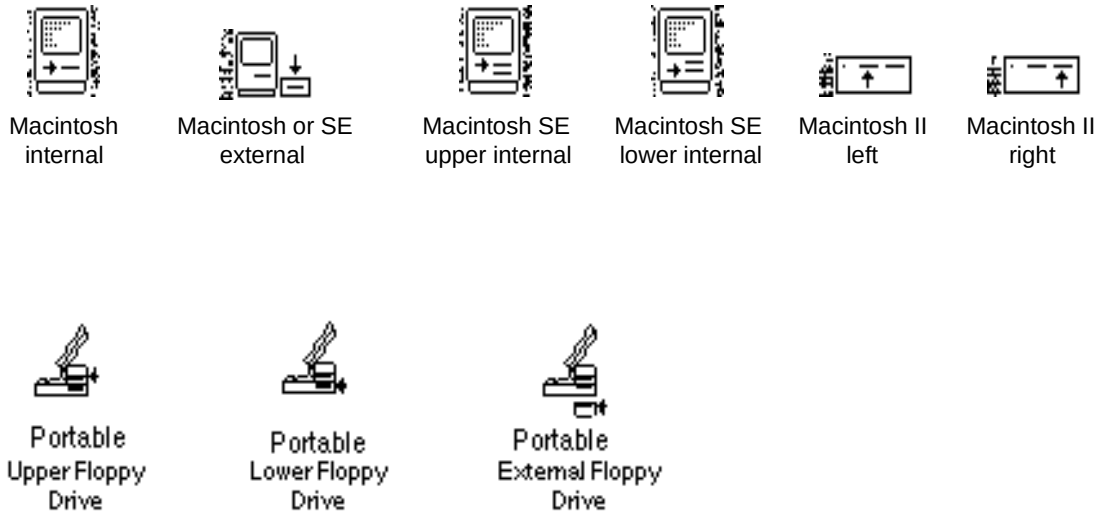
Table 3-4 Cache control codes

csParam+1	Result
<0	Remove the cache.
0	Don't remove or install the cache.
>0	Install the cache.

.Return Physical Drive Icon (csCode=21)

This call returns a pointer to an icon that shows the selected drive's physical location. The supported icons are shown in Figure 3–8. Note that only the icons for a particular machine are included in that machine's disk driver.

Figure 3-8 Drive icons



Return Media Icon (csCode=22)

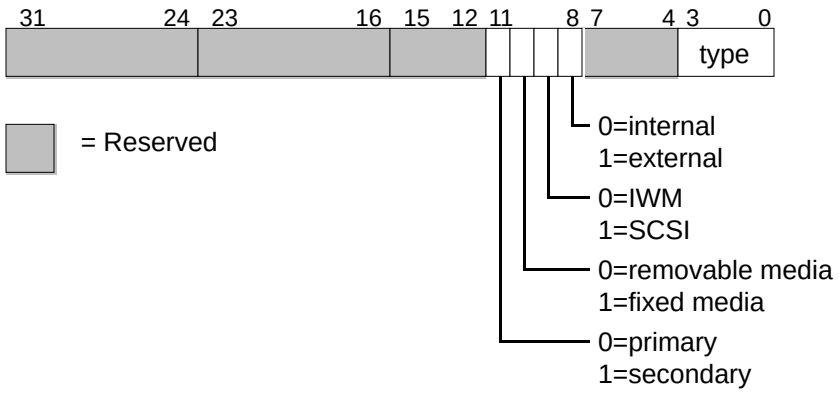
This call returns a pointer to an icon that shows the selected drive's media type. The floppy disk icon is stored in the driver. The Hard Disk 20 icon is stored in the disk drive's ROM. Figure 3–9 shows the icons.

Figure 3-9 Media icons

Return Drive Info (csCode=23)

This control call returns a 32-bit value in *csParam* that describes the location and attributes of the selected drive. Figure 3-10 shows the format of the Return Drive Information

Figure 3-10 Return drive information format



The attributes field occupies bits 8 to 11 and describes the location (internal/external, primary/secondary), drive interface (IWM/SCSI), and media type (fixed/removable).

Most of the bits are currently not used and are reserved for future expansion. The drive type field occupies bits 0 to 3 and describes the kind of drive that is connected. Currently six different types are supported; they are listed in Table 3–5.

Table 3-5 Drive types

Type	Description
0	no such drive
1	unspecified drive
2	400 KB
3	800 KB
4	FDHD (400 KB/800 KB GCR, 720 KB/1440 KB MFM)
5	reserved
6	reserved
7	Hard Disk 20
8-15	reserved

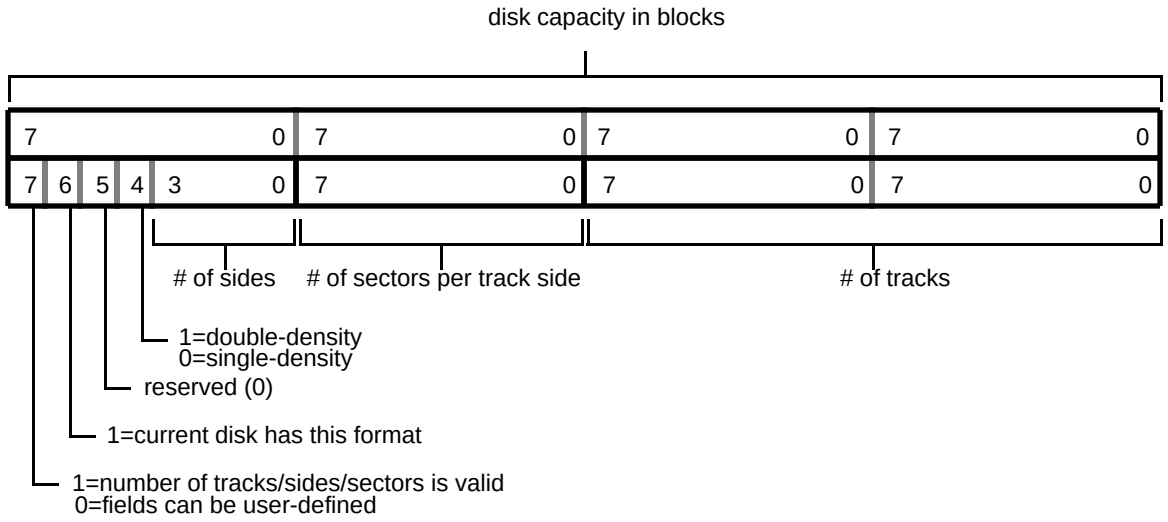
Status calls

The disk driver currently supports three status calls, which are described below. As with the control calls, the status opcode is passed to the driver in the *csCode* field of the I/O parameter block (byte 26). The returned status information is passed back starting at the *csParam* field of the I/O parameter block (byte 28).

Return Format List (csCode=6)

This call is supported in the SWIM-compatible disk driver and will be supported in future versions of the disk driver, whether or not MFM disks are supported. This call returns a list of all disk formats possible with the current combination of disk controller, drive, and media. Upon entry, *csParam* contains a value specifying the maximum number of formats to return, and *csParam+2* contains a pointer to a table that will contain the list. On exit, *csParam* will contain the number of formats returned (no more than specified), and the table will contain the list of formats. If no disk is inserted in the drive, the call will return a *noDriveErr* code. The format information is given in an 8-byte record as shown in Figure 3-11.

Figure 3-11 Return format record



If a track, side, or sector field is zero when the *valid* bit is set to 1, the field is considered to be a "don't care" as far as describing the format of the disk. The formats supported by the driver are listed in Table 3–6.

Table 3-6 Combinations of drives and media

Format	Capacity (in blocks)	TSS valid[1]	SD/DD	Sides	Sectors[2]	Tracks
400 KB GCR	800	yes	SD	1	10	80
800 KB GCR	1600	yes	SD	2	10	80
720 KB MFM[3]	1440	yes	SD	2	9	80
1440 KB MFM[3,4]	2880	yes	DD	2	18	80
Hard Disk 20	38965	no	SD	0	0	0

Notes:

[1] track, sector, and side information

[2] average number of sectors

[3] requires SWIM and FDHD

[4] requires HD media

Drive Status (csCode=8)

Drive Status returns information about a particular drive, starting at *csParam*; the values returned are listed in Table 3–7.

Table 3-7 Drive status return values

Offset	Description	Parameters
0		current track value of current track
2		write protect bit 7 = 1, write-protected
		bit 7 = 0, write-enabled
3		disk in place? <0 = disk is being ejected
		0 = no disk is currently in the drive
		1 = disk was just inserted but no read/write requests have been made for this disk
		2 = OS has tried to mount the disk (that is, read request to driver)
		3 = same as “2” except that this is a high-density disk formatted as 400KB or 800KB GCR
		8 = same as “2” except except that this is a Hard Disk 20 (8 means disk is nonejectable)
4		drive installed? –1 = no drive installed
		0 = don’t know
		1 = drive installed
5		number of sides 0 = single-sided
		–1 = double-sided

Table 3-7 Drive status return values (Continued)

Offset	Description	Parameters
6		drive queue element 6 = qLink—pointer to next queue element
		10 = qType—type of queue (drvQType)
		12 = dqDrive—drive number
		14 = dqRefNum—disk driver's reference number
		16 = dqFSID—file system ID
18		double-sided format? 0 = current disk has single-sided format
		–1 = current disk has double-sided format
19		new interface 0 = old drive interface (400KB)
		–1 = new interface (800KB and later)
20		soft error count (2 bytes) number of soft errors encountered

MFM Status (csCode=10)

This call is supported in the SWIM-compatible disk driver and will be supported in future versions of the disk driver. By making this call and then checking the returned error code, it is possible to determine whether or not the version you are using can read and write MFM disks. Also, the information returned is helpful in determining the installed hardware configuration. The information is returned starting at *csParam*. Table 3-8 lists the values returned.

Table 3-8 MFM status return values

Offset	Description	Parameters
0		drive type –1 = FDHD (MFM/GCR) 0 = 400KB or 800KB GCR
1		disk format –1 = MFM 0 = GCR (valid only when installed)
2		MFM format –1 = 1440KB disk 0 = 720KB disk
3		disk controller –1 = SWIM 0 = IWM

A sample program

As an example of how to use the new disk driver, a sample application program is included here.

```
{
_____________________________________________________________________

    FDHD™ Driver Demo Application

    Copyright © 1988 by Apple Computer, Inc.


This is a short application to show off both the old and new control and status calls for the FDHD disk driver.
The application does this by presenting a list of information about each 3.5-inch floppy or original Hard Disk
20 drive connected to the Macintosh, and the format of any disks that are in those drives.

_____________________________________________________________________
}

PROGRAM HDFDDriverDemo;

 USES          MemTypes,QuickDraw,OsIntf,ToolIntf,PackIntf,PasLibIntf;

 CONST  DemoMenu    = 256;                          {menu's resource ID and item numbers}
              AboutItem     = 1;
              QuitItem = 3;
```

```
DemoDLOG  = 256;
{"demo" dialog's resource ID and item numbers we use:}
    Eject1Btn= 1;
{disk eject buttons}
Eject2Btn= 2;
Eject3Btn= 3;
ChipTypTxt= 11;                {disk controller type}
BaseDrvLoc= 12;                    {base drive location}
BaseDrvIcn= 15;                    {base drive icon item}
BaseDskIcn= 18;                    {base disk icon item}
BaseDrvNum= 21;                {base drive number}
BaseDrvTyp= 24;                {base drive type}
BaseDskFmt= 27;                {base disk format}

                                   StrRsrcID          = 256;
                                                              {descriptive
                               string list resource ID and string numbers:}
IWMStr  = 1;                   {chip types}
SWIMStr= 2;
Int1DrvStr = 3;                    {primary internal drive}
Int2DrvStr = 4;                    {secondary internal drive}
Ext1DrvStr = 5;                    {primary external drive}
Ext2DrvStr = 6;                    {secondary external drive}
Drive400K     = 7;             {drive types}
Drive800K     = 8;
FDHD = 9;
Hard Disk 20 = 12;

                                   DefDiskIcon        = 256;
                                                          {default drive/disk
                               icon in case we have errors}

HDFDRefNum      = -5;              {FDHD driver's reference number}
FmtListCode  = 6;              {csCode for "format list" status call}
DrvStsCode    = 8;                 {csCode for "drive status" status call}
MFMStsCode = 10;                   {csCode for "MFM status" status call}
DrvIconCode  = 21;                 {csCode for "drive icon" control call}
DskIconCode  = 22;                 {csCode for "disk icon" control call}
DrvInfoCode  = 23;                 {csCode for "drive info" control call}
```

```
TYPE    DrvFmtRec = PACKED RECORD                    {disk format description:}
        capacity: LONGINT;                           { number of blocks on the disk}
        flagsNHeads: SignedByte;                     { [flags][number of heads]}
        sectors: SignedByte;                         { number of sectors per track side}
        cylinders: INTEGER;                          { number of tracks [cylinders]}
        END;

        FmtInfoRec    = RECORD                       {format list:}
                                                     numFormats    : INTEGER;
                                                     { number of formats we want/are returned}
            fmtBlock : Ptr;                          { where to put them}
            END;

        FmtInfoPtr    = ^FmtInfoRec;

        MFMSts        = PACKED RECORD                {info about chips, drives, disks:}
                                                     isHDFD: SignedByte;
                                                          {-1=HDFD, 0=400K/800K drive}
            diskFormat    : SignedByte;              { -1=MFM, 0=GCR}
                                                     twoMegFmt    : SignedByte;
                                                     {-1=1440K, 0=720K (if diskFormat=-1)}
            isSWIM        : SignedByte;              {-1=SWIM, 0=IWM}
            END;

        MFMStsPtr     = ^MFMSts;

        DrvStsPtr= ^DrvSts;


VARtheEvent  : EventRecord;
        demoDialog    : DialogPtr;                   {"the" window}
        whichWindow : WindowPtr;                     {the window FindWindow is talking about}
        pb: ParamBlockRec;                           {parameter block for control/status calls}
        itemHit       : INTEGER;                     {item number returned by DialogSelect}
        driveIcon,                                        {drive icon for each column}
                                                          diskIcon  : ARRAY[0..2,0..31] OF LONGINT;
                                                                {disk icon for each column}
        driveNum      : ARRAY[0..2] OF INTEGER;           {drive number for each column}
```

```
{
_____________________________________________________________

                    D I A L O G   R O U T I N E S
_____________________________________________________________
}

 { hides the specified dialog control }

PROCEDURE HideDControl(theItem:INTEGER);
      VARtheType        : INTEGER;
            theControl     : ControlHandle;
            theRect        : Rect;
      BEGIN
        GetDItem(demoDialog,theItem,theType,Handle(theControl),theRect);
        HideControl(theControl);
      END;


 { sets the highlighting of the specified dialog control }

PROCEDURE HiliteDControl(theItem,theHilite:INTEGER);
      VARtheType        : INTEGER;
            theControl     : ControlHandle;
            theRect        : Rect;
      BEGIN
            GetDItem(demoDialog,theItem,theType,Handle(theControl),theRect);
            HiliteControl(theControl,theHilite);
      END;


 { changes the text of a staticText item to theText }

PROCEDURE SetDText(theItem:INTEGER; theText:Str255);
      VARtheType        : INTEGER;
            theHandle      : Handle;
            theRect        : Rect;
      BEGIN
            GetDItem(demoDialog,theItem,theType,theHandle,theRect);
            SetIText(theHandle,theText);
      END;
```

```
{ draws a drive icon for the given drive }

PROCEDURE DrawDriveIcon(theDialog:DialogPtr; theItem:INTEGER);
    VARtheBits      : BitMap;
        theType     : INTEGER;
        theHandle   : Handle;
    BEGIN
        IF driveNum[theItem-BaseDrvIcn]<>0 THEN BEGIN
            theBits.baseAddr:=@driveIcon[theItem-BaseDrvIcn];
            theBits.rowBytes:=4;
            GetDItem(theDialog,theItem,theType,theHandle,theBits.bounds);
                                CopyBits(theBits, theDialog^.portBits, theBits.bounds, theBits.bounds,srcCopy,NIL);
        END;
    END;


{ draws a disk icon for the given drive }

PROCEDURE DrawDiskIcon(theDialog:DialogPtr; theItem:INTEGER);
    VARtheBits      : BitMap;
        theType     : INTEGER;
        theHandle   : Handle;
    BEGIN
        IF driveNum[theItem-BaseDskIcn]<>0 THEN BEGIN
            theBits.baseAddr:=@diskIcon[theItem-BaseDskIcn];
            theBits.rowBytes:=4;
            GetDItem(theDialog,theItem,theType,theHandle,theBits.bounds);
                                CopyBits(theBits,theDialog^.portBits,theBits.bounds, theBits.bounds,srcCopy,NIL);
        END;
    END;
```

```
{
_____________________________________________________________

                MISCELLANEOUS DISK STUFF
_____________________________________________________________
}

 { Pascal doesn't support a SWAP operation, and since the Hard Disk 20's }
 { drive size fields are in the opposite order from what we need, we'll }
 { have to swap them ourselves. The hex code following the function  }
 { translates to:                              }

 {    $205F    MOVEA.L     (SP)+,A0;                    Get the pointer to "driveSize" }
 {    $2010    MOVE.L (A0),D0;                  D0.L=[driveSize][driveS1] (backwards) }
 {    $4840    SWAP     D0;                         D0.L=[driveS1][driveSize] (how we want it) }
 {    $2E80    MOVE.L D0,(SP);                  Put the result on the stack }


FUNCTION GetHD20Size(rawSize:Ptr):LONGINT; INLINE $205F, $2010, $4840, $2E80;

 { check what format this disk has and enable the Eject button }

PROCEDURE NewDisk(theDrive:INTEGER);
      VARtheFormat       : LONGINT;
           I                  : INTEGER;
           formatList       : FmtInfoPtr;
           formatInfo       : ARRAY[0..3] OF DrvFmtRec;
           driveStatus      : DrvStsPtr;
           theString : Str255;
```

```
BEGIN
    IF theDrive IN [1..3] THEN                          {it might be one of ours}
        IF driveNum[theDrive-1]<>0 THEN BEGIN               {it is, so continue}
            theFormat:=0;                                   {no format yet}

{ First, assume we're working with a recent enough }
{ version of the driver to support the Format List call.  If }
{ so, we can determine how big this disk is…                  }

                        formatList:=FmtInfoPtr(@pb.csParam);

                    {type coercion (hazard of Pascal)…}
                        formatList^.numFormats:=4;

                    {we want up to 4 entries}
                        formatList^.fmtBlock:=@formatInfo;

                    {and here's where to put them}

                        pb.ioCompletion:=NIL;

                    {no completion routine}
                        pb.ioRefNum:=HDFDRefNum;

                    {FDHD driver's reference number}
                        pb.ioVRefNum:=theDrive;

                    {drive number}

                        pb.csCode:=FmtListCode;

                    {go get the list}
                        IF PBStatus(@pb,FALSE)=NoErr THEN
                BEGIN
                    {got something back}
                                I:=formatList^.numFormats;
                                WHILE (I>0) AND
                (theFormat=0) DO BEGIN
                    {scan the list for this disk's format}
                                        I:=I-1;
```

```
                                                                    IF
                                                        BTST(formatInfo[I].flagsNHeads,6) THEN
                                                          {bit 6=1 means this is the current format}

                                                          theFormat:=formatInfo[I].capacity DIV 2;
                                                              { save the disk's size in K-bytes}
            END;
        END;

{ If theFormat=0 then the Format List call }
{ isn't supported (because it's an old driver version)…  }

        IF theFormat=0 THEN BEGIN                          {haven't figured it out yet}
            pb.csCode:=DrvStsCode;                             {get drive status}
            IF PBStatus(@pb,FALSE)=NoErr THEN BEGIN

                                                        driveStatus:=DrvStsPtr(@pb.csParam);
                                                                {type coercion…}
                IF driveStatus^.diskInPlace=8 THEN         {Hard Disk 20}

                                                theFormat:=GetHD20Size(@driveStatus^.driveSize) DIV 2      {blocks-
                                              > K-bytes}
                ELSE
                IF driveStatus^.twoSideFmt=0 THEN
                    theFormat:=400                             { 400K}
                ELSE theFormat:=800;                       { 800K}
            END;
        END;
```

```
                                        NumToString(theFormat,theString);
                                                                {convert the
                                disk size to a string,}
                                        theString:=CONCAT(theString,'K');

                                 {append a special K to the size,}

                                 SetDText((BaseDskFmt-1)+theDrive,theString);
                                                { and display it}

                                 HiliteDControl((Eject1Btn-1)+theDrive,0);
                                                        {enable the Eject
                                button}
                                    END;
                                 END;

{
_____________________________________________________________

                                                I N I T I A L I Z A T I O N
_____________________________________________________________
}

PROCEDURE Initialize;
                                CONST   ControlErr      = -17;
                                VAR column,I,driveType  : INTEGER;
                                  theHandle,defIcon     : Handle;
                                  theString : Str255;
                                  mfmStatus     : MFMStsPtr;
                                  driveStatus   : DrvStsPtr;

                                { sets a dialog userItem's draw procedure to theProc }

PROCEDURE SetUserProc(theItem:INTEGER; theProc:ProcPtr);
                                VAR  theType   : INTEGER;
                                  theHandle : Handle;
                                  theRect   : Rect;
                                BEGIN
                                  GetDItem(demoDialog,theItem,theType,theHandle,theRect);
```

```
SetDItem(demoDialog,theItem,theType,Handle(theProc),theRe
ct);
END;
```

```
BEGIN
   InitGraf(@thePort);

   {initialize the managers}
   InitFonts;
   FlushEvents(everyEvent,0);
   InitWindows;
   InitMenus;
   TEInit;
   InitDialogs(NIL);
   InitCursor;



demoDialog:=GetNewDialog(DemoDLOG,NIL,WindowPtr(-1)
);              {load in the demo dialog window}

{ find out what kind of disk controller chip we're using }

   pb.ioCompletion:=NIL;
                                                {no
completion routine}
   pb.ioRefNum:=HDFDRefNum;
                                                {FDHD
driver's reference number}
   pb.ioVRefNum:=1;
                                                {any drive
number will do}

   I:=IWMStr;

   {assume we're working with an IWM}
   pb.csCode:=MFMStsCode;
                                                {get MFM
status}
   IF PBStatus(@pb,FALSE)=NoErr THEN BEGIN
         mfmStatus:=MFMStsPtr(@pb.csParam);
                                    {type coercion…}
         IF mfmStatus^.isSWIM<0 THEN I:=SWIMStr;
                           {we've got a SWIM chip}
   END;
```

```
    GetIndString(theString,StrRsrcID,I);
                                {get the chip type string}
    SetDText(ChipTypTxt,theString);
                                { and display it}

{ set the icon draw procedures and fill in all drive information }

    defIcon:=GetResource('ICON',DefDiskIcon);
                                {get the default disk icon}

                                                                { in
case we get errors}
    theHandle:=Handle(@pb.csParam);
                                {a little type
coercion…}
    driveStatus:=DrvStsPtr(@pb.csParam);
                                {here too…}

    FOR column:=0 TO 2 DO BEGIN
        SetUserProc(BaseDrvIcn+column,@DrawDriveIcon);
                {set the drive icon's draw proc}
        SetUserProc(BaseDskIcn+column,@DrawDiskIcon);
                    {set the disk icon's draw proc}
```

```
{ find out if the drive is installed or not, and if it belongs to the
FDHDdriver }

        driveStatus^.installed:=-1;
                                                {make sure
this one is inited}
        pb.ioVRefNum:=column+1;
                                        {drive number}
        pb.csCode:=DrvStsCode;
                                                {find out
about this drive}
        IF (PBStatus(@pb,FALSE)=NoErr)
AND(driveStatus^.installed>=0)
                THEN BEGIN              {if the drive is installed
then:}
                        driveNum[column]:=pb.ioVRefNum;
                                { save its number,}
                        NumToString(pb.ioVRefNum,theString);
                        { convert it to a string,}
                        SetDText(BaseDrvNum+column,theString);
                        { and display it}

                        IF driveStatus^.diskInPlace<2 THEN
                                {the disk isn't quite mounted,}
                                HiliteDControl(Eject1Btn+column,255)
                        { so disable its Eject button}
                        ELSE
                                NewDisk(column+1);
                                        { otherwise find out
about it}

{ Check the drive type here in case the "get drive info" }
{ call isn't supported on this particular machine…      }

                        IF driveStatus^.diskInPlace=8 THEN BEGIN
                {only Hard Disk 20 is not ejectable}
                                driveType:=HD20;
                                        {save its drive type}
                                HideDControl(Eject1Btn+column);
                                { and get rid of its Eject button--}
                        END
```

```
{  it's not ejectable, remember?}
                ELSE
                IF driveStatus^.sides=0 THEN
                    {otherwise go by number of sides}
                    driveType:=Drive400K
                ELSE driveType:=Drive800K;

{ Get the drive's icon or use our generic one if we can't get it }

                pb.csCode:=DrvIconCode;
                IF PBControl(@pb,FALSE)=NoErr THEN

  BlockMove(theHandle^,@driveIcon[column],128)
                ELSE
BlockMove(defIcon^,@driveIcon[column],128);
```

```
{ Get the disk's icon or use our generic one if we can't get it }

                pb.csCode:=DskIconCode;
                IF PBControl(@pb,FALSE)=NoErr THEN

  BlockMove(theHandle^,@diskIcon[column],128)
                ELSE

  BlockMove(defIcon^,@diskIcon[column],128);

{ Find out the drive's type and location }

                pb.csCode:=DrvInfoCode;
                IF PBControl(@pb,FALSE)=NoErr THEN
BEGIN
  driveType:=LOWRD(BAND(LONGINT(theHandle^),
$0000000F))+
(Drive400K-2);

                                        IF
                    BTST(LONGINT(theHandle^),8)
                    THEN                {internal
                    or external drive}
                        I:=Ext1DrvStr
                ELSE
                        I:=Int1DrvStr;
                IF BTST(LONGINT(theHandle^),11)
THEN
                        I:=I+1;{secondary drive}
                END
                ELSE
                IF column=0 THEN
                    I:=Int1DrvStr
                                {older drivers may not
support this}
                ELSE
                    I:=(Ext1DrvStr-1)+column;
                        { call, so just go by drive
number}

                GetIndString(theString,StrRsrcID,driveType);
```

```
                { get the type's name}
                    SetDText(BaseDrvTyp+column,theString);
                    { and display it}
                    GetIndString(theString,StrRsrcID,I);
                        { get the drive location string}
                    SetDText(BaseDrvLoc+column,theString);
                        { and display it}

                END
                ELSE BEGIN
                                                    {if no drive is
installed}
                    driveNum[column]:=0;
                                        { then set the drive# to
zero}
                    HideDControl(Eject1Btn+column);
                                {and make the control invisible}
                END;
            END;

            ShowWindow(demoDialog);
                                        {let 'em see the window
now}
    END;
```

```
{
_____________________________________________________________

                                M A I N
_____________________________________________________________
}

    BEGIN
        Initialize;                                         { initialize everything }

        REPEAT
            IF GetNextEvent(everyEvent,theEvent) THEN BEGIN         { handle an event}
                IF theEvent.what=diskEvt THEN                                       { a
disk was just inserted, so…}
                    NewDisk(LOWRD(theEvent.message));
    { find out about it}

                IF theEvent.what=mouseDown THEN
    { mouse click in the goAway box?}
                    IF FindWindow(theEvent.where,whichWindow)=inGoAway THEN
                        IF TrackGoAway(demoDialog,theEvent.where) THEN BEGIN
                            DisposDialog(demoDialog);           { get rid of our window}
                            EXIT(HDFDDriverDemo);                   { and return to the Finder}
                        END;

                IF IsDialogEvent(theEvent) THEN                                     {it's in
our dialog window}
                    IF DialogSelect(theEvent,whichWindow,itemHit) THEN

                                                                        IF itemHit IN
                                                                    [Eject1Btn..Eject3Btn]
                                                                    THEN BEGIN      {it's
                                                                    an Eject button, so…}


                                                                            IF
                                                                    Eject(NIL,itemHit-
                                                                    (Eject1Btn-1))=0 THEN ;
                                                                            { eject the
                                                                    disk,}
```

```
                                                    SetDText((BaseDskFmt-
                                                Eject1Btn)+itemHit,");
                                                    { erase the disk format
                                                entry,}
                    HiliteDControl(itemHit,255);        { and disable the eject button}
                END;
            END;
        UNTIL FALSE;
    END.
```

FDHD Driver Demo resources

Here are the resources required by the sample application program FDHD Driver Demo.

```
/*
_____________________________________________________________________________

       Driver Demo Resource Source File

       25-Apr-88

       Copyright © 1988 by Apple Computer, Inc.
_____________________________________________________________________________
*/

#include "Types.r";

/* "demo" dialog window containing drive and disk info */

resource 'DLOG' (256, preload) {
       { 40, 30,295,475}, documentProc, invisible, goAway, 0, 256, "Driver Demo"
};


/* "demo" dialog's item list */

resource 'DITL' (256, preload) {
       {
              {180,120,210,205}, button {enabled, "Eject"};  /* drive 1's eject button */
              {180,230,210,315}, button {enabled, "Eject"};  /* drive 2's eject button */
              {180,340,210,425}, button {enabled, "Eject"};  /* drive 3's eject button */
              { 10, 10, 26,110}, staticText {disabled, "Drive Location"};
              { 38, 10, 54,100}, staticText {disabled, "Drive Icon"};
              { 78, 10, 94,100}, staticText {disabled, "Disk Icon"};
              {110, 10,126,100}, staticText {disabled, "Drive #"};
              {130, 10,146,100}, staticText {disabled, "Drive Type"};
              {150, 10,166,100}, staticText {disabled, "Disk Format"};
              {230, 10,246,145}, staticText {disabled, "Disk Controller Chip:"};
              {230,150,246,190}, staticText {disabled, ""};  /* disk controller type */
              { 10,125, 26,200}, staticText {disabled, ""};  /* drive locations */
              { 10,235, 26,310}, staticText {disabled, ""};
              { 10,345, 26,420}, staticText {disabled, ""};
```

```
            { 30,145, 62,177}, userItem {disabled};       /* drive icons */
            { 30,255, 62,287}, userItem {disabled};
            { 30,365, 62,397}, userItem {disabled};
            { 70,145,102,177}, userItem {disabled};       /* disk icons */
            { 70,255,102,287}, userItem {disabled};
            { 70,365,102,397}, userItem {disabled};
            {110,155,126,167}, staticText {disabled, ""}; /* drive numbers */
            {110,265,126,277}, staticText {disabled, ""};
            {110,375,126,387}, staticText {disabled, ""};
            {130,120,146,210}, staticText {disabled, ""}; /* drive types */
            {130,230,146,320}, staticText {disabled, ""};
            {130,340,146,430}, staticText {disabled, ""};
            {150,135,166,190}, staticText {disabled, ""}; /* disk formats */
            {150,245,166,300}, staticText {disabled, ""};
            {150,355,166,410}, staticText {disabled, ""}
      }
};
```

```
/* descriptive strings */

resource 'STR#' (256, preload) {
      {
            "IWM";                                  /* disk controller types */
            "SWIM";
            "Internal 1";                           /* primary internal drive */
            "Internal 2";                           /* secondary internal drive */
            "External 1";                           /* primary external drive */
            "External 2";                           /* secondary external drive */
            "Single-Sided";                         /* drive types */
            "Double-Sided";
            "SuperDrive";
            "";                                     /* (fillers) */
            "";
            "Hard Disk 20"
      }
};


/* default drive and/or disk icon to use if we get an error trying to get an */
/* icon from the driver (probably because it's a REALLY old driver version)  */

resource 'ICON' (256, preload) {
      $"7FFFFF78"         /*   xxxxxxxxxxxxxxxxxxxxxx xxxx     */
      $"98FFFF84"         /*  x  xx   xxxxxxxxxxxxxxxx    x    */
      $"80FFC702"         /*  x       xxxxxxxxxx   xxx      x   */
      $"90FFC702"         /*  x  x    xxxxxxxxxx   xxx      x   */
      $"80FFC702"         /*  x       xxxxxxxxxx   xxx      x   */
      $"90FFC702"         /*  x  x    xxxxxxxxxx   xxx      x   */
      $"80FFC702"         /*  x       xxxxxxxxxx   xxx      x   */
      $"90FFC702"         /*  x  x    xxxxxxxxxx   xxx      x   */
      $"80FFC702"         /*  x       xxxxxxxxxx   xxx      x   */
      $"90FFFF02"         /*  x  x    xxxxxxxxxxxxxxx      x   */
      $"8B7FFE02"         /*  x   x xx xxxxxxxxxxxxx       x   */
      $"80000002"         /*  x                            x   */
      $"80000002"         /*  x                            x   */
      $"8AAAAAA2"         /*  x   x x x x x x x x x x x x  x   */
      $"90000012"         /*  x  x                      x  x   */
      $"80000002"         /*  x                            x   */
      $"90000012"         /*  x  x                      x  x   */
```

```
        $"80000002"         /*  x                              x   */
        $"90000012"         /*  x  x                        x  x   */
        $"80000002"         /*  x                              x   */
        $"90000012"         /*  x  x                        x  x   */
        $"80000002"         /*  x                              x   */
        $"90000012"         /*  x  x                        x  x   */
        $"80000002"         /*  x                              x   */
        $"90000012"         /*  x  x                        x  x   */
        $"80000002"         /*  x                              x   */
        $"90000012"         /*  x  x                        x  x   */
        $"E0000002"         /*  xxx                            x   */
        $"F0000012"         /*  xxxx                        x  x   */
        $"80000002"         /*  x                              x   */
        $"90000012"         /*  x  x                        x  x   */
        $"7FFFFFFC"         /*   xxxxxxxxxxxxxxxxxxxxxxxxxxxx    */
};
```

Sound Manager

The Sound Manager for Macintosh Portable is the same as that documented for the Macintosh II in *Inside Macintosh*, Volume V, and supplemented by any applicable Technical Notes. The Sound Manager has incorporated the functions of the Sound Driver.

Modem

Support for an internal modem is provided. See Chapter 6, “The Power Manager,” and Chapter 9, “Options.”

Sleep State and Operating State

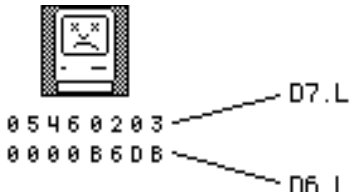
The Macintosh Portable ROM software supports the ability to put the computer into the sleep state (clock to DC, all RAM and registers retained) and to bring it back to the operating state. These functions are implemented in the power manager firmware and the power manager processor. The OS requests the sleep state through a time-out scheme or direct user action. Return to the operating state (waking) is due to an event such as a keystroke or wake-up timer going off. See Chapter 6, “The Power Manager.”

RAM and ROM Expansion

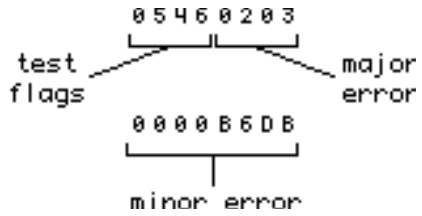
Memory expansion is done using internal RAM expansion cards in the machine and is supported by the ROM. ROM expansion/replacement is likewise available by using an internal expansion connector (slot) to which are brought the necessary signals (see Chapter 5, “Hardware”, for an explanation). The 4 MB of ROM address space between $A0 0000 and $DF FFFF is available to you See the address map, Figure 3-1. Refer to Macintosh Technical Note #255, “Macintosh Portable ROM Expansion,” for additional information.

Diagnostics—The “sad Macintosh” icon

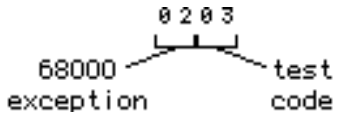
The bootup code in the Macintosh contains a series of startup tests that are run to insure that the fundamental operations of the machine are working properly. If any of those tests fail, a “sad Macintosh” icon appears on the screen with a code below that describes what failure occurred. Here is a typical example of a “sad Macintosh” display with an error code below it:



The two codes are actually the contents of the two CPU data registers D6 and D7. The upper word (upper 4 hex digits, in this case 0546) of D7 contains miscellaneous flags that are used by the start-up test routines and are unimportant to just about everybody except a few test engineers within Apple. The lower word of D7 is the major error code. The major error code identifies the general area the test routines were in when a failure occurred. D6 is the minor error and usually contains additional information about the failure, something like a failed bit mask.



The major error is further broken into the upper byte that contains the number of any 68000 exception that occurred ($00 meaning that no exception occurred), and the lower byte that usually contains the test that was being run at the time of failure. If an unexpected exception occurred during a particular test, then the exception number is logically ORed into the major error code. This way both the exception that occurred as well as the test that was running can be decoded from the major error code:



In this example, the code says that an address error exception ($0200) occurred during the RAM test for Bank A ($03); $0200 ORed with $03 = $0203.

Major error codes

Below is a brief description of the various test codes that might appear in the major error code:

▲ **Warning** Some of these codes may mean slightly different things in Macintosh models other than the Macintosh Portable. These descriptions describe specifically how they are used in the Macintosh Portable. ▲

$01 - ROM test failed. Minor error code is $FFFF, means nothing.

$02 - RAM test failed. Minor error code indicates which RAM bits failed.

$05 - RAM external addressing test failed. Minor error code indicates a failed address line.

$06 - Unable to properly access the VIA 1 chip during VIA initialization. Minor error code not applicable.

$08 - Data bus test at location 8 bytes off of top of memory failed. Minor error code indicates the bad bits as a 16–bit mask for bits 15–00. This may indicate either a bad RAM chip or data bus failure.

$0B - Unable to properly access the SCSI chip. Minor error code not applicable.

$0C - Unable to properly access the IWM (or SWIM) chip. Minor error code not applicable.

$0D - Not applicable to Macintosh Portable. Unable to properly access the SCC chip. Minor error code not applicable.

$0E - Data bus test at location $0 failed. Minor error code indicates the bad bits as a 16–bit mask for bits 15–00. This may indicate either a bad RAM chip or data bus failure.

$10 - ***Macintosh Portable only.*** Video RAM test failed. Minor error code indicates which RAM bits failed.

$11 - ***Macintosh Portable only.*** Video RAM addressing test failed. Minor error code contains the following:

upper word = failed address (16-bit)

msb of lower word = data written

lsb of lower word = data read

Data value written also indicates which address line is being actively tested.

$12 - ***Macintosh Portable only.*** Deleted

$13 - ***Macintosh Portable only.*** Deleted

$14 - ***Macintosh Portable only.*** Power manager processor was unable to turn on all the power to the board. This may have been due to a communication problem with the power manager. If so, the minor error code will contain a power manager error code, explained in the next section.

$15 - ***Macintosh Portable only.*** Power manager failed its self-test. Minor error code contains the following:

msw = error status of transmission to power manager (see "Power manager processor failures (Macintosh Portable only)".

lsw = power manager self-test results (0 means it passed, non-zero means it failed)

$16 - ***Macintosh Portable only.*** A failure occurred while trying to size and configure the RAM. Minor error code not applicable.

Minor error codes—Power manager processor failures (Macintosh Portable only)

If a communication problem occurs during communication with the power manager, the following error codes will appear somewhere in the minor error code (usually in the lower half of the code, but not always):

$CD38	Power manager was never ready to start handshake.
$CD37	Timed out waiting for reply to initial handshake.
$CD36	During a send, power manager did not start a handshake.
$CD35	During a send, power manager did not finish a handshake.
$CD34	During a receive, power manager did not start a handshake.
$CD33	During a receive, power manager did not finish a handshake.

Diagnostic Code Summary

Below is a summarized version of the sad Macintosh error codes:

Test Codes

$01	ROM checksum test.
$02	RAM test.
$05	RAM addressing test.
$06	VIA 1 chip access.
$08	Data bus test at top of memory.
$0B	SCSI chip access.
$0C	IWM (or SWIM) chip access.
$0D	Not applicable to Macintosh Portable. SCC chip access.
$0E	Data bus test at location $0.
$10	Macintosh Portable only. Video RAM test.
$11	Macintosh Portable only. Video RAM addressing test.
$14	Macintosh Portable only. Power manager board power on.
$15	Macintosh Portable only. Power manager self-test.
$16	Macintosh Portable only. RAM sizing.

Power manager communication error codes

$CD38	Initial handshake.
$CD37	No reply to initial handshake.
$CD36	During send, no start of a handshake.
$CD35	During a send, no finish of a handshake.
$CD34	During a receive, no start of a handshake.
$CD33	During a receive, no finish of a handshake.

CPU exception codes (as used by the startup tests)

$0100	Bus error exception code
$0200	Address error exception code
$0300	Illegal error exception code
$0400	Zero divide error exception code
$0500	Check inst error exception code
$0600	cpTrapcc,Trapcc,TrapV exception code
$0700	Privilege violation exception code
$0800	Trace exception code
$0900	Line A exception code
$0A00	Line F exception code
$0B00	Unassigned exception code
$0C00	CP protocol violation
$0D00	Format exception
$0E00	Spurious interrrupt exception code
$0F00	Trap inst exception code
$1000	Interrupt level 1
$1100	Interrupt level 2
$1200	Interrupt level 3
$1300	Interrupt level 4
$1400	Interrupt level 5
$1500	Interrupt level 6
$1600	Interrupt level 7

Script Manager

The Script Manager is part of the ROM image.

Notification Manager

The Notification Manager is part of the ROM image. See Macintosh Technical Note #184, April 2, 1988.