FLOPPY

INTRODUCTION

You are reading the Introduction. First comes a section on the low level 1108 floppy implementation. This includes discussion of FLOPPYIOCBs, DISKADDRESSes, and FLOPPYRESULTs. The low level floppy command functions are described in this section.

Next comes a section on Pilot floppy basics. This section describes file device \PFLOPPYFDEV and the functions installed on \PFLOPPYFDEV which are called by generic FILEIO functions. We describe the peculiar marker pages (PMPAGEs)...

LOW LEVEL FLOPPY

This section describes the lowest level code of the 1108 FLOPPY implementation.

Miscellaneous Global Variables

\FLOPPY.CYLINDERS (Variable)

\FLOPPY.TRACKSPERCYLINDER (Variable)

\FLOPPY.SECTORSPERTRACK (Variable)

The number of cylinders on the floppy, the number of tracks per cylinder on the floppy, and the number of sectors per track in the DATA part of the floppy. On the 1108, \FLOPPY.CYLINDERS=77, \FLOPPY.TRACKSPERCYLINDER=2, and \FLOPPY.SECTORSPERTRACK=15. Some of these values are different for the 1186.

\FLOPPYIOCB and 1108 Input Output Control Blocks

\FLOPPYIOCB (Variable)

Points to the 1108 floppy input output control block. Lisp is run by the CP Central Processor and must communicate via the input output control block to the IOP Input Output Processor the various floppy commands that need to be executed.

It will appear that \FLOPPYIOCB is bound to CURSORBITMAP.EM. This is just a peculiarity of how the Lisp virtual memory works.

An input output control block looks like

FLOPPYIOCB (Datatype)

```
(DATATYPE FLOPPYIOCB
```

```
((\BUFFERLOLOC WORD)
 (\BUFFERHILOC WORD)
 (NIL WORD)
 (SECTORLENGTHDIV2 WORD)
 (TROYORIBM BITS 12)
 (DENSITY BITS 4)
 (DISKADDRESS FIXP)
 (SECTORCOUNT WORD)
 (FLOPPYRESULT WORD)
 (SAMEPAGE FLAG)
 (COMMAND BITS 15)
 (SUBCOMMAND WORD)
 (SECTORLENGTHDIV4 BITS 8)
 (ENCODEDSECTORLENGTH BITS 8)
 (SECTORSPERTRACK BITS 8)
 (GAP3 BITS 8)
 (NIL 3 WORD)))
```

The Lisp code sets up and the IOP reads all fields except for FLOPPYRESULT. The IOP sets and the Lisp code reads the FLOPPYRESULT.

\BUFFERHILOC and \BUFFERLOLOC are the high and low part of the pointer to the page that is being read from or being written to if any. We are not able to put (BUFFER POINTER) instead of (\BUFFERLOLOC WORD) and (\BUFFERHILOC WORD) because \BUFFERHILOC and \BUFFERLOLOC occur in the wrong order. The reason for this is that Mesa pointers are swapped and Lisp pointers are not.

SECTORLENGTHDIV2 is the length in bytes of the sectors being read or written divided by 2.

TROYORIBM is always IBM.

DENSITY is usually DOUBLE. DENSITY is SINGLE for track CYLINDER=0, HEAD=0 on Pilot floppies.

DISKADDRESS is a FIXP encoding a CYLINDER+HEAD+SECTOR combination. Each DISKADDRESS points to a page (aka sector) on the floppy. The particular format of DISKADDRESSes is described elsewhere in this document.

SECTORCOUNT is looked at by the IOP only when the COMMAND is to format the floppy. SECTORCOUNT indicates how many tracks to format.

FLOPPYRESULT is the status result word which is set by the IOP. The FLOPPYRESULT is a set of flags which can be usefully inspected with the FLOPPYRESULT BLOCKRECORD described elsewhere in this document.

SAMEPAGE is always NIL.

COMMAND can be any of the following constants

C.NOP	**(Constant)**
C.READSECTOR	**(Constant)**
C.WRITESECTOR	**(Constant)**
C.FORMATTRACK	**(Constant)**
C.RECALIBRATE	**(Constant)**
C.INITIALIZE	**(Constant)**

SUBCOMMAND is always SC.NOP.

GAP3, SECTORLENGTHDIV4, ENCODEDSECTORLENGTH, and SECTORSPERTRACK are obscure numbers that need to be set up for the IOP when COMMAND is to format.

1108 Floppy DISKADDRESSes

Locations of pages on a floppy are referrred to by the hardware by cylinder+head+sector number combinations called DISKADDRESSes. DISKADDRESSes are represented by FIXPs and are accessed and created with the help of the record declaration

DISKADDRESS (Accessfns)

```
(ACCESSFNS DISKADDRESS
  ((CYLINDER (LRSH DATUM 16))
   (HEAD (LRSH (LOGAND DATUM 65535) 8))
   (SECTOR (LOGAND DATUM 255)))
   (CREATE (IPLUS (LLSH CYLINDER 16)
                  (LLSH HEAD 8)
                  SECTOR)))
```

CYLINDER can be between 0 and \FLOPPY.CYLINDERS.

HEAD can be 0 or 1 to indicate which side of the floppy should be read.

The part of the floppy pointed to by a CYLINDER+HEAD combination is known as a track. Each track contains a certain amount of redundant operation for self checking puposes conducted by the IOP assembly language code plus actual data stored in pages called sectors.

SECTOR can be between 1 and 15 for tracks that are formatted IBM double density 512 bytes per sector. The upper limit for SECTOR is determined according to the formatting of the track as given by the following table

Format	Number of Sectors
IBMS128	26
IBMS256	15
IBMS512	8
IBMS1024	4
IBMD128	36
IBMD256	26
IBMD512	15
IBMD1024	8

The table above can be found coded into the function \FLOPPY.SECTORSPERTRACK.

Pilotconverts between Pilot page numbers and DISKADDRESSes by using the functions \PFLOPPY.PAGENOTODISKADDRESS and \PFLOPY.DISKADDRESSTOPAGENO.

\FLOPPYRESULT and 1108 Result Words

\FLOPPYRESULT (Variable)

Points to the 1108 floppy result word offsetted into the \FLOPPYIOCB input output control block. Lisp is run by the CP

Central Processor and must communicate via the input output control block to the IOP Input Output Processor the various floppy commands that need to be executed. After a command is executed, the Lsip code looks at the status word to see if any error conditions were signalled indicating a failed operation.

It will appear that \FLOPPYRESULT is bound to MOUSEX.EM.This is just a peculiarity of how the Lisp virtual memory works.

An 1108 result word looks like

FLOPPYRESULT **(Blockrecord)**

```
(BLOCKRECORD FLOPPYRESULT
  ((DOOROPENED FLAG)
   (MPERROR FLAG)
   (TWOSIDED FLAG)
   (DISKID FLAG)
   (ERROR FLAG)
   (NIL FLAG)
   (RECALIBRATEERROR FLAG)
   (DATALOST FLAG)
   (NOTREADY FLAG)
   (WRITEPROTECT FLAG)
   (DELETEDDATA FLAG)
   (RECORDNOTFOUND FLAG)
   (CRCERROR FLAG)
   (TRACK0 FLAG)
   (NIL FLAG)
   (BUSY FLAG)))
```

It should be pointed out that some of the error bits in the \FLOPPYRESULT are a bit noisy. Certain errors turn out to be transients that can be overcome simply by reissuing the last command. The bits MPERROR and CRCERROR are notably noisy. The Lisp code takes the first occurence of certain errors as an indication that perhaps the floppy drive has lost its way and needs to be recalibrated. In that case, a recalibrate command is issued and then the command that failed is tried again. After sufficient retrying of a command that fails (the amount of retrying depending on the kind of command and the particular settings of the error bits), the bad news is announced to the user. Most of this retrying and so on has been determined empirically and is coded into the function \FLOPPY.RUN.

DOOROPENED is T if the floppy drive door has opened since the last floppy command. This is an error condition which has to be cleared. (When DOOREOPENED gets set, ERROR also gets set.) To clear this bit, an INITIALIZE command must be issued.

MPERROR is T if the IOP floppy handler tried to crash the machine. We suspect this condition gets set from time to time because of bugs in the IOP assembly language code written by OSD. The MPERROR feature is a patch on top of the OSD assembly language code installed by Mitch Lichtenberg. Instead of crashing the machine the flag MPERROR now gets set and Lisp deals with that. It has turned out to be the case that reissuing the last command to the IOP without change invariably works. When MPERROR is set to T, the remainder of the FLOPPYRESULT word is treated as an error code instead of the flags DOOROPENED, `TWOSIDED,DISKID, ... BUSY`. The MPCODEs stored in the remainder of the FLOPPYRESULT word can be

```
580 Domino NoValidCommand Error
581 Domino UnImplFloppyCmd Error
```

```
582 Domino InvalidEscapeCmd Error
583 Domino CommandTrack Error
584 Domino TrackToBig Error
585 Domino BadDmaChannel Error
586 Domino NoDmaEndCount1 Error
587 Domino NoDmaEndCount2 Error
597 Domino Error In NOOP Patch
598 Domino Error in Reset Patch
```

TWOSIDED is T if the floppy is two sided.

DISKID = T if the floppy drive is Schogart Associated model SA850 and NIL if the floppy drive is Schogart Associated model SA800.

ERROR = T if there was an error in trying to perofrm the last command issued to the IOP. To clear this bit, an INITIALIZE command must be issued.

RECALIBRATEERROR = T if there was an error while recalibrating I suppose. I don't think I've ever seen this flag set in practice except perhaps if you issue a recalibrate command when there is no floppy in the floppy drive. The Lisp code in that situation would tend to error out on a command preceding any recalibration.

DATALOST = T problem reading or writing. Maybe the track of the floppy which is being read from or written into isn't formatted in the way that was expected.

NOTREADY = T implies the IOP is not ready I guess. I'm not sure I've ever seen this.

WRITEPROTECT = T means the floppy is writeprotected. Any attempt to write on the floppy will fail so the Lisp code checks whether this flag is on and signals an error if need be before attempting to open an ouptut stream or format a floppy.

DELETEDDATA = I have no idea.

RECORDNOTFOUND = T problem reading or writing. Maybe the track of the floppy which is being read from or written into isn't formatted in the way that was expected.

CRCERROR = Cyclic Redundancy Check. There is redundant information on the floppy which serves as a check to the IOP assembly language code that the floppy drive head is where it is supposed to be. This error bit is a bit noisy and spurious CRCERRORs can be overcome by reissuing the last command to the IOP. If after several retries the CRCERROR has not gone away, then the CRCERROR is treated as real and the user is hit with the bad news. A hard CRCERROR is caused by the track of the floppy which is being read from or written into not being formatted in the way that was expected.

TRACK0 = T after a recalibrate command. Says that the recalibrate successfully found CYLINDER=0 I guess. Not really paid attention to by the Lisp code.

BUSY = T implies the IOP is busy I guess. I'm not sure if this would mean that the IOP is still preocessing the most recently issued command or not. The official way to wait for the IOP to finish the most recently issued command is with the loop

(while (NOT (ZEROP (fetch (IOPAGE DLFLOPPYCMD) of \IOPAGE))) do (BLOCK))

Low Level 1108 Floppy Command Functions

(\FLOPPY.RUN FLOPPYIOCB NOERROR)

This function makes the IOP really go.

FLOPPYIOCB should already be prepared by other functions like \FLOPPY.NOP, \FLOPPY.READSECTOR, \FLOPPY.WRITESECTOR, \FLOPPY.FORMATTRACKS, \FLOPPY.RECALIBRATE, or \FLOPPY.INITIALIZE described below.

First, if there is a buffer involved (the command is to read or to write), the buffer is locked down by calling the function \FLOPPY.LOCK.BUFFER.

Next, the FLOPPYIOCB passed in is \BLTed on to \FLOPPYIOCB. \FLOPPYIOCB points to the beginning of the 16 words in real memory that the IOP will look at and interpret as an input output control block to be processed.

Next, the IOP is notified that there is an input output control block in need of processing via

(replace (IOPAGE DLFLOPPYCMD) of \IOPAGE with \FLOPPYIOCBADDR)

The IOP periodically looks at the \IOPAGE for things to do and acts when it sees a nonzero DLFLOPPYCMD field in the \IOPAGE. After replacing the DLFLOPPYCMD field in the \IOPAGE, the Lisp code must wait until the IOP finishes via the loop

(while (NOT (ZEROP (fetch (IOPAGE DLFLOPPYCMD) of \IOPAGE))) do (BLOCK))

After this loop finishes, \FLOPPY.RUN looks at the \FLOPPYRESULT result word to see if any error flags have been set by the IOP. Supposing things have gone well, \FLOPPY.UNLOCK.BUFFER is called to unlock the buffer (if any) pointed to by the FLOPPYIOCB and \FLOPPY.RUN returns T indicating success.

If error bits in the \FLOPPYRESULT result word have been set, then \FLOPPY.RUN may try to recover in certain ways. This may involve reissuing a command and/or some intervening recalibration commands.

If an error persists, then \FLOPPY.RUN returns NIL if NOERROR = T and otherwise a break and an error message happen to the user.

(\FLOPPY.LOCK.BUFFER FLOPPYIOCB)

Calls \LOCKPAGES on the buffer pointed to by the FLOPPYIOCB \BUFFERLOLOC and \BUFFERHILOC fields. It is necessary that the buffer be locked down before a command can be issued to the IOP. This function also does \GETBASEing and \PUTBASEing into the BUFFER to make the BUFFER look dirty to the IOP. This is known as "touching pages". A reasonable person would think that this touching ought not to be forced on the Lisp floppy handler, but that's just the way it is. If the Lisp floppy handler doesn't touch the buffer, the IOP will sometimes cause a fatal 510 crash. \FLOPPY.LOCK.BUFFER gets called by \FLOPPY.RUN before a

command to the IOP is allowed to begin exectuing.

(\FLOPPY.UNLOCK.BUFFER FLOPPYIOCB)

Calls \UNLOCKPAGES on the buffer pointed to by the FLOPPYIOCB. \FLOPPY.UNLOCK.BUFFER gets called by \FLOPPY.RUN after a command to the IOP has finished executing.

(\FLOPPY.NOP NOERROR)

Calls \FLOPPY.RUN to perform a NOP command. This command can be used to get the state of the DOOROPENED and WRITEPROTECT flags of the \FLOPPYRESULT without doing actual operations.

(\FLOPPY.READSECTOR FLOPPYIOCB DISKADDRESS PAGE NOERROR)

Calls \FLOPPY.RUN to perform a read. PAGE should be a virtual memor page VMEMPAGEP. Typically PAGE is a virtual memory page that has worked its way down from FILEIO functions to \FLOPPY.READSECTOR via the functions \PFLOPPY.READPAGES and \PFLOPPY.READPAGENO. FLOPPYIOCB will normally be \FLOPPY.IBMD512.FLOPPYIOCB. Data at the page of the floppy located at DISKADDRESS is read into PAGE.

(\FLOPPY.WRITESECTOR FLOPPYIOCB DISKADDRESS PAGE NOERROR)

Calls \FLOPPY.RUN to perform a write. PAGE should be a virtual memor page VMEMPAGEP. Typically PAGE is a virtual memory page that has worked its way down from FILEIO functions to \FLOPPY.WRITESECTOR via the functions \PFLOPPY.WRITEPAGES and \PFLOPPY.WRITEPAGENO. FLOPPYIOCB will normally be \FLOPPY.IBMD512.FLOPPYIOCB. The contents of PAGE are written into the page of the floppy located at DISKADDRESS.

(\FLOPPY.FORMATTRACKS FLOPPYIOCB DISKADDRESS KOUNT NOERROR)

Calls \FLOPPY.RUN to perform formatting. This function will format KOUNT tracks on the

(fetch (DISKADDRESS HEAD) of DISKADDRESS)

side of the floppy beginning with cylinder

(fetch (DISKADDRESS CYLINDER) of DISKADDRESS)

The quantity (fetch (DISKADDRESS SECTOR) of DISKADDRESS) is ignored for purposes of formatting.

KOUNT is spelled with a K just because COUNT is a CLISP word and CLISP gets in your way if you spell the word with a C.

The kind of formatting that takes place depends on the kind of FLOPPYIOCB. If FLOPPYIOCB is \FLOPPY.IBMD512.FLOPPYIOCB, then tracks are formatted IBM double density 512 bytes per sector. If FLOPPYIOCB is \FLOPPY.IBMD256.FLOPPYIOCB, then tracks are formatted IBM double density 256 bytes per sector.If FLOPPYIOCB is \FLOPPY.IBMS128.FLOPPYIOCB, then tracks are formatted IBM

single density 128 bytes per sector.

Pilot floppies are formatted this way:

CYLINDER=0, HEAD=0 => IBMS128 format

CYLINDER=0, HEAD=1 => IBMD256 format

CYLINDER >0, HEAD=0 or 1 => IBMD512 format

(\FLOPPY.RECALIBRATE NOERROR)

This function is called to ask the floppy drive hardware to recalibrate itself. The floppy drive head positions itself over tracks on a floppy with the aid of a stepping motor which steps in from CYLINDER=0 which is specially recognizable. The stepping motor can slip some and it is occasionally necessary to ask the floppy drive hardware to recalibrate itself which means refind CYLINDER=0. CYLINDER = 0 is kind of a light pole on a very dark street for the floppy hardware. Once the hardware is back in the light of CYLINDER = 0, it knows where it is again.

\FLOPPY.RECALIBRATE is sometimes called by the Lisp implementation after certain errors are detected while performing normal commands like reading and writing. The assumption in those situations is that the stepping motor has slipped some and that recalibrating will make things well enough again that the command which failed will succeed if it is reissued after the recalibration. (Of course, after enough times of trying this strategy, the error just has to be announced to the user.)

\FLOPPY.RECALIBRATE is also called at the beginning of certain major operations. For example, \FLOPPY.RECALIBRATE is called several times by the formatting function \PFLOPPY.FORMAT. After all, if we're going to format a floppy which involves writing on to the floppy, we want to do the best job we can.

(\FLOPPY.INITIALIZE NOERROR)

\FLOPPY.INITIALIZE initializes the IOP assembly language code floppy handler which Lisp must talk with. This function has to be the first function that is called when FLOPPY is started up. Thus, \FLOPPY.EVENTFN calls \FLOPPY.INITIALIZE after coming back from a LOGOUT, SYSOUT, MAKESYS, or SAVEVM.

\FLOPPY.INITIALIZE also has to be called to clear error conditions as they arise. This might happen, for example, if the user trys to read from the floppy drive but the floppy drive door is open or there is no floppy. This would set the DOOROPENED error bit in the \FLOPPYRESULT word. To clear these error conditions like DOOROPENED, \FLOPPY.INITIALIZE needs to be called.

PILOT FLOPPY

This section describes the connection between FLOPPY and FILEIO. FILEIO is the system source file that makes device independent part of operations like OPENSTREAM, READ, PRINT, CLOSEF, RENAMEFILE, DELFILE, and DIRECTORY possible. FILEIO ultimately winds up calling functions stored in the fields of

file device records.

\PFLOPPYFDEV

\PFLOPPYFDEV (Variable)

The major FDEV record for floppy is bound to \PFLOPPYFDEV. At the time of this writing, \PFLOPPYFDEV looks like the following image

---

{FDEV}#66,3400 Inspector

DEVICENAME	FLOPPY
RESETABLE	T
RANDOMACCESSP	T
NODIRECTORIES	T
PAGEMAPPED	T
FDBINABLE	T
FDBOUTABLE	T
FDEXTENDABLE	T
BUFFERED	T
REMOTEP	NIL
SUBDIRECTORIES	NIL
CLOSEFILE	\PFLOPPY.CLOSEFILE
DELETEFILE	\PFLOPPY.DELETEFILE
DIRECTORYNAMEP	TRUE
EVENTFN	\FLOPPY.EVENTFN
GENERATEFILES	\PFLOPPY.GENERATEFILES
GETFILEINFO	\PFLOPPY.GETFILEINFO
GETFILENAME	\PFLOPPY.GETFILENAME
HOSTNAMEP	\FLOPPY.HOSTNAMEP
OPENFILE	\PFLOPPY.OPENFILE
READPAGES	\PFLOPPY.READPAGES
REOPENFILE	\PFLOPPY.OPENFILE
SETFILEINFO	\PFLOPPY.SETFILEINFO
TRUNCATEFILE	\PFLOPPY.TRUNCATEFILE
WRITEPAGES	\PFLOPPY.WRITEPAGES
BIN	\BUFFERED.BIN
BOUT	\BUFFERED.BOUT
PEEKBIN	\BUFFERED.PEEKBIN
READP	\PAGEDREADP
BACKFILEPTR	\PAGEDBACKFILEPTR
DEVICEINFO	{PINFO}#65,4764
FORCEOUTPUT	\PAGED.FORCEOUTPUT
LASTC	NIL
SETFILEPTR	\PAGEDSETFILEPTR
GETFILEPTR	\PAGEDGETFILEPTR
GETEOFPTR	\PAGEDGETEOFPTR
EOFP	\PAGEDEOFP
BLOCKIN	\BUFFERED.BINS
BLOCKOUT	\BUFFERED.BOUTS
RENAMEFILE	\PFLOPPY.RENAMEFILE
RELEASEBUFFER	NIL
GETNEXTBUFFER	\PAGED.GETNEXTBUFFER
SETEOFPTR	\PAGED.SETEOFPTR
FREEPAGECOUNT	NIL
MAKEDIRECTORY	NIL
WINDOWOPS	NIL
WINDOWDATA	NIL
CHECKFILENAME	NIL
HOSTALIVEP	NIL
OPENP	\GENERIC.OPENP
OPENFILELST	NIL

REGISTERFILE \ADD-OPEN-STREAM

Pilot Floppy FDEV Functions

The following FLOPPY functions can be called by FILEIO

(\PFLOPPY.OPENFILE FILE ACCESS RECOG OTHERINFO FDEV OLDSTREAM)

Gets called when FILEIO opens a stream for input or output. If input, then \PFLOPPY.OPENOLDFILE eventually gets called. If output, then \PFLOPPY.OPENNEWFILE eventually gets called.

\PFLOPPY.OPENFILE returns a stream datatype. The DEVICE of the stream will be \PFLOPPYFDEV. Two other fields on the stream, F1 and F2, point to the allocation record (PFALLOC) and leader page (PLPAGE) for the stream. The PFALLOC and PLPAGE can be conveniently accessed by using the FLOPPYSTREAM ACCESSFNS. When other floppy functions are passed the stream to work with, FLOPPY looks at the arguments passed, the fields of the stream, and the fields of the PFALLOC and PLPAGE to determine how to act.

\PFLOPPY.DIR.GET is called to search for the allocation record of an old FILE. \PFLOPPY.DIR.PUT is called to store the allocation record of a newly created file.

(\PFLOPPY.READPAGES STREAM FIRSTPAGE# BUFFERS)

Reads sectors off floppy into virtual memory pages BUFFERS. FIRSTPAGE# is in FILEIO's scheme of counting the pages of a file, beginning with 0 for the first page of a file. \PFLOPPY.READPAGES therefore fills BUFFERS with data read from the floppy beginning with the FIRSTPAGE# of STREAM.

FIRSTPAGE# is in FILEIO's scheme of counting the pages of a file. FILEIO page numbers are converted into Pilot page numbers lieing somewhere between \PFLOPPYFIRSTDATAPAGE and \PFLOPPYLASTDATAPAGE. Pilot page numbers are in turn converted into DISKADDRESSes which record head, sector, and cylinder of a page on a floppy.

\PFLOPPY.READPAGES calls \FLOPPY.READPAGE which computes a Pilot page number from the stream's PFALLOC and the FILEIO page number passed in as an argument.

\FLOPPY.READPAGE calls \PFLOPPY.PAGENOTODISKADDRESS to convert the Pilot page number into a DISKADDRESS and then calls \FLOPPY.READSECTOR to read the sector at the computed disk address.

(\PFLOPPY.WRITEPAGES STREAM FIRSTPAGE# BUFFERS)

Other than that writing is taking place instead of reading, \PFLOPPY.WRITEPAGES is similar to \PFLOPPY.READPAGES.

Writes contents of virtual memory pages BUFFERS on to sectors of floppyfills BUFFERS with data . FIRSTPAGE# is in FILEIO's scheme of counting the pages of a file, beginning with 0 for the first page of a file. \PFLOPPY.WRITEPAGES therefore writes to the floppy beginning with the location corresponding to the

FIRSTPAGE# of STREAM.

FIRSTPAGE# is in FILEIO's scheme of counting the pages of a file. FILEIO page numbers are converted into Pilot page numbers lieing somewhere between \PFLOPPYFIRSTDATAPAGE and \PFLOPPYLASTDATAPAGE. Pilot page numbers are in turn converted into DISKADDRESSes which record head, sector, and cylinder of a page on a floppy.

\PFLOPPY.WRITEPAGES calls \FLOPPY.WRITEPAGE which computes a Pilot page number from the stream's PFALLOC and the FILEIO page number passed in as an argument.

\FLOPPY.WRITEPAGE calls \PFLOPPY.PAGENOTODISKADDRESS to convert the Pilot page number into a DISKADDRESS and then calls \FLOPPY.WRITESECTOR to write the sector at the computed disk address.

(\PFLOPPY.TRUNCATEFILE FILE LASTPAGE LASTOFFSET)

Called just before closing an output file. The effect as far as FLOPPY is concerned is to take the allocation record (PFALLOC) for FILE and split the allocation record into two records if necessary. The first PFALLOC created this way is just big enough to store the truncated file. The second PFALLOC created this way becomes a free block. Since FLOPPY does not know how big an output file will turn out to be when it is first opened, FLOPPY must go through the process of allocating a reasonable size block, growing the block on occasion when \PFLOPPY.WRITEPAGES is about to cause the block to overflow, and finally truncate--i.e. split--the block into actual file and free block when \PFLOPPY.TRUNCATEFILE gets called.

\PFLOPPY.TRUNCATEFILE calls \PFLOPPY.TRUNCATE which does the actual splitting of a PFALLOC . \PFLOPPY.TRUNCATE splits a PFALLOC into two PFALLOCs equal to file and free block. The list of PFALLOCS cached on the floppy file device is updated and sufficiently many \PFLOPPY.WRITEPAGENOs of new or updated Pilot marker pages are written out to the floppy.

(\PFLOPPY.CLOSEFILE FILE)

Called to close a file. Writes out the leader page (PLPAGE) of FILE and the two marker pages that go around the leader page+ file.

(\PFLOPPY.DELETEFILE FILE FDEV)

Called to delete a file. \PFLOPPY.DIR.GET gets called to search for the allocation record for FILE. \PFLOPPY.DIR.REMOVE gets called to remove the allocation record from the cached directory alist. \PFLOPPY.DEALLOCATE gets called to turn the removed PFALLOC into a free block. The two marker pages surrounding FILE are updated to indicate that there is a free block between the marker pages and then they are written out. (Otherwise, at least until the free block gets used for other purposes, the contents of the file are still there, but have become inaccessible.)

(\PFLOPPY.RENAMEFILE OLDDEVICE OLDFILE NEWDEVICE NEWFILE OLDRECOG NEWRECOG)

Called to rename a file. \PFLOPPY.DIR.GET gets called to search for the allocation record for FILE. The PFALLOC found in the directory alist is removed with \PFLOPPY.DIR.REMOVE and then

reentered with \PFLOPPY.DIR.PUT under the new file name which \PFLOPPY.RENAMEFILE computes. Finally, the leader page for the PFALLOC is changed to have the new file name and then is written out to the floppy.

(\PFLOPPY.GETFILEINFO FILE ATTRIBUTE FDEV)

Called by GETFILEINFO. All file attributes like WRITEDATE, CREATIONDATE, LENGTH, and TYPE are stored on the leader page of a file. \PFLOPPY.GETFILEINFO returns values, sometimes suitably converted, out of the leader page.

(\PFLOPPY.SETFILEINFO FILE ATTRIBUTE VALUE)

Called by SETFILEINFO. All file attributes like WRITEDATE, CREATIONDATE, LENGTH, and TYPE are stored on the leader page of a file. The VALUE for ATTRIBUTE, sometimes suitably converted, is stored into the leader page. The updated leader page is then written out to the floppy if FILE is not open. (If FILE is open then the leader page will be written out to the floppy when FILE is closed.)

(\PFLOPPY.GETFILENAME FILE RECOG FDEV)

Called by FINDFILE and INFILEP. This function can get called if {FLOPPY} (or a directory name like {FLOPPY}<MYDIR>) is on the user's DIRECTORIES search path. When a function like OPENSTREAM fails to find a file on the user's connected directory, \PFLOPPY.GETFILENAME may get asked about FILE to see if FILE is on {FLOPPY}.

\PFLOPPY.GETFILENAME is coded very similar to \PFLOPPY.DIR.GET which searches the floppy directory alist stored in (fetch (PFLOPPYFDEV DIR) of \FLOPPYFDEV).

\PFLOPPY.GETFILENAME returns NIL if no file is found. NIL is also returned if there is no floppy in the floppy drive or if there is no floppy drive on the machine that the user is using.

(\PFLOPPY.GENERATEFILES FDEV PATTERN DESIREDPROPS OPTIONS)

Called by DIRECTORY and the DIR LISPXMACRO. Like all corresponding GENERATEFILES functions installed on other file devices, \PFLOPPY.GENERATEFILES returns a file generator FILEGENOBJ which is a record that contains a GENFILESTATE plus two function names--in FLOPPY's case, \PFLOPPY.NEXTFILEFN and \PFLOPPY.FILEINFOFN.

The GENFILESTATE is a record that amounts to a stack of all the desired FLOPPY files according to the PATTERN, DESIREDPROPS, and OPTIONS that have been supplied. The files that go into the GENFILESTATE are collected by walking along the floppy directory alist which is stored on (fetch (PFLOPPYFDEV DIR) of \FLOPPYFDEV). A lot of the work is done by calling the function DIRECTORY.MATCH.

\PFLOPPY.NEXTFILEFN takes a GENFILESTATE, pops a file name out of the GENFILESTATE which also side effects the GENFILESTATE, and then returns the file name. If the GENFILESTATE has gone empty, then \PFLOPPY.NEXTFILEFN just returns NIL.

\PFLOPPY.FILEINFOFN is asked to supply information about the

file that is at the top of the GENFILESTATE stack. This is done quite easily by calling the same function (\PFLOPPY.GETFILEINFO1) that \PFLOPPY.GETFILEINFO calls.

(\FLOPPY.HOSTNAMEP NAME FDEV)

This kind of function is intended for file devices that have nicknames. The HOSTNAMEP functions get called when the user specifies a file name containing a nickname as the host part of the file name. Since {FLOPPY} doesn't have any nicknames, \FLOPPY.HOSTNAMEP returns T iff NAME is FLOPPY.

(\FLOPPY.EVENTFN FDEV EVENT)

This function gets called before and after any LOGOUT, SYSOUT, MAKESYS, or SAVEVM.

Pilot Page Numbers

The hardware refers to locations of pages on a floppy by cylinder, head, and sector numbers. This way of enumerating pages on a floppy is cumbersome and the OSD Pilot design makes the somewhat sensible choice of making up its own numbering system for the pages on a floppy. We refer to these numbers as pilot page numbers.

\PFLOPPYFIRSTDATAPAGE (Variable)

\PFLOPPYLASTDATAPAGE (Variable)

Each cylinder+head+sector combination can be encoded as a single FIXP called a DISKADDRESS. The functions to convert between Pilot page numbers and DISKADDRESSes are

(\PFLOPPY.PAGENOTODISKADDRESS PAGENO)

(\PFLOPPY.DISKADDRESSTOPAGENO DISKADDRESS)

The functions to read and write pages to particular pilot page numbers are

(\PFLOPPY.READPAGENO PAGENO PAGE NOERROR)

(\PFLOPPY.WRITEPAGENO PAGENO PAGE NOERROR)

Pilot Marker Pages

Marker pages delimit the files and free blocks found on a Pilot floppy. The pattern of the data stored on a floppy is

DATA = MP ALLOC MP ALLOC MP ALLOC MP ... MP ALLOC MP

Generally a ALLOC is a free block or a leader page + file combination. One ALLOC is the filelist.

The declaration for Pilot floppy marker pages is

PMPAGE (Datatype)

(DATATYPE PMPAGE

```
((SEAL WORD)
 (VERSION WORD)
 (* Previous marker page entry *)
 (PLENGTH SWAPPEDFIXP)
 (PTYPE WORD)
 (PFILEID SWAPPEDFIXP)
 (PFILETYPE WORD)
 (NIL 121 WORD)
 (* Next marker page entry *)
 (NLENGTH SWAPPEDFIXP)
 (NTYPE WORD)
 (NFILEID SWAPPEDFIXP)
 (NFILETYPE WORD)
 (NIL 121 WORD)))
```

SEAL.PMPAGE **(Constant)**

VERSION.PMPAGE **(Constant)**

The SEAL and VERSION fields of a PMPAGE are the same for all marker pages. The magic constants are SEAL.PMPAGE and VERSION.PMPAGE. The SEAL.PMPAGE magic constant is arbitrary enough that there is only a slight chance that a non marker page would begin with this particular word. Hence, the scavenger can search for marker pages by looking for pages that begin with SEAL.PMPAGE.

The fields PLENGTH, PTYPE, PFILEID, PFILETYPE describe the ALLOC preceding the marker page. The fields NLENGTH, NTYPE, NFILEID, NFILETYPE describe the ALLOC following the marker page.

PLENGTH, NLENGTH = Length of ALLOC in pages.

PTYPE, NTYPE = Free, file, or filelist.

PFILEID, NFILEID = Pretty worthless. Used by Mesa. Each ALLOC has its own fileid number in the Mesa floppy handler. Not used by the Xerox Lisp software.

PFILETYPE, NFILETYPE =Free, file, or filelist. At first glance you might think that there isn't much point in having both PTYPE, NTYPE and PFILETYPE, NFILETYPE. If you think that, you are right. Please remember before you throw the rotten oranges that this is not my design. Blame it on OSD.

Pilot Leader Pages

The pattern of the data stored on a floppy is

DATA = MP ALLOC MP ALLOC MP ALLOC MP ... MP ALLOC MP

Generally an ALLOC is a free block or a leader page + file combination. One ALLOC is the filelist. Thus the possible patterns for an ALLOC look like

ALLOC = FREE
ALLOC = LP FILE
ALLOC = FILELIST

We note that all leader pages that occur on a Pilot floppy are immediately preceded by a marker page. All files on a Pilot floppy

are preceded by a leader page. Each leader page + file combination is surrounded by two marker pages.

An exception to the rule: Microcode files on boot floppies do not have leader pages. They look like ALLOC = FILE. You may wonder why a microcode file's ALLOC shouldn't also be ALLOC = LP FILE. There isn't any reason for the exception. This is just another feature of the OSD design.

The declaration for Pilot floppy leader pages is

PLPAGE (Datatype)

```
(DATATYPE PLPAGE
  ((SEAL WORD)
    (VERSION WORD)
    (MESATYPE WORD)
    (* Offset 6 *)
    (\CREATIONDATE SWAPPEDFIXP)
    (\WRITEDATE SWAPPEDFIXP)
    (PAGELENGTH SWAPPEDFIXP)
    (HUGEPAGESTART SWAPPEDFIXP)
    (HUGEPAGELENGTH SWAPPEDFIXP)
    (HUGELENGTH SWAPPEDFIXP)
    (\NAMELENGTH WORD)
    (NAMEMAXLENGTH WORD)
    (* Offset 17 *)
    (\NAME 50 WORD)
    (* Offset 67 *)
    (UFO1 WORD)
    (UFO2 WORD)
    (DATAVERSION WORD)
    (\TYPE WORD)
    (NIL 183 WORD)
    (\BYTESIZE WORD))
```

The SEAL and VERSION fields of a PLPAGE are the same for all marker pages. The magic constants are SEAL.PLPAGE and VERSION.PLPAGE. The SEAL.PLPAGE magic constant is arbitrary enough that there is only a slight chance that a non marker page would begin with this particular word. Hence, the scavenger can search for marker pages by looking for pages that begin with SEAL.PLPAGE.

MESATYPE is fairly obscure. I think it marks the file according to what kind of application produced the file. Each application can register itself with OSD who doles out the numbers.

\CREATIONDATE is the obvious. However, the numeric encryption is according to Mesa's standards.

\WRITEDATE is the obvious. However, the numeric encryption is according to Mesa's standards.

PAGELENGTH is the length of the file after the leader page in pages.

HUGEPAGESTART, HUGEPAGELENGTH, HUGELENGTH are pretty arcane. For ordinairy files, HUGEPAGESTART = 0 and HUGEPAGELENGTH = PAGELENGTH. For all files, HUGELENGTH = the length of the file in bytes. For sysout and huge pilot files, it is possible that HUGEPAGESTART > 0 and HUGEPAGELENGTH < PAGELENGTH. Or some funny business like this.

\NAMELENGTH, NAMEMAXLENGTH, \NAME indicate the name of the file following the leader page. Oddly enough, NAMEMAXLENGTH is just a constant field for all leader pages.

UFO1, UFO2. Flying saucers. Treated as constant fields by Lisp.

DATAVERSION. Another constant field.

\TYPE = binary or text.

Pilot Sector 9

The name of the floppy is stored on the page at location CYLINDER = 0, HEAD = 0, SECTOR = 9. All of CYLINDER = 0 is wasted except for sector 9 . There is nothing valuable stored on the 8 sectors before sector 9, or any of the sectors on CYLINDER = 0.

All the tracks from CYLINDER=1 to \FLOPPY.CYLINDERS are formatted IBM double density 512 bytes per sector. Strangely, the two sides of CYLINDER = 0 are formatted differently. As part of the OSD design, CYLINDER = 0, HEAD = 0 is formatted IBM single density 128 bytes per sector and CYLINDER = 0, HEAD = 1 is formatted IBM double density 256 bytes per sector.

The pattern of the data stored in sector 9 is conveniently represented by a datatype called PSECTOR9.

PSECTOR9 **(Datatype)**

```
(DATATYPE PSECTOR9
  ((SEAL WORD)
    (VERSION WORD)
    (CYLINDERS WORD)
    (TRACKSPERCYLINDER WORD)
    (SECTORSPERTRACK WORD)
    (PFILELISTSTART WORD)
    (PFILELISTFILEID SWAPPEDFIXP)
    (PFILELISTLENGTH WORD)
    (ROOTFILEID SWAPPEDFIXP)
    (NIL WORD)
    (PILOTMICROCODE WORD)
    (DIAGNOSTICMICROCODE WORD)
    (GERM WORD)
    (PILOTBOOTFILE WORD)
    (FIRSTALTERNATESECTOR WORD)
    (COUNTBADSECTORS WORD)
    (NEXTUNUSEDFILEID SWAPPEDFIXP)
    (CHANGING FLAG)
    (NIL BITS 15)
    (\LABELLENGTH WORD)
    (\LABEL 106 WORD)))
```

Lisp verifies that the SEAL of the PSECTOR9 that it reads in is equal to magic constant SEAL.PSECTOR9. This action serves to check that the floppy that is being read is in fact a Pilot floppy.

A floppy's name is stored in \LABELLENGTH and \LABEL. The floppy name is accessed by the user through the function FLOPPY.NAME.

Everything else about PSECTOR9 is not useful to Lisp and is not

used by any of the Lisp code. However Mesa does use some of this cruft and Lisp has to keep the cruft consistent with what Mesa would like to see.

Occasionally, the filelist (also useless) moves around on the floppy. When this happens, PFILELISTSTART, PFILELISTFILEID, PFILELISTLENGTH also have to be updated.

NEXTUNUSEDFILEID has to do with the Mesa floppy handler's notion of what a fileid is. This notion is irrelevant to the Lisp implementation of FLOPPY, but the Lisp implementation of FLOPPY does have to keep NEXTUNUSEDFILEID equal to one plus the number of files currently stored on the floppy.

Every time a new file is created and closed, NEXTUNUSEDFILEID has to be increased by one, the filelist has to change, and both the PSECTOR9 and the filelist have to be written out to the floppy.

(\PFLOPPY.SAVE.PSECTOR9 NOERROR)

Saves the cached PSECTOR9 out to the floppy.

Pilot Filelist

The Pilot filelist is not used by the Lisp FLOPPY but must be maintained by the Lisp FLOPPY code to be compatible with the Mesa floppy handler. The filelist has the form

FILELIST = SEAL VERSION NENTRIES MAXENTRIES ENTRY
ENTRY ENTRY ... ENTRY

Every time a new file is created and closed, an ENTRY has to be added to FILELIST and FILELIST has to be written out. Every time a file is deleted, an ENTRY has to be deleted from FILELIST and FILELIST has to be written out. Other than to maintain the FILELIST in this way for Mesa's benefit, Lisp does not use the FILELIST. Lisp uses the marker pages on a floppy to find where the files are and could care less about the filelist.

The filelist and filelist entry record declarations are

```
(BLOCKRECORD PFILELIST
  ((SEAL WORD)
    (VERSION WORD)
    (NENTRIES WORD)
    (MAXENTRIES WORD)))

(DATATYPE PFLE
  ((FILEID SWAPPEDFIXP)
    (TYPE WORD)
    (START WORD)
    (LENGTH WORD)))
```

(\PFLOPPY.ADD.TO.PFILELIST PFALLOC)

Create a PFLE file list entry on the filelist for this PFALLOC.

(\PFLOPPY.DELETE.FROM.PFILELIST PFALLOC)

Deletes the PFLE file list entry on the filelist corresponding to this PFALLOC.

(\PFLOPPY.SAVE.PFILELIST NOERROR)

Saves the filelist out to the floppy.

Pilot Floppy Format

The overall architecture of a Pilot floppy looks like this

FLOPPY = CYLINDER0 DATA

CYLINDER0 = CYLINDER0HEAD0 CYLINDER0HEAD1

CYLINDER0HEAD0 = GARBAGE SECTOR9 GARBAGE

CYLINDER0HEAD1 = GARBAGE

DATA = DATA = MP ALLOC MP ALLOC MP ... MP ALLOC MP

ALLOC = FREE
ALLOC = LP FILE
ALLOC = FILELIST

Exactly one ALLOC is the FILELIST. SECTOR9 and the FILELIST are for the most part useless. However, the Lisp implementation has to maintain SECTOR9 and FILELIST in a way that will make the Mesa floppy handler happy.

The Lisp implementation finds out what files are present on a floppy by following the marker pages (MPs) in the DATA which always begins at \PFLOPPYFIRSTDATAPAGE andends at \PFLOPPYLASTDATAPAGE. There are fields on the marker pages telling how long and what kind of ALLOC appears on either side of the marker page.

The formatting function for Pilot floppies is

(\PFLOPPY.FORMAT NAME AUTOCONFIRMFLG SLOWFLG)

\PFLOPPY.FORMAT formats the tracks of the floppy according to the peculiar OSD design and then makes the floppy look like it is a Pilot floppy with an empty directory. The DATA on a freshly formatted floppy has the form

DATA = MP FILELIST MP FREE MP

Cached Pilot Floppy Information

Duplicate directory information about a floppy is cached in core. This is held on to by a PFINFO datatype.

PFINFO (Datatype)

(DATATYPE PFINFO (OPEN PFILELIST PFALLOCS DIR PSECTOR9))

We may describe the fields of the PFINFO slightly out of order.

OPEN is a flag saying whether directory information for the current floppy has been cached or not in th remaining fields of the PFINFO.

PFILELIST is the Pilot filelist. The filelist is unimportant except that it must be maintained for Mesa's benefit. An updated PFILELIST has to be written out each time a file is written etc.

PSECTOR9 is the Pilot sector 9 record. This is where the floppy name lives. Otherwise, this record is also pretty worthless as far as Lisp is concerned. But there are fields on the PSECTOR9 that have to be maintained for Mesa's benefit. An updated PFILELIST has to be written out each time a file is written etc.

DIR is the cached flopy directory alist which is an alist of alists of alists.

PFALLOCS is a list of PFALLOC records which describe successive allocations on the floppy.

Cached Pilot Floppy DIR Alist

The PFINFO DIR is the cached flopy directory alist which is an alist of alists of alists. It has the form

```
((name (extension (version . PFALLOC) ...
                          (version . PFALLOC)) ...
               (extension (version . PFALLOC) ...
                          (version . PFALLOC))) ...
 (name (extension (version . PFALLOC) ...
                          (version . PFALLOC)) ...
               (extension (version . PFALLOC) ...
                          (version . PFALLOC))))
```

Say for example that we have 6 versions of the file {FLOPPY}FLOPPY.TEDIT stored on a floppy. Then the cached DIR alist could look like

```
((FLOPPY (TEDIT (1 . {PFALLOC}#55,72740)
                (2 . {PFALLOC}#55,72704)
                (3 . {PFALLOC}#55,72650)
                (4 . {PFALLOC}#55,72434)
                (5 . {PFALLOC}#55,72400)
                (6 . {PFALLOC}#55,72340))))
```

Functions to manage and access the DIR alist are

(\PFLOPPY.DIR.GET FILENAME RECOG)

(\PFLOPPY.DIR.PUT FILENAME RECOG PFALLOC)

(\PFLOPPY.DIR.REMOVE PFALLOC)

(\PFLOPPY.DIR.VERSION VERSION RECOG VALIST FILENAME)

Cached Pilot Floppy PFALLOCS

The PFINFO PFALLOCS is a list of PFALLOC records which

describe successive allocations on the floppy. Some of the PFALLOCs are leader page+file combinations. Some are free blocks. One of the PFALLOCs is the allocation record for the filelist (this PFALLOC tells us where the contents of PFILELIST should be stored).

Each PFALLOC record has the form

PFALLOC **(Datatype)**

```
(DATATYPE PFALLOC
  (FILENAME
    (PREV FULLXPOINTER)
    NEXT
    START
    PMPAGE
    PLPAGE
    PFLE
    (WRITEFLG FLAG)
    (DELETEFLG FLAG)))
```

FILENAME is the name of the file that the PFALLOC corresponds to. If the PFALLOC does not correspond to a file, then FILENAME will be a list like (FREE) or (PFILELIST).

PREV points back to the preceding PFALLOC in the PFALLOCS list.

NEXT points to the next PFALLOC in the PFALLOCS list.

START is the pilot page number of the first page of storage on the floppy corresponding to the PFALLOC (the page after the marker page preceding the allocation).

PMPAGE is a cached copy of the marker page preceding the allocation corresponding to the PFALLOC.

PLPAGE is a cached copy of the leader page, if any, that begins the allocation. (Only file allocations will have leader pages. If the PFALLOC does not correspond to a file allocation then the PLPAGE field is left NIL.)

PFLE is the file list entry in the Pilot filelist corresponding to this PFALLOC.

WRITEFLG = T if there is a stream writing to the allocation corresponding to this PFALLOC. That is, this PFALLOC corresponds to a file, and a stream is writing to that file.

DELETEFLG = T if the PFALLOC corresponds to a file whose deletion is pending. This may happen if some process is reading a file that a second process DELFILEs. The PFALLOC is deleted for real (i.e. made into a free block) when the first process gives up control of the stream that is reading the file with CLOSEF.

PILOT FLOPPY STORAGE ALLOCATION

This section describes the Pilot floppy storage allocation strategy.

(\PFLOPPY.ALLOCATE LENGTH)

This function is called to generate the initial allocation PFALLOC assigned to a stream. The PFALLOC must be of a definite length and so there are other functions like \PFLOPPY.TRUNCATE and \PFLOPPY.EXTEND which know how to split an incompletely used up PFALLOC or a PFALLOC that is about to overflow.

\PFLOPPY.ALLOCATE returns a PFALLOC pointing to a free block of storage that is at least LENGTH pages long. If LENGTH=NIL is supplied, then the length is defaulted to somewhere between DEFAULT.ALLOCATION and MINIMUM.ALLOCATION with preference towards DEFAULT.ALLOCATION.

The strategy is to first select the largest free block available. Not being able to find a free block causes \PFLOPPY.GAINSPACE to be called and then \PFLOPPY.ALLOCATE retries.

Having obtained the PFALLOC pointing to the largest free block available, \PFLOPPY.ALLOCATE determines if the free block is big enough. If the free block is not big enough then \PFLOPPY.GAINSPACE is called and \PFLOPPY.ALLOCATE retries.

Now being in possession of a PFALLOC pointing to a sufficiently large free block, the question is whether or not to split the PFALLOC. If the length of PFALLOC exceeds LENGTH by MINIMUM.ALLOCATION then PFALLOC is split into two free blocks one of which will have length LENGTH and will be returned. Otherwise, the PFALLOC is returned as is.

Splitting excessively large PFALLOCs is done in order that no stream hog the floppy and thereby be unfair to other streams. It may happen, for example in a freshly formatted floppy, that there is just one or very few but large free blocks. It would be unfair to hand out the only and very large free block to a stream and thereby prevent a second stream from opening.

On the other hand, splitting a PFALLOC whose length exceeds LENGTH by just a little bit is also undesirable because it leads to fragmentation and consumption of storage by the marker page overhead that must occur for each allocation. Thus, the length of PFALLOC not exceeding LENGTH by MINIMUM.ALLOCATION is considered to be close enough.

\PFLOPPY.ICHECK is called at the end of each \PFLOPPY.ALLOCATE to do an integrity check of the cached incore description of the floppy.

(\PFLOPPY.DEALLOCATE PFALLOC)

Deallocation is fairly easy. The two marker pages surrounding the allocation pointed to by PFALLOC are made to say that the type of the allocation between them is a free block.

\PFLOPPY.ICHECK is called at the end of each \PFLOPPY.DEALLOCATE to do an integrity check of the cached incore description of the floppy.

(\PFLOPPY.TRUNCATE PFALLOC LENGTH)

Truncation is also fairly easy. If PFALLOC is already of length LENGTH or smaller then nothing needs to be done.

Otherwise \PFLOPPY.TRUNCATE changes the local situation on

the floppy from one of

MP ALLOC MP

to one of

MP ALLOC MP FREE MP

by setting the length of PFALLOC to LENGTH, creating another PFALLOC to take up the slack and to be considered as a free block, and creating and updating and writing out three marker pages.

\PFLOPPY.ICHECK is called at the end of each \PFLOPPY.TRUNCATE to do an integrity check of the cached incore description of the floppy.

(\PFLOPPY.EXTEND PFALLOC)

This function gets called when an output stream is about to overflow its initial allocation. \PFLOPPY.EXTEND is charged with extending the length of PFALLOC's storage allocation.

If PFALLOC is followed by a free block, then it suffices to let PFALLOC cannibalize the following free block. The situation

MP ALLOC MP FREE MP

is changed to one of

MP ALLOC MP

by changing the length of PFALLOC to the sum of the length of PFALLOC's length, one page from the marker page between PFALLOC and the free block which is eliminated, and the length of the free block.

If PFALLOC is not followed by a free block, but instead is bumping up against valuable data, then a different strategy is used. \PFLOPPY.ALLOCATE is called to find a new and larger storage area NEW for the data stored at the allocation pointed to by PFALLOC. The data stored in the allocation pointed to by PFALLOC is then copied into the allocation pointed to by NEW. This is slightly time consuming, so the Lisp code prints the message "Reallocating" in the PROMPTWINDOW when this kind of activity is about to happen. After copying the data pointed to by PFALLOC into the area pointed to by NEW, NEW nad PFALLOC are made to exchange places. NEW is changed to point to where the data used to live and PFALLOC is changed to point to where the data has moved to. The marker pages around NEW and around PFALLOC have to be updated and written out to the floppy. The result, therefore, has the appearance of data crawling out of its restricted cavity and kicking out the free block living from the bigger cavity and then the free block moves over to where the data used to live.

\PFLOPPY.ICHECK is called at the end of each \PFLOPPY.EXTEND to do an integrity check of the cached incore description of the floppy.

(\PFLOPPY.GAINSPACE LENGTH)

This function is charged with going out and hunting up space on a floppy. \PFLOPPY.GAINSPACE returns after a free block of length

LENGTH has been made available.

The first thing to be done is to merge adjacent free blocks into larger free blocks. This is done by calling function \PFLOPPY.GAINSPACE.MERGE. \PFLOPPY.ICHECK is called at the end of each \PFLOPPY.GAINSPACE.MERGE to do an integrity check of the cached incore description of the floppy.

At this point \PFLOPPY.GAINSPACE checks to see if a free block of length LENGTH has been made available, and if so, returns.

Otherwise, \PFLOPPY.GAINSPACE calls FLOPPY.FREE.PAGES to determine whether a sufficiently large free block can be gained just by compacting the floppy. If the number of free pages available is greater than LENGTH, then compacting the floppy will have as one of its effects the collection of all free space into a single free block. Therefore, if the number of free pages available is greater than LENGTH, then \PFLOPPY.GAINSPACE calls FLOPPY.COMPACT to compact the floppy and then returns the single free block that is created by the compactor.

If calling FLOPPY.FREE.PAGES tells \PFLOPPY.GAINSPACE that there wouldn't be a large enough free block created just by compacting the floppy, then \PFLOPPY.GAINSPACE generates the usual contiuable FILE SYSTEM RESOURCES EXCEEDED error break. If the user responds by deleting some files and typing OK, then \PFLOPPY.GAINSPACE continues onward by going back to the top of its list of things to do and retrying its attempt to find a free block of length LENGTH.

(\PFLOPPY.FREE.PAGES)

Just sums up the lengths of all the PFALLOCs pointing to free blocks. This function gets called by the user function FLOPPY.FREE.PAGES.

(\PFLOPPY.ICHECK)

Does an integrity check on the Lisp implementation's incore cached directory information.

The Lisp floppy implementation does not blithely trust itself to always be doing the right thing, at least as far as the incore description of how the floppy is arranged. It is somewhat important that the in core description be absolutely legal and in agreement with reality out on the floppy, because incorrect cached directory structures may cause floppy operations to scramble the user's floppy confusing contents of files or real directory structure on the floppy.

\PFLOPPY.ICHECK gets called near the end of each major function in the Pilot floppy allocation code.

SYSOUT

Sysouting to floppy is kind of a hack. The sysout file that gets written has to be broken into smaller files that will fit on individual floppies. The small files are not ordinairy files but are Huge Pilot files. The HUGEPAGESTART and HUGEPAGELENGTH fields in the leader pages of these files become important.

\SFLOPPYFDEV (Variable)

The usual FLOPPY file device is \PFLOPPYFDEV. The sysout FLOPPY file device is \SFLOPPYFDEV. (FLOPPY.MODE 'SYSOUT) by the user makes the switch between the two file devices. The function SYSOUT seems to do (FLOPPY.MODE 'SYSOUT) automatically for the user, but in fact, floppy functions at a lower level detect SYSOUT on the stack of function calls by a certain amount of grungeyness and make the switch between file devices when necessary.

(\SFLOPPY.OPENHUGEFILE FILE ACCESS RECOG OTHERINFO FDEV OLDSTREAM)

Installed on \SFLOPPYFDEV and gets called when FILEIO opens a stream for input or output when FLOPPY is in SYSOUT mode. \SFLOPPY.OPENHUGEFILE returns a stream datatype. The stream can be either input or output.

If the stream is to be an input stream, \SFLOPPY.INPUTFLOPPY is called. If the stream is to be an output stream, \SFLOPPY.OUTPUTFLOPPY is called.

The stream returned is in every way like a Pilot stream returned by \PFLOPPY.OPENFILE with the exception that the DEVICE of the stream is \SFLOPPYFDEV. Two other fields on the stream, F1 and F2, point to the allocation record (PFALLOC) and leader page (PLPAGE) for the stream. The PFALLOC and PLPAGE can be conveniently accessed by using the FLOPPYSTREAM ACCESSFNS.

(\SFLOPPY.READPAGES STREAM FIRSTPAGE# BUFFERS)

Installed on \SFLOPPYFDEV and called by FILEIO when FLOPPY is in SYSOUT mode. Reads sectors off floppies into virtual memory pages BUFFERS. FIRSTPAGE# is in FILEIO's scheme of counting the pages of a file, beginning with 0 for the first page of a file. \SFLOPPY.READPAGES therefore fills BUFFERS with data read from floppies beginning with the FIRSTPAGE# of STREAM.

\SFLOPPY.READPAGES is implemented by calling \PFLOPPY.READPAGE. When \SFLOPPY.READPAGES is about to run off the end of a floppy, \SFLOPPY.CLOSEFLOPPY and \SFLOPPY.INPUTFLOPPY are called to bring in the next floppy.

(\SFLOPPY.WRITEPAGES STREAM FIRSTPAGE# BUFFERS)

Installed on \SFLOPPYFDEV and called by FILEIO when FLOPPY is in SYSOUT mode. Writes contents of virtual memory pages BUFFERS on to sectors of floppies. FIRSTPAGE# is in FILEIO's scheme of counting the pages of a file, beginning with 0 for the first page of a file. \SFLOPPY.WRITEPAGES therefore writes to the floppy beginning with the location corresponding to the FIRSTPAGE# of STREAM.

\SFLOPPY.WRITEPAGES is implemented by calling \PFLOPPY.WRITEPAGE. When \SFLOPPY.WRITEPAGES is about to run off the end of a floppy, \SFLOPPY.CLOSEFLOPPY and \SFLOPPY.OUTPUTFLOPPY are called to bring in the next floppy.

(\SFLOPPY.CLOSEHUGEFILE STREAM)

Installed on \SFLOPPYFDEV and called by FILEIO to close a sysout file when FLOPPY is in SYSOUT mode.

For output streams, does the usual \CLEARMAP plus a

\SFLOPPY.CLOSEFLOPPY on the last floppy being written.

There is some grungeyness to switch back to an old FLOPPY mode if the FLOPPY mode was reset by calling function SYSOUT.

(\SFLOPPY.INPUTFLOPPY FLOPPYNAME FILENAME OTHERINFO OLDSTREAM)

Called by \SFLOPPY.READPAGES when \SFLOPPY.READPAGES needs to go to the next floppy. \SFLOPPY.INPUTFLOPPY prompts the user to "Insert floppy" and does a (FLOPPY.WAIT.FOR.FLOPPY T) until the user does so.

Once the user has inputted the next floppy and FLOPPY.WAIT.FOR.FLOPPY has returned, \SFLOPPY.INPUTFLOPPY calls \PFLOPPY.OPENFILE to open a regular Pilot floppy stream to the piece of sysout file stored on the floppy inserted.

If no OLDSTREAM is supplied, then the Pilot floppy stream created by the call to \PFLOPPY.OPENFILE is returned. Otherwise, the PFALLOC and PLPAGE cached in the F1 and F2 fields of the Pilot floppy stream are pulled out and stuck into OLDSTREAM, after which OLDSTREAM is returned.

(\SFLOPPY.OUTPUTFLOPPY FLOPPYNAME FILENAME OTHERINFO OLDSTREAM)

Called by \SFLOPPY.WRITEPAGES when \SFLOPPY.WRITEPAGES needs to go to the next floppy. \SFLOPPY.OUTPUTFLOPPY prompts the user to "Insert floppy" and does a (FLOPPY.WAIT.FOR.FLOPPY T) until the user does so.

Once the user has inputted the next floppy and FLOPPY.WAIT.FOR.FLOPPY has returned, \SFLOPPY.OUTPUTFLOPPY tries to format the floppy. If there is any problem with formatting the floppy, such as the floppy being writeprotected, then the user is again asked to input a floppy.

Once the new floppy is formatted, \SFLOPPY.OUTPUTFLOPPY calls \PFLOPPY.OPENFILE to open a regular Pilot floppy stream which will be used to create a piece of the sysout file being stored on floppies.

If no OLDSTREAM is supplied, then the Pilot floppy stream created by the call to \PFLOPPY.OPENFILE is returned. Otherwise, the PFALLOC and PLPAGE cached in the F1 and F2 fields of the Pilot floppy stream are pulled out and stuck into OLDSTREAM, after which OLDSTREAM is returned.

(\SFLOPPY.CLOSEFLOPPY STREAM LASTFLOPPYFLG)

If the STREAM is an input stream, then \SFLOPPY.CLOSEFLOPPY just returns without doing anything.

If the STREAM is an output stream, then the leader pages and marker pages for the piece of sysout file stored on this floppy are written out. And as usual with Pilot streams, the updated filelist and PSECTOR9 also have to be written out.

HUGEPILOT

HUGEPILOT mode is implemented in a way similar to SYSOUT mode. The Huge Pilot file that gets written has to be broken into smaller files that will fit on individual floppies. The small files are not ordinairy files but are Huge Pilot files. The HUGEPAGESTART and HUGEPAGELENGTH fields in the leader pages of these files become important.

\HFLOPPYFDEV (Variable)

The usual FLOPPY file device is \PFLOPPYFDEV. The Huge Pilot FLOPPY file device is \HFLOPPYFDEV. (FLOPPY.MODE 'HUGEPILOT) by the user makes the switch between the two file devices.

(\HFLOPPY.OPENHUGEFILE FILE ACCESS RECOG OTHERINFO FDEV OLDSTREAM)

Installed on \HFLOPPYFDEV and gets called when FILEIO opens a stream for input or output when FLOPPY is in HUGEPILOT mode. \HFLOPPY.OPENHUGEFILE returns a stream datatype. The stream can be either input or output.

If the stream is to be an input stream, \HFLOPPY.INPUTFLOPPY is called. If the stream is to be an output stream, \HFLOPPY.OUTPUTFLOPPY is called.

The stream returned is in every way like a Pilot stream returned by \PFLOPPY.OPENFILE with the exception that the DEVICE of the stream is \HFLOPPYFDEV. Two other fields on the stream, F1 and F2, point to the allocation record (PFALLOC) and leader page (PLPAGE) for the stream. The PFALLOC and PLPAGE can be conveniently accessed by using the FLOPPYSTREAM ACCESSFNS.

(\HFLOPPY.READPAGES STREAM FIRSTPAGE# BUFFERS)

Installed on \HFLOPPYFDEV and called by FILEIO when FLOPPY is in HUGEPILOT mode. Reads sectors off floppies into virtual memory pages BUFFERS. FIRSTPAGE# is in FILEIO's scheme of counting the pages of a file, beginning with 0 for the first page of a file. \HFLOPPY.READPAGES therefore fills BUFFERS with data read from floppies beginning with the FIRSTPAGE# of STREAM.

\HFLOPPY.READPAGES is implemented by calling \PFLOPPY.READPAGE. When \HFLOPPY.READPAGES is about to run off the end of a floppy, \HFLOPPY.CLOSEFLOPPY and \HFLOPPY.INPUTFLOPPY are called to bring in the next floppy.

(\HFLOPPY.WRITEPAGES STREAM FIRSTPAGE# BUFFERS)

Installed on \HFLOPPYFDEV and called by FILEIO when FLOPPY is in HUGEPILOT mode. Writes contents of virtual memory pages BUFFERS on to sectors of floppies. FIRSTPAGE# is in FILEIO's scheme of counting the pages of a file, beginning with 0 for the first page of a file. \HFLOPPY.WRITEPAGES therefore writes to the floppy beginning with the location corresponding to the FIRSTPAGE# of STREAM.

\HFLOPPY.WRITEPAGES is implemented by calling \PFLOPPY.WRITEPAGE. When \HFLOPPY.WRITEPAGES is about to run off the end of a floppy, \HFLOPPY.CLOSEFLOPPY

and \HFLOPPY.OUTPUTFLOPPY are called to bring in the next floppy.

(\HFLOPPY.CLOSEHUGEFILE STREAM)

Installed on \HFLOPPYFDEV and called by FILEIO to close a Huge Pilot file when FLOPPY is in HUGEPILOT mode.

For output streams, does the usual \CLEARMAP plus a \HFLOPPY.CLOSEFLOPPY on the last floppy being written.

(\HFLOPPY.INPUTFLOPPY FLOPPYNAME FILENAME OTHERINFO OLDSTREAM)

Called by \HFLOPPY.READPAGES when \HFLOPPY.READPAGES needs to go to the next floppy. \HFLOPPY.INPUTFLOPPY prompts the user to "Insert floppy" and does a (FLOPPY.WAIT.FOR.FLOPPY T) until the user does so.

Once the user has inputted the next floppy and FLOPPY.WAIT.FOR.FLOPPY has returned, \HFLOPPY.INPUTFLOPPY calls \PFLOPPY.OPENFILE to open a regular Pilot floppy stream to the piece of Huge Pilot file stored on the floppy inserted.

If no OLDSTREAM is supplied, then the Pilot floppy stream created by the call to \PFLOPPY.OPENFILE is returned. Otherwise, the PFALLOC and PLPAGE cached in the F1 and F2 fields of the Pilot floppy stream are pulled out and stuck into OLDSTREAM, after which OLDSTREAM is returned.

(\HFLOPPY.OUTPUTFLOPPY FLOPPYNAME FILENAME OTHERINFO OLDSTREAM)

Called by \HFLOPPY.WRITEPAGES when \HFLOPPY.WRITEPAGES needs to go to the next floppy. \HFLOPPY.OUTPUTFLOPPY prompts the user to "Insert floppy" and does a (FLOPPY.WAIT.FOR.FLOPPY T) until the user does so.

Once the user has inputted the next floppy and FLOPPY.WAIT.FOR.FLOPPY has returned, \HFLOPPY.OUTPUTFLOPPY tries to format the floppy. If there is any problem with formatting the floppy, such as the floppy being writeprotected, then the user is again asked to input a floppy.

Once the new floppy is formatted, \HFLOPPY.OUTPUTFLOPPY calls \PFLOPPY.OPENFILE to open a regular Pilot floppy stream which will be used to create a piece of the Huge Pilot file being stored on floppies.

If no OLDSTREAM is supplied, then the Pilot floppy stream created by the call to \PFLOPPY.OPENFILE is returned. Otherwise, the PFALLOC and PLPAGE cached in the F1 and F2 fields of the Pilot floppy stream are pulled out and stuck into OLDSTREAM, after which OLDSTREAM is returned.

(\HFLOPPY.CLOSEFLOPPY STREAM LASTFLOPPYFLG)

If the STREAM is an input stream, then \HFLOPPY.CLOSEFLOPPY just returns without doing anything.

If the STREAM is an output stream, then the leader pages and marker pages for the piece of Huge Pilot file stored on this floppy

are written out. And as usual with Pilot streams, the updated filelist and PSECTOR9 also have to be written out.