File created: 11-May-2023 21:39:23 **{DSK}<cygdrive>c>Users>Larry>home>il>MEDLEY>SOURCES>LOCALFILE.;2**

edit by: lmm

changes to: (FNS \LFDeleteFile)

previous date: 19-Jan-93 10:55:28 {DSK}<cygdrive>c>Users>Larry>home>il>MEDLEY>SOURCES>LOCALFILE.;1

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS

```
(RPAQQ LOCALFILECOMS
       (
```

;;; This is the Dandelion/Dove local hard disk file system.

```
        (DECLARE%: EVAL@COMPILE DONTCOPY (FILES (SOURCE)
                                                DISKVMEMDECLS)
               (FILES MESATYPES)
               (LOCALVARS . T))
```

;;; Declare low-level data types on which all file system modules depend.

```
        (FNS \PFFetchString \PFReplaceString)
        (DECLARE%: EVAL@COMPILE DONTCOPY (COMS * PILOTFILECOMPILECOMS))
        (INITRECORDS PageGroup FileDescriptor)
```

;;; Define the various modules of the file system.

```
        (COMS * LFCOMS)
        (COMS * LFDIRECTORYCOMS)
        (COMS * SCAVENGEDSKDIRECTORYCOMS)
        (COMS * LFPILOTFILECOMS)
        (COMS * LFALLOCATIONMAPCOMS)
        (COMS * LFFILEMAPCOMS)
        (PROP MAKEFILE-ENVIRONMENT LOCALFILE)))
```

;;; This is the Dandelion/Dove local hard disk file system.

```
(DECLARE%: EVAL@COMPILE DONTCOPY

(FILESLOAD (SOURCE)
       DISKVMEMDECLS)

(FILESLOAD MESATYPES)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(LOCALVARS . T)
)
)
```

;;; Declare low-level data types on which all file system modules depend.

```
(DEFINEQ
```

(\PFFetchString

```
  [LAMBDA (startLoc lengthLoc maxLength)                    (* amd "10-Feb-86 18:25")
```

;;; Returns a string containing lengthLoc characters read starting from startLoc and capitalized.

```
    (PROG [(STR (ALLOCSTRING (MIN (\GETBASE lengthLoc 0)
                                  maxLength]
          [for POS from 1 to (NCHARS STR) do (RPLCHARCODE STR POS (\GETBASEBYTE startLoc (SUB1 POS]
          (RETURN STR])
```

(\PFReplaceString

```
  [LAMBDA (startLoc lengthLoc maxLength newString)          (* amd "10-Feb-86 18:26")
```

;;; Writes out newString beginning at startLoc, and indicates the length in the word beginning at lengthLoc.

```
    (SETQ newString (MKSTRING newString))
    (PROG ((LENGTH (MIN (NCHARS newString)
                        maxLength)))
      ;; First write out characters
          (for POS from 0 to (SUB1 LENGTH) as CHAR instring newString do (\PUTBASEBYTE startLoc POS CHAR))
      ;; Then write out length of string
```

```
          (\PUTBASE lengthLoc 0 LENGTH)
          (RETURN newString])

)

(DECLARE%: EVAL@COMPILE DONTCOPY

(RPAQQ PILOTFILECOMPILECOMS
       (;; Assorted system constants
        (CONSTANTS (pilotVersion 8))
        (CONSTANTS (maxPagesPerFile 8388607)
               (lastPageNumber (SUB1 maxPagesPerFile))
               (nullVolumePage 0)
               (maxLogicalVolumes 10))
        (CONSTANTS (hardMicrocode 0)
               (bftGerm 2))
        ;; Interesting Pilot file types.
        (CONSTANTS (tUnassigned 0)
               (tPhysicalVolumeRootPage 1)
               (tSubVolumeMarkerPage 4)
               (tLogicalVolumeRootPage 5)
               (tFreePage 6)
               (tVolumeAllocationMap 7)
               (tVolumeFileMap 8)
               (tRootDirectory 18)
               (tLispDirectory 10048)
               (tLispFile 10049)
               (tDiagnosticMicrocode 65535)
               (pilotVolume 0)
               (nonPilotVolume 3))
        ;; Logical volume root page, physical volume root page, and marker page types
        (CONSTANTS (logicalVolumeSeal 45771))
        (RECORDS Page RandomPage FileID VolumeID DiskFileID LVBootFiles RootFileArray LogicalVolumeDescriptor)
        (CONSTANTS (physicalVolumeSeal 41610))
        (RECORDS PVBootFiles SubVolumeDesc SubVolumeArray PhysicalVolumeDescriptor)
        (RECORDS LogicalSubVolumeMarker SubVolumeMarkerPage)
        (MACROS LVEqual SwapIn&Dirty LvBasePageAddr MarkerPageAddr)
        ;; Volume root directory stuff
        (CONSTANTS (rootDirSeal 30167)
               (rootDirVersion 2)
               (rootDirMaxEntries 84))
        (RECORDS RootDirEntry RootDirEntryArray RootDirectory)
        ;; Miscellaneous records
        (RECORDS PageGroup FileDescriptor)
        (RECORDS Label)
        ;; The following are for diagnostic purposes.
        (MACROS DISPLAYWORDS DISPLAYLABEL DISPLAYPAGE)))

;; Assorted system constants

(DECLARE%: EVAL@COMPILE

(RPAQQ pilotVersion 8)

(CONSTANTS (pilotVersion 8))
)

(DECLARE%: EVAL@COMPILE

(RPAQQ maxPagesPerFile 8388607)

(RPAQ lastPageNumber (SUB1 maxPagesPerFile))

(RPAQQ nullVolumePage 0)

(RPAQQ maxLogicalVolumes 10)

(CONSTANTS (maxPagesPerFile 8388607)
       (lastPageNumber (SUB1 maxPagesPerFile))
       (nullVolumePage 0)
       (maxLogicalVolumes 10))
)

(DECLARE%: EVAL@COMPILE

(RPAQQ hardMicrocode 0)

(RPAQQ bftGerm 2)

(CONSTANTS (hardMicrocode 0)
       (bftGerm 2))
```

```
)
```

;; Interesting Pilot file types.

```
(DECLARE%: EVAL@COMPILE

(RPAQQ tUnassigned 0)

(RPAQQ tPhysicalVolumeRootPage 1)

(RPAQQ tSubVolumeMarkerPage 4)

(RPAQQ tLogicalVolumeRootPage 5)

(RPAQQ tFreePage 6)

(RPAQQ tVolumeAllocationMap 7)

(RPAQQ tVolumeFileMap 8)

(RPAQQ tRootDirectory 18)

(RPAQQ tLispDirectory 10048)

(RPAQQ tLispFile 10049)

(RPAQQ tDiagnosticMicrocode 65535)

(RPAQQ pilotVolume 0)

(RPAQQ nonPilotVolume 3)

(CONSTANTS (tUnassigned 0)
       (tPhysicalVolumeRootPage 1)
       (tSubVolumeMarkerPage 4)
       (tLogicalVolumeRootPage 5)
       (tFreePage 6)
       (tVolumeAllocationMap 7)
       (tVolumeFileMap 8)
       (tRootDirectory 18)
       (tLispDirectory 10048)
       (tLispFile 10049)
       (tDiagnosticMicrocode 65535)
       (pilotVolume 0)
       (nonPilotVolume 3))
)
```

;; Logical volume root page, physical volume root page, and marker page types

```
(DECLARE%: EVAL@COMPILE

(RPAQQ logicalVolumeSeal 45771)

(CONSTANTS (logicalVolumeSeal 45771))
)

(DECLARE%: EVAL@COMPILE

[RECORD Page NIL (CREATE (NCREATE 'VMEMPAGEP))
       (TYPE? (TYPENAMEP DATUM 'VMEMPAGEP]

(RECORD RandomPage NIL (TYPE? (EQ (fetch (POINTER WORDINPAGE) of DATUM)
                                  0)))

(MESATYPE FileID (2 WORD))

(MESATYPE VolumeID (5 WORD))

(MESARECORD DiskFileID ((fID VolumeID)
                        (firstPage SWAPPEDFIXP)
                        (da SWAPPEDFIXP))                    (* Booting information)
       )

(MESAARRAY LVBootFiles ((0 5))
       DiskFileID                                            (* Booting information)
       )

(MESAARRAY RootFileArray ((6 14))
       FileID)

[MESARECORD LogicalVolumeDescriptor ((seal WORD)              (* Validation ; absolutely must be first field)
                                     (version WORD)           (* must be 2nd field)
                                     (vID VolumeID)           (* ID of This Volume)
                                     (labelLength WORD)       (* Length of th ASCII name of this volume)
                                     (label 40 BYTE)          (* Volume name in AScII)
                                     (type WORD)
```

```
                                    (volumeSize SWAPPEDFIXP)    (* Number of pages in this volume)
                                    (bootingInfo LVBootFiles)   (* Defines 6 PILOT file types)
                                    (NIL WORD)
                                    (NIL BITS 15)
                                    (changing FLAG)             (* Change field decls from here on only)
                                                                (* boolean _ T)
                                    (freePageCount SWAPPEDFIXP) (* Number of free pages remaining)
                                    (vamStart SWAPPEDFIXP)
                                    (vfmStart SWAPPEDFIXP)      (* Relative address of the start of the volume file map)
                                    (lowerBound SWAPPEDFIXP)
                                    (volumeRootDirectory SWAPPEDFIXP)
                                    (rootFileID RootFileArray)
                                    (lastIDAllocated SWAPPEDFIXP)

        (* Highest numbered File.ID given out on this volume. We reserve the first set of IDs for Pilot's own use.
        In particular, files of type IN PilotRootFileType may have their ID the same as their File.Type.)

                                    (scavengerLogVolume VolumeID)
                                    (lastTimeOpendForWrite SWAPPEDFIXP)
                                    (NIL 131 WORD)
                                    (checksum WORD)             (* Must be the last field)
                                    )
       (ACCESSFNS (LVlabel (\PFFetchString (LOCF (fetch (LogicalVolumeDescriptor label) of DATUM))
                                   (LOCF (fetch (LogicalVolumeDescriptor labelLength) of DATUM))
                                   40)
                           (\PFReplaceString (LOCF (fetch (LogicalVolumeDescriptor label) of DATUM))
                                  (LOCF (fetch (LogicalVolumeDescriptor labelLength) of DATUM))
                                  40 NEWVALUE)))
       (CREATE (PROG ((lv (create Page)))
                     (replace (LogicalVolumeDescriptor seal) of lv with logicalVolumeSeal)
                     (RETURN lv)))
       (TYPE? (AND (type? Page DATUM)
                   (EQ (fetch (LogicalVolumeDescriptor seal) of DATUM)
                       logicalVolumeSeal]
)

(DECLARE%: EVAL@COMPILE

(RPAQQ physicalVolumeSeal 41610)

(CONSTANTS (physicalVolumeSeal 41610))
)

(DECLARE%: EVAL@COMPILE

(MESAARRAY PVBootFiles ((0 3))
       DiskFileID)

(MESARECORD SubVolumeDesc ((lvID VolumeID)
                           (lvSize SWAPPEDFIXP)
                           (lvPage SWAPPEDFIXP)
                           (pvPage SWAPPEDFIXP)
                           (nPages SWAPPEDFIXP)))

(MESAARRAY SubVolumeArray ((0 9))
       SubVolumeDesc)

[MESARECORD PhysicalVolumeDescriptor ((seal WORD)                (* Validation)
                                      (version WORD)
                                      (labelLength WORD)
                                      (pvID VolumeID)
                                      (bootingInfo PVBootFiles)  (* Defines 4 PILOT file types)
                                      (label 40 BYTE)            (* Ascii name of the volume)
                                      (subVolumeCount WORD)
                                      (subVolumeMarkerID VolumeID)
                                                                 (* Marker pages belong to this Pseudo File)
                                      (badPageCount SWAPPEDFIXP)
                                      (maxBadPages SWAPPEDFIXP)
                                      (onLineCount WORD)
                                      (subVolumes SubVolumeArray)
                                                                 (* See SubVolumeDesc record for description of each of six
                                                                 entries stored here)
                                      (NIL 47 WORD)
                                      (localTimeParametersValid WORD)
                                      (localTimeParameters 2 WORD)
                                      (checksum WORD))
       (ACCESSFNS (PVlabel (\PFFetchString (LOCF (fetch (PhysicalVolumeDescriptor label) of DATUM))
                                   (LOCF (fetch (PhysicalVolumeDescriptor labelLength) of DATUM))
                                   40)
                           (\PFReplaceString (LOCF (fetch (PhysicalVolumeDescriptor label) of DATUM))
                                  (LOCF (fetch (PhysicalVolumeDescriptor labelLength) of DATUM))
                                  40 NEWVALUE)))
       (CREATE (PROG ((physicalVol (create Page)))
                     (replace (PhysicalVolumeDescriptor seal) of physicalVol with physicalVolumeSeal)
                     (RETURN physicalVol)))
       (TYPE? (AND (type? Page DATUM)
                   (EQ (fetch (PhysicalVolumeDescriptor seal) of DATUM)
```

```
                        physicalVolumeSeal]
)

(DECLARE%: EVAL@COMPILE

(MESARECORD LogicalSubVolumeMarker ((seal WORD)
                                    (version WORD)
                                    (labelLength BITS 6)
                                    (type BITS 2)
                                    (NIL BITS 8)
                                    (label 20 WORD)
                                    (bootingInfo LVBootFiles)
                                    (volumeRootDirectory SWAPPEDFIXP)))

(MESARECORD SubVolumeMarkerPage ((logical LogicalSubVolumeMarker)
                                                                  (* Incomplete)
                                 )
       (CREATE (create Page))
       (TYPE? (type? Page DATUM)))
)

(DECLARE%: EVAL@COMPILE

(PUTPROPS LVEqual MACRO ((a b)
                         (MESAEQUAL (fetch (LogicalVolumeDescriptor vID) of a)
                                (fetch (LogicalVolumeDescriptor vID) of b)
                                VolumeID)))

(PUTPROPS SwapIn&Dirty MACRO (OPENLAMBDA (page)
                                (\PUTBASE page 0 (\GETBASE page 0))))

(PUTPROPS LvBasePageAddr MACRO ((vol)
                                (fetch (SubVolumeDesc pvPage) of (FMESAELT (fetch (PhysicalVolumeDescriptor
                                                                                    subVolumes)
                                                                                  of \PhysVolumePage)
                                                                           SubVolumeArray vol))))

(PUTPROPS MarkerPageAddr MACRO [(vol)
                                (fetch (SubVolumeDesc nPages) of (FMESAELT (fetch (PhysicalVolumeDescriptor
                                                                                    subVolumes)
                                                                                  of \PhysVolumePage)
                                                                           SubVolumeArray
                                                                           (OR (FIXP vol)
                                                                               (\PFVolumeNumber vol])
)
```

;; Volume root directory stuff

```
(DECLARE%: EVAL@COMPILE

(RPAQQ rootDirSeal 30167)

(RPAQQ rootDirVersion 2)

(RPAQQ rootDirMaxEntries 84)

(CONSTANTS (rootDirSeal 30167)
       (rootDirVersion 2)
       (rootDirMaxEntries 84))
)

(DECLARE%: EVAL@COMPILE

(MESARECORD RootDirEntry ((type WORD)
                          (file SWAPPEDFIXP)))

(MESAARRAY RootDirEntryArray ((0 rootDirMaxEntries))
       RootDirEntry)

[MESARECORD RootDirectory ((seal WORD)
                           (version WORD)
                           (maxEntries WORD)
                           (countEntries WORD)
                           (entries RootDirEntryArray))
       (CREATE (PROG ((rootDir (create Page)))
                     (replace (RootDirectory seal) of rootDir with rootDirSeal)
                     (replace (RootDirectory version) of rootDir with rootDirVersion)
                     (replace (RootDirectory maxEntries) of rootDir with rootDirMaxEntries)
                     (RETURN rootDir)))
       (TYPE? (AND (type? Page DATUM)
                   (EQ (fetch (RootDirectory seal) of DATUM)
                       rootDirSeal]
)
```

;; Miscellaneous records

```
(DECLARE%: EVAL@COMPILE

(DATATYPE PageGroup ((filePage SWAPPEDFIXP)
                     (volumePage SWAPPEDFIXP)
                     (nextFilePage SWAPPEDFIXP)))

(DATATYPE FileDescriptor (fileID                                ; Can be either a FIXP or a pointer to a VolumeID
                                (volNum FIXP)                   ; 0..9
                                (type WORD)                     ; Pilot file type
                                (size FIXP)                     ; Current number of (Pilot) pages allocated to this file
                                (PAGEGROUP POINTER)             ; Caches the last PageGroup found for this file

                                ))
)

(/DECLAREDATATYPE 'PageGroup '(SWAPPEDFIXP SWAPPEDFIXP SWAPPEDFIXP)
       ;; ---field descriptor list elided by lister---
       '6)

(/DECLAREDATATYPE 'FileDescriptor '(POINTER FIXP WORD FIXP POINTER)
       ;; ---field descriptor list elided by lister---
       '10)

(DECLARE%: EVAL@COMPILE

[MESARECORD Label ((fileID SWAPPEDFIXP)                         (* valid in label of every page)
                   (NIL 3 WORD)
                   (filePageLo WORD)
                   (filePageHi BITS 7)                          (* 23 bit page number, valid in label of every page)
                                                                (* always zero)
                   (pageZeroAttributes BITS 9)                  (* valid only in label of page 0)
                   (attributesInAllPages WORD)                  (* valid in label of every page)
                   (dontCare 2 WORD))
       (ACCESSFNS (filePage (\MAKENUMBER (fetch (Label filePageHi) of DATUM)
                                         (fetch (Label filePageLo) of DATUM))
                            (PROGN (replace (Label filePageHi) of DATUM with (\HINUM NEWVALUE))
                                   (replace (Label filePageLo) of DATUM with (\LONUM NEWVALUE))
                                   NEWVALUE)))
       (TYPE? (OR (type? ARRAYBLOCK DATUM)
                  (AND (GETD '\BLOCKDATAP)
                       (\BLOCKDATAP DATUM]
)

;; The following are for diagnostic purposes.

(DECLARE%: EVAL@COMPILE

(PUTPROPS DISPLAYWORDS MACRO [LAMBDA (Start Number)
                                  ;; Prints out the first Number words of the object Start
                                  [for I from 0 to (SUB1 Number) do (PRIN1 (\GETBASE Start I))
                                                                    (PRIN1 " ")
                                                                    (COND
                                                                       ((EQ (IREMAINDER (ADD1 I)
                                                                                        14)
                                                                            0)
                                                                        (TERPRI]
                                  (TERPRI])

(PUTPROPS DISPLAYLABEL MACRO [LAMBDA (vol volumePageNumber)
                                  ;; Prints the label of the given page.
                                  (PROG ((L (create Label)))
                                        (if (type? LogicalVolumeDescriptor vol)
                                            then (SETQ vol (\PFVolumeNumber vol)))
                                        (\PFTransferPage (IPLUS (LvBasePageAddr vol)
                                                                volumePageNumber)
                                                         (create Page)
                                                         'VRR L)
                                        (DISPLAYWORDS L 10])

(PUTPROPS DISPLAYPAGE MACRO [LAMBDA (vol volumePageNumber)
                                  ;; Prints out the specified page of the disk.
                                  (PROG ((P (create Page)))
                                        (if (type? LogicalVolumeDescriptor vol)
                                            then (SETQ vol (\PFVolumeNumber vol)))
                                        (\PFTransferPage (IPLUS (LvBasePageAddr vol)
                                                                volumePageNumber)
                                                         P
                                                         'VRR
                                                         (create Label))
                                        (DISPLAYWORDS P WORDSPERPAGE])
)
)
```

```
(/DECLAREDATATYPE 'PageGroup '(SWAPPEDFIXP SWAPPEDFIXP SWAPPEDFIXP)
        ;; ---field descriptor list elided by lister---
        '6)

(/DECLAREDATATYPE 'FileDescriptor '(POINTER FIXP WORD FIXP POINTER)
        ;; ---field descriptor list elided by lister---
        '10)


;;; Define the various modules of the file system.

(RPAQQ LFCOMS
       (

;;; This module handles the interface to the device-independent part of the file system: it provides a vector of standard device-specific file system
;;; operations.  This used to be the sole contents of the file LOCALFILE.

         (DECLARE%: EVAL@COMPILE DONTCOPY
                ;; File system datatypes
                (CONSTANTS (lispFileVersion 2)
                       (leaderPageSeal 54321))
                (RECORDS LFDEV DLIONSTREAM LeaderPage)
                ;; Error mechanism
                (MACROS DiskError))
         ;; Public entry
         (FNS CREATEDSKDIRECTORY PURGEDSKDIRECTORY LISPDIRECTORYP VOLUMES VOLUMESIZE)
         (FNS \DFSCurrentVolume \DFSFreeDiskPages)
         (FNS \LFEntryPoint \LFNormalizeVolumeName)
         ;; Device management
         (FNS \LFCreateDevice \LFOpenDevice \LFCloseDevice)
         (GLOBALVARS \LFdevice \LFtopMonitor \LFrunSize)
         (P (\LFCreateDevice))
         (INITVARS (\LFtopMonitor (CREATE.MONITORLOCK 'topMonitor))
                (\LFrunSize 20))
         ;; Device methods
         (FNS \LFOpenFile \LFGetStreamForFile \LFOpenOldFile \LFGenFileID \LFCreateFile \LFMakeLeaderPage
              \LFUpdateLeaderPage \LFWriteLeaderPage)
         (FNS \LFCloseFile)
         (FNS \LFDeleteFile)
         (FNS \LFReadPages)
         (FNS \LFWritePages \LFExtendFileIfNecessary \LFExtendFile)
         (FNS \LFGetFileInfo \LFSetFileInfo)
         (FNS \LFGetFileName)
         (FNS \LFEventFn)
         (FNS \LFDirectoryNameP)
         (FNS \LFTruncateFile)
         (FNS \LFRenameFile)))


;;; This module handles the interface to the device-independent part of the file system: it provides a vector of standard device-specific file system
;;; operations.  This used to be the sole contents of the file LOCALFILE.

(DECLARE%: EVAL@COMPILE DONTCOPY

(DECLARE%: EVAL@COMPILE

(RPAQQ lispFileVersion 2)

(RPAQQ leaderPageSeal 54321)

(CONSTANTS (lispFileVersion 2)
       (leaderPageSeal 54321))
)

(DECLARE%: EVAL@COMPILE

[RECORD LFDEV FDEV (SUBRECORD FDEV)
       (TYPE? (AND (type? FDEV DATUM)
                   (EQ (fetch (FDEV CLOSEFILE) of DATUM)
                       (FUNCTION \LFCloseFile))
                   (EQ (fetch (FDEV HOSTNAMEP) of DATUM)
                       (FUNCTION NILL]

[RECORD DLIONSTREAM STREAM (SUBRECORD STREAM)
       [ACCESSFNS ((FILEDESC (fetch F1 of DATUM)
                             (replace F1 of DATUM with NEWVALUE))
                   (LEADERPAGE (fetch F2 of DATUM)
                             (replace F2 of DATUM with NEWVALUE))
                   (DIRINFO (fetch F4 of DATUM)
```

```
                        (replace F4 of DATUM with NEWVALUE))
                    (DIRHOLEPTR (fetch F5 of DATUM)
                        (replace F5 of DATUM with NEWVALUE))
                    (VOLUME (\PFGetVol (fetch (FileDescriptor volNum) of (fetch (DLIONSTREAM FILEDESC) of DATUM]
        (TYPE? (AND (type? STREAM DATUM)
                    (type? FileDescriptor (fetch (DLIONSTREAM FILEDESC) of DATUM]

[MESARECORD LeaderPage ((seal WORD)
                        (version WORD)
                        (TimeCreate FIXP)
                        (TimeWrite FIXP)
                        (TimeRead FIXP)
                        (FileID FIXP)
                        (AllocatedPages FIXP)
                        (EofPage FIXP)
                        (EOffSet WORD)
                        (NameLength WORD)
                        (FileName 256 BYTE)
                        (AuthorLength WORD)
                        (AuthorName 64 BYTE)
                        (typeHolder WORD))
        (ACCESSFNS (TYPE (SELECTQ (fetch (LeaderPage typeHolder) of DATUM)
                              (0 'TEXT)
                             'BINARY)
                         (PROGN (replace (LeaderPage typeHolder) of DATUM with (SELECTQ NEWVALUE
                                                                                        (TEXT 0)
                                                                                        1))
                                NEWVALUE)))
        (ACCESSFNS (fileName (\PFFetchString (LOCF (fetch (LeaderPage FileName) of DATUM))
                                             (LOCF (fetch (LeaderPage NameLength) of DATUM))
                                             256)
                             (\PFReplaceString (LOCF (fetch (LeaderPage FileName) of DATUM))
                                               (LOCF (fetch (LeaderPage NameLength) of DATUM))
                                               256 NEWVALUE)))
        (ACCESSFNS (author (\PFFetchString (LOCF (fetch (LeaderPage AuthorName) of DATUM))
                                           (LOCF (fetch (LeaderPage AuthorLength) of DATUM))
                                           64)
                           (\PFReplaceString (LOCF (fetch (LeaderPage AuthorName) of DATUM))
                                             (LOCF (fetch (LeaderPage AuthorLength) of DATUM))
                                             64 NEWVALUE)))
        (CREATE (PROG ((leader (create Page)))
                      (replace (LeaderPage seal) of leader with leaderPageSeal)
                      (RETURN leader)))
        (TYPE? (AND (type? Page DATUM)
                    (EQ (fetch (LeaderPage seal) of DATUM)
                        leaderPageSeal]
)

(DECLARE%: EVAL@COMPILE

(PUTPROPS DiskError MACRO ((errorType fileName CONTINUEOKFLG)
                           (PROG ((\INTERRUPTABLE T))

          (* * Gross hack to allow the error to show up as a break rather than a 9318)

                                 (LISPERROR errorType fileName CONTINUEOKFLG))))
)
)

;; Public entry

(DEFINEQ

(CREATEDSKDIRECTORY
  [LAMBDA (volName smashDirectory)                                    ; Edited 8-Jan-87 17:50 by amd
    ;; Creates a directory on the specified volume, if possible.  If this constitutes the first Lisp directory on the disk, creates the local disk device to run
    ;; this directory (and any subsequent ones).  If smashDirectory, it will smash any old Lisp directory on the volume.
    (WITH.MONITOR \LFtopMonitor
        (PROG ((vol (\LFEntryPoint volName NIL T))
               markerPage)
              (if (NOT (\PFPilotVolumeP vol))
                  then (ERROR "Non-pilot volume"))
              (if smashDirectory
                  then (\LFPurgeDirectory vol))
              (if (\LFDirectoryP vol)
                  then (ERROR "Directory already created"))
              (UNINTERRUPTABLY
                  (if [NOT (type? LFDEV (\GETDEVICEFROMNAME 'DSK]
                      then (\LFCreateDevice))
                  (if (type? LFDEV (\GETDEVICEFROMNAME 'DSK))
                      then (\LFMakeVolumeDirectory vol)
                    else (\LFMakeVolumeDirectory vol T)
                         (\LFOpenDevice)))
              (\PFDsplyVolumes))
        (PACKFILENAME.STRING 'HOST 'DSK 'DIRECTORY (U-CASE volName)))])
```

(PURGEDSKDIRECTORY

```
  [LAMBDA (volName dontDeleteFiles)                                    (* hdj " 5-Jun-86 12:54")
```

;;; Purges the Lisp directory on the specified volume. If this is the last valid Lisp directory on the disk, shuts down the local disk device.

```
    (WITH.MONITOR \LFtopMonitor
        [PROG ((vol (\LFEntryPoint volName NIL T))
               (diskDevice (\GETDEVICEFROMHOSTNAME 'DSK))
               device)
              (if (NOT (\PFPilotVolumeP vol))
                  then (ERROR "Non-pilot volume"))
              (UNINTERRUPTABLY
                  ;; Close all files open on that directory
                  (for S in (\DEVICE-OPEN-STREAMS diskDevice) when (AND (type? DLIONSTREAM S)
                                                                        (EQ (fetch (DLIONSTREAM VOLUME)
                                                                                   of S)
                                                                            vol))
                     do (printout PROMPTWINDOW T "Closing " (CLOSEF S)))
                  ;; Delete all files on that directory.
                  [if (NOT dontDeleteFiles)
                      then (for F in (FILDIR (PACKFILENAME 'HOST 'DSK 'DIRECTORY (fetch (LogicalVolumeDescriptor
                                                                                         LVlabel)
                                                                                     of vol)))
                                do (printout PROMPTWINDOW T "Deleting " (DELFILE F]
                  ;; Remove the directory
                  (\LFPurgeDirectory vol)
                  ;; If this was the last Lisp directory, replace the dandelion disk diskDevice with a coredevice. Actually, all you need to do is kill
                  ;; the dlion disk diskDevice and VANILLADISK will take care of the rest
                  (OR (\LFFindDirectoryVol)
                      (\LFCloseDevice)))])])
```

(LISPDIRECTORYP

```
  [LAMBDA (volumeName)                                                 (* amd "10-Feb-86 16:04")
```

;;; Returns T if volumeName has a valid Lisp directory on it, NIL otherwise.

```
    (WITH.MONITOR \LFtopMonitor
        (SELECTQ (MACHINETYPE)
            ((DANDELION DOVE)
                [PROG ((vol (\LFEntryPoint volumeName NIL T)))
                      (RETURN (NOT (NOT (AND vol (\LFDirectoryP vol])
            NIL))])
```

(VOLUMES

```
  [LAMBDA NIL                                                          (* amd "10-Feb-86 16:04")
```

;;; Returns a list of the names of the logical volumes on this machine.

```
    (SELECTQ (MACHINETYPE)
        ((DANDELION DOVE)
             (\LFEntryPoint NIL T)
             [for vol in (\PFGetVols) collect (MKATOM (U-CASE (fetch (LogicalVolumeDescriptor LVlabel) of vol])
        NIL])
```

(VOLUMESIZE

```
  [LAMBDA (volName recompute)                                          (* amd "10-Feb-86 16:04")
```

;;; Returns the size of the specified volume.

```
    (PROG ((vol (\LFEntryPoint volName)))
          (RETURN (fetch (LogicalVolumeDescriptor volumeSize) of vol])

)

(DEFINEQ
```

(\DFSCurrentVolume

```
  [LAMBDA NIL                                                          (* hts%: "13-Feb-85 22:47")
```

;;; Returns as an atom the name of the volume which contains the currently running virtual memory. Called by DISKPARTITION.

```
    (\LFEntryPoint NIL T)
    (MKATOM (U-CASE (fetch (LogicalVolumeDescriptor LVlabel) of (\PFCurrentVol])
```

(\DFSFreeDiskPages

```
  [LAMBDA (volName recompute)                                          (* amd "10-Feb-86 16:04")
```

;;; Returns the number of free pages left on the specified volume. Called by DISKFREEPAGES.

```
    (WITH.MONITOR \LFtopMonitor
        (PROG ((vol (\LFEntryPoint volName)))
              (RETURN (\PFFreeDiskPages vol recompute)))))])

)

(DEFINEQ
```

(\LFEntryPoint

```
  [LAMBDA (volName noVolName dontDefault)                    ; Edited  8-Jan-87 17:49 by amd
```

;; Run at every entry point to the file system. Makes sure everything is set up ok, and makes all entry points share some common code.

```
    (OR (ATOM volName)
        (STRINGP volName)
        (\ILLEGAL.ARG volName))
    (SELECTQ (MACHINETYPE)
         ((DANDELION DOVE)
              NIL)
         (ERROR "Wrong machinetype"))
    (\PFEnsureInitialized)
    (if (NOT (\PFVersionOK))
        then (ERROR "Wrong Pilot version on disk"))
    (if (NOT noVolName)
        then (PROG [(vol (OR (\PFGetLVPage (\LFNormalizeVolumeName volName))
                             (AND (NOT volName)
                                  (NOT dontDefault)
                                  (\LFFindDirectoryVol NIL]
                   (if (NULL vol)
                       then (ERROR "Volume not on local disk"))
                   (RETURN vol])
```

(\LFNormalizeVolumeName

```
  [LAMBDA (volName)                                          (* amd "10-Feb-86 18:14")
```

;;; If the volume name given is a valid one, returns that; else assumes it is a full file name of some sort, and extracts the volume name from it.

```
    (if (STRPOS "{" volName)
        then (fetch (PARSEDFILENAME VOL) of (\LFParseFileName volName))
      else volName])

)
```

;; Device management

```
(DEFINEQ
```

(\LFCreateDevice

```
  [LAMBDA NIL                                                (* hdj "25-Sep-86 13:22")
```

;;; Creates and remembers the local hard disk file device, but does not open the device or any of its associated directories.

```
    (if (AND (BOUNDP '\LFdevice)
             (type? LFDEV \LFdevice))
        then \LFdevice
      else (SETQ \LFdevice (\MAKE.PMAP.DEVICE (create FDEV
                                                      NODIRECTORIES _ T
                                                      DEVICENAME _ 'DSK
                                                      CLOSEFILE _ (FUNCTION \LFCloseFile)
                                                      DELETEFILE _ (FUNCTION \LFDeleteFile)
                                                      RENAMEFILE _ (FUNCTION \LFRenameFile)
                                                      TRUNCATEFILE _ (FUNCTION \LFTruncateFile)
                                                      GETFILEINFO _ (FUNCTION \LFGetFileInfo)
                                                      GETFILENAME _ (FUNCTION \LFGetFileName)
                                                      OPENFILE _ (FUNCTION \LFOpenFile)
                                                      READPAGES _ (FUNCTION \LFReadPages)
                                                      SETFILEINFO _ (FUNCTION \LFSetFileInfo)
                                                      WRITEPAGES _ (FUNCTION \LFWritePages)
                                                      REOPENFILE _ (FUNCTION \LFOpenFile)
                                                      GENERATEFILES _ (FUNCTION \LFGenerateFiles)
                                                      EVENTFN _ (FUNCTION \LFEventFn)
                                                      DIRECTORYNAMEP _ (FUNCTION \LFDirectoryNameP)
                                                      HOSTNAMEP _ (FUNCTION NILL)
                                                      OPENP _ (FUNCTION \GENERIC.OPENP)
                                                      REGISTERFILE _ (FUNCTION \ADD-OPEN-STREAM)
                                                      UNREGISTERFILE _ (FUNCTION \GENERIC-UNREGISTER-STREAM])
```

(\LFOpenDevice

```
  [LAMBDA NIL                                                (* amd "10-Feb-86 18:03")
```

;;; Opens the local hard disk file system device and returns it if it can be opened; otherwise returns NIL. Device can be opened iff Pilot version is OK
;;; and there is at least one valid Lisp directory of the appropriate version on the disk.

```
    (WITH.MONITOR \LFtopMonitor
        (SELECTQ (MACHINETYPE)
```

```
            ((DANDELION DOVE)
                (\PFEnsureInitialized)
                (AND (\PFVersionOK)
                     (for VOL in (\PFGetVols) thereis (\LFCloseDirectory VOL)
                                                      (AND (\LFDirectoryP VOL)))
                     (\GETDEVICEFROMNAME (\DEFINEDEVICE 'DSK \LFdevice))))
            NIL))])
```

(\LFCloseDevice

```
  [LAMBDA NIL                                                    (* amd "10-Feb-86 18:04")
```

(* * comment)

```
    (WITH.MONITOR \LFtopMonitor
        (\PFEnsureInitialized T)
        (\REMOVEDEVICE \LFdevice)
        (AND (\PFVersionOK)
             (for VOL in (\PFGetVols) do (\LFCloseDirectory VOL)))
        NIL)])

)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \LFdevice \LFtopMonitor \LFrunSize)
)
```

(**LFCreateDevice**)

```
(RPAQ? \LFtopMonitor (CREATE.MONITORLOCK 'topMonitor))

(RPAQ? \LFrunSize 20)
```

;; Device methods

```
(DEFINEQ
```

(\LFOpenFile

```
  [LAMBDA (FILE ACCESS RECOG OTHERINFO FDEV OLDSTREAM)           ; Edited 21-Aug-88 14:17 by bvm
```

;; Open a file.

```
    (LET [(STREAM (WITH.MONITOR \LFtopMonitor
                        (PROG (DATE STREAM IDATE)
```

;; Normalize creationdate. User can supply a bad creationdate. If normalization is done at a lower level in
;; uninterruptable code, and if IDATE signals an error, the result will be a 9318 crash rather than an error break.

```
                              [if (SETQ DATE (OR (FASSOC 'CREATIONDATE OTHERINFO)
                                                 (FASSOC 'ICREATIONDATE OTHERINFO)))
                                  then (SETQ OTHERINFO (CONS [CONS 'CREATIONDATE
                                                                   (OR [SETQ IDATE
                                                                             (if (EQ (CAR DATE)
                                                                                     'CREATIONDATE)
                                                                                 then (IDATE (CADR DATE))
                                                                               else (FIXP (CADR DATE]
                                                                           (\ILLEGAL.ARG (CADR DATE]
                                                             (REMOVE DATE OTHERINFO]
```

;; Force everything through GetStreamForFile to (even if it was already a stream) to force the file system to check the
;; directory and rebuild the stream and all info cached in it.

```
                              (if (type? DLIONSTREAM FILE)
                                  then (SETQ FILE (fetch (DLIONSTREAM FULLFILENAME) of FILE)))
                              (SETQ STREAM (\LFGetStreamForFile FILE RECOG ACCESS (NEQ ACCESS 'INPUT)
                                                                OTHERINFO OLDSTREAM))
```

;; If GetStreamForFile returned something other than a stream, there was some error; abort.

```
                              (if (NOT (type? DLIONSTREAM STREAM))
                                  then (RETURN STREAM))
                              (if (NOT OLDSTREAM)
                                  then                                  ; Don't do this for REOPENFILE
                                       (if (EQ ACCESS 'OUTPUT)
                                           then                         ; File is EMPTY even if it is old
                                                (replace EPAGE of STREAM with (replace EOFFSET of STREAM with 0)))
                                       (\LFUpdateLeaderPage STREAM (AND (NOT (FMEMB 'DON'T.CHANGE.DATE OTHERINFO))
                                                                        ACCESS)
                                              IDATE)                    ; Update access dates.
                                       (if IDATE
                                           then                         ; Don't be tempted to change it if other things change
                                                (replace NONDEFAULTDATEFLG of STREAM with T)))
```

;; Set the validation field to be the creation date

```
                              (replace (DLIONSTREAM VALIDATION) of STREAM with (fetch (LeaderPage TimeCreate)
                                                                                      of (fetch (DLIONSTREAM LEADERPAGE)
                                                                                                of STREAM)))
```

;; Return the stream you've just built.

```
                              (RETURN STREAM)))]
```

```
        (COND
           ((type? DLIONSTREAM STREAM)
            STREAM)
           ((NULL STREAM)
            NIL)
           ((TYPEP STREAM 'CONDITION)
            (CL:ERROR STREAM))
           (T (CL:ERROR STREAM :PATHNAME FILE])
```

(\LFGetStreamForFile

```
[LAMBDA (NAME RECOG ACCESS CREATEFLG OTHERINFO OLDSTREAM)       ; Edited 20-Aug-88 17:43 by bvm
```

;; Creates a STREAM for dsk file NAME, creating it if necessary when CREATEFLG is true.

```
(PROG ((FILESPEC (\LFFileSpec NAME RECOG))
       (volNum NIL)
       (DIRPTR NIL))
      (RETURN (COND
                 ((NULL FILESPEC)
```

;; If the file does not have a valid file specification, don't create a stream; just return NIL.

```
                  NIL)
                 [(SETQ DIRPTR (fetch (DFSFileSpec FSDIRPTR) of FILESPEC))
```

;; If the directory code found a pointer into the directory, then the file already exists; just open it up

```
                  (LET [(FULLNAME (\LFFullFileName (fetch (DFSFileSpec EXPANDEDNAME) of FILESPEC]
                       (if (AND (NULL OLDSTREAM)
                                (\FILE-CONFLICT FULLNAME ACCESS \LFdevice))
                           then                                  ; Busy. Don't check from REOPENFILE
                                (MAKE-CONDITION 'XCL:FILE-WONT-OPEN :PATHNAME FULLNAME)
                         else (\LFOpenOldFile (create FileDescriptor
                                                      fileID _ (\LFReadFileID
                                                                 [\LFGetDirectory (SETQ volNum
                                                                                        (fetch (ExpandedName VOLNUM)
                                                                                           of (fetch (DFSFileSpec
                                                                                                        EXPANDEDNAME
                                                                                                     )
                                                                                              of FILESPEC]
                                                                 DIRPTR)
                                                      volNum _ volNum
                                                      type _ tLispFile)
                                              FULLNAME DIRPTR]
                 ((NULL (fetch (ExpandedName VERSION) of (fetch (DFSFileSpec EXPANDEDNAME) of FILESPEC)))
                  NIL)
                 ((IGREATERP (fetch (ExpandedName VERSION) of (fetch (DFSFileSpec EXPANDEDNAME) of FILESPEC))
                        MAX.SMALLP)
                  (printout PROMPTWINDOW T "Version number too high")
                  'XCL:FS-RESOURCES-EXCEEDED)
                 (CREATEFLG (\LFCreateFile (fetch (DFSFileSpec EXPANDEDNAME) of FILESPEC)
                                           OTHERINFO])
```

(\LFOpenOldFile

```
[LAMBDA (fileDesc fullFileName directoryPointer)                ; Edited 20-Aug-88 18:05 by bvm
```

;; Open an old (existing) file and return the resultant stream

```
(LET* ((leaderPage (create LeaderPage))
       (STREAM (create DLIONSTREAM
                       FULLFILENAME _ fullFileName
                       FILEDESC _ fileDesc
                       DIRINFO _ directoryPointer
                       DEVICE _ \LFdevice
                       LEADERPAGE _ leaderPage))
       SIZE LASTPAGE OFFSET)
```

;; Use the volume file map to find out what size the file is; record this in the stream you are building.

```
      (SETQ SIZE (\PFFindFileSize fileDesc))
      (replace (FileDescriptor size) of fileDesc with SIZE)
```

;; Read in the leader page for the file. The leader page has stream-level eof information on it. It also has backing file length info on it. If
;; this latter matches the length found from the vfm, then believe the leader page and use its eof info for the stream; else, the leader page
;; is probably screwed up, so just make the stream's eof be the entire backing file. (This means you won't lose any info, but might gain
;; about half a page of nulls.)

```
      (\PFGetPage fileDesc 0 (\PFFindPageAddr fileDesc 0)
             leaderPage)
      (if (EQL (fetch (LeaderPage AllocatedPages) of leaderPage)
               SIZE)
          then (SETQ LASTPAGE (fetch EofPage of leaderPage))
               (SETQ OFFSET (fetch EOffSet of leaderPage))
        else (SETQ LASTPAGE (SUB1 SIZE))
             (SETQ OFFSET BYTESPERPAGE))
      (replace (DLIONSTREAM EPAGE) of STREAM with LASTPAGE)
      (replace (DLIONSTREAM EOFFSET) of STREAM with OFFSET)
```

;; Finally return the stream you've just built

```
      STREAM])
```

(\LFGenFileID

```
[LAMBDA (vol)                                                    (* amd "10-Feb-86 16:04")

;;; Generates and returns a new file ID and updates the ID count for the logical volume

    (add (fetch (LogicalVolumeDescriptor lastIDAllocated) of vol)
         1])
```

(\LFCreateFile

```
[LAMBDA (fileName info)                                          ; Edited 20-Aug-88 17:37 by bvm
    ;; fileName: UNAME, pages: FIXP (estimated length of file;  currently not taken advantage of), info: PLIST
    ;; Creates a file by allocating the pages for it and returning a stream to it.
    (UNINTERRUPTABLY
        (PROG ((vol (\PFGetVol (fetch (ExpandedName VOLNUM) of fileName)))
               stream DIRINDEX)
              (SETQ stream (create DLIONSTREAM
                                  FULLFILENAME _ (\LFFullFileName fileName)
                                  FILEDESC _ (create FileDescriptor
                                                     fileID _ (\LFGenFileID vol)
                                                     volNum _ (\PFVolumeNumber vol)
                                                     type _ tLispFile)
                                  DEVICE _ \LFdevice))
        ;; Make sure there's enough space for the directory entry.
              (if [NULL (SETQ DIRINDEX (\LFFindDirHole stream fileName (\LFGetDirectory vol]
                  then (RETURN 'XCL:FS-RESOURCES-EXCEEDED))
        ;; Allocate pages for file;  this will update size field of FileDescriptor
              (if (NULL (\PFNewPages vol (fetch (DLIONSTREAM FILEDESC) of stream)
                                     (create PageGroup
                                             filePage _ 0
                                             volumePage _ 0
                                             nextFilePage _ \LFrunSize)))
                  then (RETURN 'XCL:FS-RESOURCES-EXCEEDED))
        ;; Create leader page for the new file and put it and cache it
              (replace (DLIONSTREAM LEADERPAGE) of stream with (\LFMakeLeaderPage (fetch (DLIONSTREAM FILEDESC)
                                                                                        of stream)
                                                                                  (\LFFileName fileName)
                                                                                  info))
        ;; Enter the new file in the directory
              (\LFMakeDirEntry stream fileName (\LFGetDirectory vol)
                     DIRINDEX)
              (RETURN stream)))])
```

(\LFMakeLeaderPage

```
[LAMBDA (file fileName Info)                                     ; Edited 16-Apr-87 17:55 by jop
    ;; Make, put, and return leader page for file
    (DECLARE (GLOBALVARS DEFAULTFILETYPE))
    (PROG ((TYPE (OR (CADR (FASSOC 'TYPE Info))
                     DEFAULTFILETYPE))
           (CurrentTime (OR (FIXP (CADR (FASSOC 'CREATIONDATE Info)))
                            (IDATE)))
           (Author (OR (CADR (FASSOC 'AUTHOR Info))
                       (USERNAME)))
           (LeaderPage (create LeaderPage)))
          (replace (LeaderPage TYPE) of LeaderPage with TYPE)
          (replace (LeaderPage TimeCreate) of LeaderPage with CurrentTime)
          (replace (LeaderPage TimeWrite) of LeaderPage with CurrentTime)
          (replace (LeaderPage FileID) of LeaderPage with (fetch (FileDescriptor fileID) of file))
          (replace (LeaderPage AllocatedPages) of LeaderPage with (fetch (FileDescriptor size) of file))
          (replace (LeaderPage EofPage) of LeaderPage with 0)
          (replace (LeaderPage EOffSet) of LeaderPage with 0)
          (replace (LeaderPage fileName) of LeaderPage with fileName)
          (replace (LeaderPage author) of LeaderPage with Author)
          (replace (LeaderPage version) of LeaderPage with lispFileVersion)
          (\PFPutPage file 0 (\PFFindPageAddr file 0)
                 LeaderPage)
          (RETURN LeaderPage])
```

(\LFUpdateLeaderPage

```
[LAMBDA (stream access createDate)                               ; Edited 20-Aug-88 17:59 by bvm
  (UNINTERRUPTABLY
      (PROG [(leaderPage (fetch (DLIONSTREAM LEADERPAGE) of stream))
             (time (AND access (DAYTIME]
      ;; Update end of file info
            (replace (LeaderPage EofPage) of leaderPage with (fetch (STREAM EPAGE) of stream))
            (replace (LeaderPage EOffSet) of leaderPage with (fetch (STREAM EOFFSET) of stream))
```

```
        ;; Update info saying how many pages have been allocated to the file
            (replace (LeaderPage AllocatedPages) of leaderPage with (fetch (FileDescriptor size)
                                                                        of (fetch (DLIONSTREAM FILEDESC)
                                                                              of stream)))
        ;; Update access times
            (SELECTQ access
                ((OUTPUT BOTH APPEND)
                    (replace (LeaderPage TimeWrite) of leaderPage with time)
                    (replace (LeaderPage TimeCreate) of leaderPage with (OR createDate (SETQ createDate time)))
                    (replace (DLIONSTREAM VALIDATION) of stream with createDate))
                NIL)
            (SELECTQ access
                ((INPUT BOTH)
                    (replace (LeaderPage TimeRead) of leaderPage with time))
                NIL)
        ;; and write out the refreshed leader page
            (\LFWriteLeaderPage stream)))])
```

(\LFWriteLeaderPage

```
  [LAMBDA (stream)                                                    (* hts%: " 5-Jan-85 16:15")
    (PROG ((vol (fetch (DLIONSTREAM VOLUME) of stream))
           (fileDesc (fetch (DLIONSTREAM FILEDESC) of stream)))
          (\PFPutPage fileDesc 0 (\PFFindPageAddr fileDesc 0)
                 (fetch (DLIONSTREAM LEADERPAGE) of stream])

)

(DEFINEQ
```

(\LFCloseFile

```
  [LAMBDA (STREAM)                                                    (* hdj "25-Sep-86 13:43")
```

;;; Closes the specified stream.

```
    (WITH.MONITOR \LFtopMonitor
```

;;; Write out and dispense with buffers for this stream.

```
        (\CLEARMAP STREAM)
        (if (NEQ (fetch ACCESS of STREAM)
                 'INPUT)
            then
```

;;; Update the stream eof info, trim the backing file so that it is just big enough to hold the stream, and record all the eof info on the stream's leader page.
;;; Minimum backing file length for the stream is computed as follows: 1 page for leader page; 1 page because stream pages (in particular EPAGE) are
;;; numbered from 0, not 1; EPAGE of stream pages; less 1 page if the EOFFSET is 0

```
                (UNINTERRUPTABLY
                    (\LFTruncateFile STREAM)
                    (\PFTrimHelper (fetch (DLIONSTREAM VOLUME) of STREAM)
                           (fetch (DLIONSTREAM FILEDESC) of STREAM)
                           (PLUS 1 1 (fetch EPAGE of STREAM)
                                  (if (EQ (fetch EOFFSET of STREAM)
                                          0)
                                      then -1
                                    else 0)))
                    (\LFUpdateLeaderPage STREAM)))
        (\PFSaveBuffers (fetch (DLIONSTREAM VOLUME) of STREAM))
        STREAM)])

)

(DEFINEQ
```

(\LFDeleteFile

```
  [LAMBDA (fileName dev)                                              ; Edited 11-May-2023 21:36 by lmm
                                                                      (* hdj "23-Jun-86 16:47")
    (WITH.MONITOR \LFtopMonitor
        (PROG ((stream (\LFGetStreamForFile fileName 'OLDEST 'BOTH NIL NIL)))
              (if (OR (NOT (type? DLIONSTREAM stream))
                      (FDEVOP 'OPENP dev (fetch FULLFILENAME of stream)
                              NIL dev))
                  then (RETURN))
              (UNINTERRUPTABLY
                  (\LFRemoveDirEntry stream (\LFGetDirectory (fetch (DLIONSTREAM VOLUME) of stream)))
                  ;; Take the entire file out of the BTree and out of the allocation map
                  (\PFTrimHelper (fetch (DLIONSTREAM VOLUME) of stream)
                         (fetch (DLIONSTREAM FILEDESC) of stream)
                         0)
                  ;; save buffers
                  (\PFSaveBuffers (fetch (DLIONSTREAM VOLUME) of stream)))
              (RETURN (fetch (DLIONSTREAM FULLFILENAME) of stream)))])
```

```
)

(DEFINEQ
```

(\LFReadPages

```
  [LAMBDA (stream streamFirstPage buffers)                    ; Edited 22-Oct-87 16:03 by amd
```

;; Reads a bunch of pages from stream, starting at firstPage. Returns number of bytes read.

;; Modified ' 4-Jul-85 04:47:22' by HTS to extend the backing file whenever it tries to read past the end of the backing file. This generally ensures ;; that data subsequently written on these buffer pages will not be lost if you run out of disk space

;; If asked to read a page which is off the end of the stream, it will zero the page. Odd though it may seem, reading off the end of the file is ;; reasonable behavior for copybytes: buffer pages must come from somewhere, and copybytes may not have to write the whole page, and in ;; general copybytes does not know whether a page is actually in a file or off the end of it. Seems inefficient, but since reading past eof does not ;; actually require disk access, its not that bad.

;; Extend backing file if necessary to accomodate buffers.

```
    (\LFExtendFileIfNecessary stream streamFirstPage buffers)
```

;; Write out the buffers to the backing file.

```
    (for buffer inside buffers as streamPageNumber from streamFirstPage as backingFilePageNumber
       from (ADD1 streamFirstPage) bind (file _ (fetch (DLIONSTREAM FILEDESC) of stream))
                                        lastStreamPage offset
       first (\UPDATEOF stream)
             (SETQ lastStreamPage (PLUS (fetch (DLIONSTREAM EPAGE) of stream)
                                        (if (CL:ZEROP (fetch (DLIONSTREAM EOFFSET) of stream))
                                            then -1
                                          else 0)))
       sum (if (ILEQ streamPageNumber lastStreamPage)
               then
```

;; If page inside stream, then it has presumably already been written; read it in.

```
                    (\PFGetPage file backingFilePageNumber (\PFFindPageAddr file backingFilePageNumber)
                          buffer)
```

;; If this was the last page in the file, then fill in the trailing bytes with nulls.

```
                    (if (EQL streamPageNumber lastStreamPage)
                        then (SETQ offset (fetch (DLIONSTREAM EOFFSET) of stream))
                             (if (CL:ZEROP offset)
                                 then (SETQ offset BYTESPERPAGE)
                               else (\CLEARBYTES buffer offset (DIFFERENCE BYTESPERPAGE offset)))
                             offset
                      else BYTESPERPAGE)
             else
```

;; If this was outside the stream, clear the buffer.

```
                  (\CLEARWORDS buffer WORDSPERPAGE)
                  0])

)

(DEFINEQ
```

(\LFWritePages

```
  [LAMBDA (stream streamFirstPage buffers)                    ; Edited 16-Apr-87 16:08 by jop
```

;; Writes a bunch of pages to stream, starting at streamFirstPage

;; Extend backing file if necessary to accomodate buffers.

```
    (if (fetch (STREAM REVALIDATEFLG) of stream)
        then
```

;; Need to update creationdate, since a SAVEVM etc has occurred since the last write. Otherwise, it is possible to see a change to the ;; file but no change to the creationdate

```
             (\LFUpdateLeaderPage stream 'OUTPUT)
             (replace (STREAM REVALIDATEFLG) of stream with NIL))
    (\LFExtendFileIfNecessary stream streamFirstPage buffers)
```

;; Write out the buffers to the backing file.

```
    (for buffer inside buffers as backingFilePageNumber from (ADD1 streamFirstPage)
       bind (file _ (fetch (DLIONSTREAM FILEDESC) of stream)) do (\PFPutPage file backingFilePageNumber
                                                                        (\PFFindPageAddr file
                                                                               backingFilePageNumber)
                                                                        buffer))
    NIL])
```

(\LFExtendFileIfNecessary

```
  [LAMBDA (stream streamFirstPage buffers)                    (* hts%: "13-Aug-85 14:21")
```

;;; Extends the backing file for stream to make space for buffers. Must not be called from uninterruptable or monitorlocked code. Causes a continuable ;;; error if there are not enough free pages for the extension.

```
    (PROG ((runLength (if (NLISTP buffers)
                          then 1
                        else (LENGTH buffers)))
           minBackingFileSize)
```

;; Backing file (Pilot file) enumeration starts with leader page of file, Lisp stream page enumeration does not include the leader page; hence the ;; first 1.0 Pages are enumerated from 0 but size is enumerated from 1; hence the second 1.0

```
          (SETQ minBackingFileSize (PLUS 1 1 streamFirstPage (SUB1 runLength)))
```

;; Extend backing file if necessary.

```
          (until (WITH.MONITOR \LFtopMonitor
                    (if (GREATERP minBackingFileSize (fetch (FileDescriptor size) of (fetch (DLIONSTREAM FILEDESC)
                                                                                        of stream)))
                        then (\LFExtendFile stream minBackingFileSize)
                      else T))
             do (LISPERROR "FILE SYSTEM RESOURCES EXCEEDED" (fetch (DLIONSTREAM FULLFILENAME) of stream)
                       T])
```

(\LFExtendFile

```
  [LAMBDA (stream minBackingFileSize)                                    (* hts%: "13-Aug-85 13:07")
```

;;; Extends the backing file for stream so that its backing file is at least minBackingFileSize.

```
    (PROG ((vol (fetch (DLIONSTREAM VOLUME) of stream))
           (fileDesc (fetch (DLIONSTREAM FILEDESC) of stream)))
          (UNINTERRUPTABLY
              (OR [\PFNewPages vol fileDesc (create PageGroup
                                                    filePage _ (fetch (FileDescriptor size) of fileDesc)
                                                    volumePage _ 0
                                                    nextFilePage _ (MAX minBackingFileSize
                                                                        (IPLUS (fetch (FileDescriptor size)
                                                                                  of fileDesc)
                                                                               \LFrunSize]
                  (RETURN NIL))
              (\UPDATEOF stream)
              (\LFUpdateLeaderPage stream))
          (RETURN stream])

)

(DEFINEQ
```

(\LFGetFileInfo

```
  [LAMBDA (stream attribute device)                                      ; Edited 20-Aug-88 17:19 by bvm
```

;;; Get the value of the attribute for a file. If stream is a filename, then the file is not open. If stream is a STREAM, then it is open and has valid
;;; information in it.

```
    (WITH.MONITOR \LFtopMonitor
        [AND [OR (type? DLIONSTREAM stream)
                 (type? DLIONSTREAM (SETQ stream (\LFGetStreamForFile stream 'OLD 'INPUT NIL NIL]
             (PROG ((infoPage (fetch (DLIONSTREAM LEADERPAGE) of stream)))
                   (RETURN (SELECTQ attribute
                                (LENGTH (\UPDATEOF stream)
                                        (IPLUS (ITIMES (fetch (STREAM EPAGE) of stream)
                                                       BYTESPERPAGE)
                                               (fetch (STREAM EOFFSET) of stream)))
                                (SIZE (\UPDATEOF stream)
                                      (IPLUS (fetch (STREAM EPAGE) of stream)
                                             (FOLDHI (fetch (STREAM EOFFSET) of stream)
                                                     BYTESPERPAGE)))
                                (TYPE (fetch (LeaderPage TYPE) of infoPage))
                                (WRITEDATE (GDATE (fetch (LeaderPage TimeWrite) of infoPage)))
                                (READDATE (GDATE (fetch (LeaderPage TimeRead) of infoPage)))
                                (CREATIONDATE (GDATE (fetch (LeaderPage TimeCreate) of infoPage)))
                                (IWRITEDATE (fetch (LeaderPage TimeWrite) of infoPage))
                                (IREADDATE (fetch (LeaderPage TimeRead) of infoPage))
                                (ICREATIONDATE
                                    (fetch (LeaderPage TimeCreate) of infoPage))
                                (AUTHOR (fetch (LeaderPage author) of infoPage))
                                NIL])])
```

(\LFSetFileInfo

```
  [LAMBDA (stream attribute value dev)                                   ; Edited 20-Aug-88 17:18 by bvm
    (WITH.MONITOR \LFtopMonitor
        [AND [OR (type? DLIONSTREAM stream)
                 (type? DLIONSTREAM (SETQ stream (\LFGetStreamForFile stream 'OLD 'INPUT NIL NIL]
             (PROG ((infoPage (fetch (DLIONSTREAM LEADERPAGE) of stream)))
                   (RETURN (if (SELECTQ attribute
                                   (TYPE (replace (LeaderPage TYPE) of infoPage with value))
                                   (CREATIONDATE (replace (LeaderPage TimeCreate) of infoPage
                                                     with (OR (IDATE value)
                                                              (\ILLEGAL.ARG value))))
                                   (ICREATIONDATE
                                       (replace (LeaderPage TimeCreate) of infoPage with value))
                                   NIL)
                               then (\LFUpdateLeaderPage stream)
                                    T])])

)

(DEFINEQ
```

(\LFGetFileName

```
  [LAMBDA (FileName Recog Dev)                                    (* amd "10-Feb-86 16:04")
```

;;; Maps a filename onto a fully specified filename if it exists, or onto NIL if it doesn't exist.

```
    (WITH.MONITOR \LFtopMonitor
        [LET ((fileSpec (\LFFileSpec FileName Recog)))
             (AND fileSpec (\LFFullFileName (fetch (DFSFileSpec EXPANDEDNAME) of fileSpec])])

)

(DEFINEQ
```

(\LFEventFn

```
  [LAMBDA (Dev Event)                                             ; Edited 13-Sep-88 16:38 by hayata
```

;; Determines dliondisk fdev behaviour across major system events. Must make the file system wake up properly on different machines, or even on
;; the same machine with a different disk partitioning.

```
    (WITH.MONITOR \LFtopMonitor
        (SELECTQ Event
            ((AFTERLOGOUT AFTERSYSOUT AFTERMAKESYS AFTERSAVEVM)
                 (\LFCloseDevice)
                 (\PFEnsureInitialized T)                         ; force reinitialization
                 (\LFOpenDevice)                                  ; reopen if possible
                 [if (DEFINEDP 'DSKDISPLAY)
                     then (DSKDISPLAY (DSKDISPLAY 'CLOSED]        ; handle the DSKDISPLAY window, if there is one
                 ;; If on an alien machine, make sure you won't attempt to reopen files. Note that if you're still on a dlion or dove, the reopenfile
                 ;; method will not break, but will simply return NIL if the file isn't there (e.g. if someone deleted it since this Lisp image was last
                 ;; run, or if the disk changed).
                 (SELECTQ (MACHINETYPE)
                      ((DANDELION DOVE)
                          NIL)
                      (LET NIL (replace (FDEV REOPENFILE) of Dev with (FUNCTION NILL))
                           (\REMOVEDEVICE Dev)))
                 ;; revalidate open streams (should probably move this into the SELECTQ above)
                 (\PAGED.REVALIDATEFILELST Dev))
            ((BEFORELOGOUT BEFORESYSOUT BEFOREMAKESYS BEFORESAVEVM)
                                                                  ;; BVM claims you should flush open streams associated with
                                                                  ;; this device only before logout
                 (if (EQ Event 'BEFORELOGOUT)
                     then (\FLUSH.OPEN.STREAMS Dev))
                 (for vol in (\PFGetVols) when (\LFDirectoryP vol) do
                                                                  ; flush output buffers.
                                                                       (\PFSaveBuffers vol)))
            NIL))])

)

(DEFINEQ
```

(\LFDirectoryNameP

```
  [LAMBDA (DirSpec)                                               (* amd "10-Feb-86 16:04")
```

;;; Implements the DIRECTORYNAMEP method for the dlionfs. If DirSpec is a reasonable directory specification, returns the canonical form of that
;;; directory; otherwise returns NIL

;;; DirSpec (a) must parse correctly, (b) must have a proper directory associated with it, and (c) might have a subdirectory nestled in it.

```
    (WITH.MONITOR \LFtopMonitor
        [LET (PARSED DIR SUBDIREND)
             (AND (SETQ PARSED (\LFParseFileName DirSpec))
                  (SETQ DIR (\LFFindDirectoryVol (fetch (PARSEDFILENAME VOL) of PARSED)))
                  (PACKFILENAME.STRING 'HOST 'DSK 'DIRECTORY (U-CASE (fetch (LogicalVolumeDescriptor LVlabel)
                                                                        of DIR))
                        'NAME
                        (AND (SETQ SUBDIREND (FIXP (LASTCHPOS (CHARCODE >)
                                                              (fetch (PARSEDFILENAME NAME) of PARSED)
                                                              1)))
                             (U-CASE (SUBSTRING (fetch (PARSEDFILENAME NAME) of PARSED)
                                                1 SUBDIREND])])

)

(DEFINEQ
```

(\LFTruncateFile

```
  [LAMBDA (STREAM PAGE# OFFSET)                                   (* amd "10-Feb-86 16:04")
```

;;; Used to shorten or lengthen STREAM. If lengthening, pad the file with nulls. Used by SETEOFPTR and FORCEOUTPUT.

;; Normalize arguments

```
    (\UPDATEOF STREAM)
    (OR (FIXP PAGE#)
```

```
        (SETQ PAGE# (fetch (DLIONSTREAM EPAGE) of STREAM)))
    (OR (FIXP OFFSET)
        (SETQ OFFSET (fetch (DLIONSTREAM EOFFSET) of STREAM)))
```

;; If lengthening stream, pad it with nulls.

```
    (UNINTERRUPTABLY
        (PROG ((FILEPTR (\GETFILEPTR STREAM))
               [curEof (PLUS (TIMES (fetch (DLIONSTREAM EPAGE) of STREAM)
                                    BYTESPERPAGE)
                             (TIMES (fetch (DLIONSTREAM EOFFSET) of STREAM]
               (curPages (fetch (LeaderPage AllocatedPages) of (fetch (DLIONSTREAM LEADERPAGE) of STREAM)))
               (needPages (IQUOTIENT (DIFFERENCE (PLUS (ITIMES (PLUS PAGE# 1)
                                                               BYTESPERPAGE)
                                                       OFFSET BYTESPERPAGE)
                                                 1)
                                     BYTESPERPAGE)))
              (if (IGREATERP needPages curPages)
                  then (\LFExtendFile STREAM needPages))
              (\SETFILEPTR STREAM curEof)
              (to (DIFFERENCE (PLUS (TIMES PAGE# BYTESPERPAGE)
                                    OFFSET)
                              curEof)
                  do (\BOUT STREAM 0))
              (\SETFILEPTR STREAM FILEPTR)))
```

;; Record the new file length

```
    (replace (DLIONSTREAM EPAGE) of STREAM with PAGE#)
    (replace (DLIONSTREAM EOFFSET) of STREAM with OFFSET)
    (\LFUpdateLeaderPage STREAM)
    (\PFSaveBuffers (fetch (DLIONSTREAM VOLUME) of STREAM))
    NIL])

)

(DEFINEQ
```

(\LFRenameFile

```
  [LAMBDA (OLD-DEVICE OLD-NAME NEW-DEVICE NEW-NAME)                ; Edited 20-Feb-87 17:59 by amd
    (if (NEQ OLD-DEVICE NEW-DEVICE)
        then (\GENERIC.RENAMEFILE OLD-DEVICE OLD-NAME NEW-DEVICE NEW-NAME)
      else ;; The following test should be in the generic rename function. How come it's here? [bvm: Generic system isn't supposed to know.
           ;; However, this recognize hack is silly.  You should check whether the file is open AFTER you call \LFFileSpec and have obtained its full
           ;; name, so that you don't have to do recognition twice (in \recognize-hack and \LFFileSpec).]
           (if (NOT (FDEVOP 'OPENP OLD-DEVICE (\RECOGNIZE-HACK OLD-NAME 'OLD OLD-DEVICE)
                            NIL OLD-DEVICE))
               then (CL:WHEN [NULL (fetch (DFSFileSpec FSDIRPTR) of (\LFFileSpec OLD-NAME 'OLD]
                             (LISPERROR "FILE NOT FOUND" OLD-NAME))
                    (PROG [[dir (\LFFindDirectory (CADR (\LFParseFileName OLD-NAME]
                                (FILESPEC (\LFFileSpec NEW-NAME 'NEW]
                          (if [EQ dir (\LFFindDirectory (CADR (\LFParseFileName NEW-NAME]
                              then (WITH.MONITOR \LFtopMonitor
                                                 (LET* ((stream (\LFGetStreamForFile OLD-NAME 'OLD))
                                                        (oldPtr (fetch (DLIONSTREAM DIRINFO) of stream))
                                                        (newPtr (\LFFindDirHole stream (fetch (DFSFileSpec EXPANDEDNAME)
                                                                                          of FILESPEC)
                                                                                dir)))
                                                       (SETQ NEW-NAME (\LFFullFileName (fetch (DFSFileSpec EXPANDEDNAME)
                                                                                          of FILESPEC)))
                                                       (if (NULL newPtr)
                                                           then (SETQ OLD-NAME "FILE SYSTEM RESOURCES EXCEEDED")
                                                         else (\LFMakeDirEntry stream (fetch (DFSFileSpec EXPANDEDNAME)
                                                                                         of FILESPEC)
                                                                               dir newPtr)
                                                              (replace (DLIONSTREAM DIRINFO) of stream with oldPtr)
                                                              (\LFRemoveDirEntry stream dir)
                                                              (replace (DLIONSTREAM DIRINFO) of stream with newPtr)
                                                              (replace (DLIONSTREAM FULLFILENAME) of stream with NEW-NAME)
                                                              (replace (LeaderPage fileName) of (fetch (DLIONSTREAM LEADERPAGE)
                                                                                                   of stream)
                                                                 with (\LFFileName (\LFUnpackName NEW-NAME)))
                                                              (replace (LeaderPage TimeWrite) of (fetch (DLIONSTREAM LEADERPAGE
                                                                                                                    )
                                                                                                   of stream)
                                                                 with (DAYTIME))
                                                              (\LFWriteLeaderPage stream))))
                                    (if (EQUAL OLD-NAME "FILE SYSTEM RESOURCES EXCEEDED")
                                        then (LISPERROR "FILE SYSTEM RESOURCES EXCEEDED" NEW-NAME T)
                                      else (RETURN NEW-NAME))
                              else (RETURN (\GENERIC.RENAMEFILE OLD-DEVICE OLD-NAME NEW-DEVICE NEW-NAME])

)

(RPAQQ LFDIRECTORYCOMS
       [
```

;;; This module handles the Lisp directory part of the file system. The Lisp directory maps literal file names onto Pilot file ID numbers (which can then be ;;; looked up in the volume file map). This module used to be in the file LFDIRECTORY.

```
        ;; Known problem: the directory is currently stored as a list rather than a tree, so searches in a large directory take quite some time.
        (DECLARE%: EVAL@COMPILE DONTCOPY (CONSTANTS (directorySize 50))
               (RECORDS GenerateFileState GeneratedFile DIRSEARCHSTATE PARSEDFILENAME ExpandedName DFSFileSpec)
               (MACROS CONDCONCAT)
               (MACROS PRINTDIRECTORY))
        ;; Format of a directory entry is :
        ;; bang (check ;  should always contain !)
        ;; type (0 = hole, 1 = file)
        ;; entryLength
        ;; fileID (4 bytes)
        ;; version# (2 bytes)
        ;; filenameLength
        ;; filename (filenameLength bytes)
        ;; Routines for mapping file names onto volumes and directories
        (FNS \LFFindDirectory \LFFindDirectoryVol \LFParseFileName)
        ;; Creating and opening directories
        (FNS \LFMakeVolumeDirectory \LFDirectoryP \LFPurgeDirectory \LFCloseDirectory)
        ;; Functions for making, deleting, and finding entries in a directory.
        (FNS \LFMakeDirEntry \LFRemoveDirEntry \LFReadFileID \LFFindDirHole \LFMakeDirHole \LFCheckBang)
        (FNS \LFDirectorySearch \LFVersions)
        (FNS \LFFileSpec \LFUnpackName \LFFullFileName \LFFileName)
        (FNS \LFDirectoryScrambled)
        (FNS \LFDWIN \LFDWOUT)
        ;; Directory enumeration
        (FNS \LFGenerateFiles \LFFindNextFile \LFSortFiles \LFHighestVersions \LFFindInfo \LFReturnNextFile
             \LFReturnInfo)
        (GLOBALVARS \LFtopMonitor)
        ;; Holding onto directory streams
        (FNS \LFGetDirectory \LFPutDirectory \LFCreateDirectories)
        (GLOBALVARS \LFdirectories)
        (P (\LFCreateDirectories))
        ;; Case array manipulation
        (FNS \LFINITCASEARRAY \LFCASEARRAYFETCH)
        (GLOBALVARS \LFCASEARRAY \DISKNAMECASEARRAY)
        (INITVARS (\LFCASEARRAY (\LFINITCASEARRAY])
```

```
;;; This module handles the Lisp directory part of the file system.  The Lisp directory maps literal file names onto Pilot file ID numbers (which can then be
;;; looked up in the volume file map).  This module used to be in the file LFDIRECTORY.
;; Known problem: the directory is currently stored as a list rather than a tree, so searches in a large directory take quite some time.

(DECLARE%: EVAL@COMPILE DONTCOPY

(DECLARE%: EVAL@COMPILE

(RPAQQ directorySize 50)

(CONSTANTS (directorySize 50))
)

(DECLARE%: EVAL@COMPILE

(TYPERECORD GenerateFileState (CURRENTFILE RESTOFFILES ATTRIBUTES))

(TYPERECORD GeneratedFile (FULLNAME NAME VERSION INFO))

(TYPERECORD DIRSEARCHSTATE (DIRPTR CHARLIST))

(TYPERECORD PARSEDFILENAME (VOL NAME VERSION))

(TYPERECORD ExpandedName (VOLNUM CHARLIST VERSION)

            (* VERSION is the version indicator (either a positive integer or one of OLD, OLDEST, NEW) -
            VOLNUM is the logical volume number, -
            and the CHARLIST is a list of characters in the name.)

       )

(TYPERECORD DFSFileSpec (EXPANDEDNAME FSDIRPTR))
)

(DECLARE%: EVAL@COMPILE

(PUTPROPS CONDCONCAT MACRO [ARGS `(CONCATLIST (for STR in %, (CONS 'LIST ARGS) when STR collect STR])
)
```

```
(DECLARE%: EVAL@COMPILE

(PUTPROPS PRINTDIRECTORY MACRO [LAMBDA (STREAM)                    (* hts%: " 6-Aug-85 12:19")

          (* * Prints the contents of a Lisp directory -- for debugging.)

                                        (SETFILEPTR (\DTEST STREAM 'STREAM)
                                               0)
                                        (bind TYPE LENGTH START until (\EOFP STREAM)
                                           do (SETQ START (GETFILEPTR STREAM))
                                              (\LFCheckBang STREAM)
                                              (SETQ TYPE (BIN STREAM))
                                              (SETQ LENGTH (BIN STREAM))
                                              (if (EQ TYPE 1)
                                                  then (printout NIL (\WIN STREAM)
                                                                     " "
                                                                     (\WIN STREAM)
                                                                     " "
                                                                     (\WIN STREAM)
                                                                     " "
                                                                     (PACKC (to (BIN STREAM) collect (BIN STREAM)))
                                                                     T))
                                              (SETFILEPTR STREAM (PLUS START (TIMES LENGTH BYTESPERWORD])
)
)

;; Format of a directory entry is :
;; bang (check ;  should always contain !)
;; type (0 = hole, 1 = file)
;; entryLength
;; fileID (4 bytes)
;; version# (2 bytes)
;; filenameLength
;; filename (filenameLength bytes)
;; Routines for mapping file names onto volumes and directories

(DEFINEQ

(\LFFindDirectory
  [LAMBDA (VOL)                                                   (* amd "10-Feb-86 16:04")

;;; Maps a volume name, descriptor, or number onto the directory stream for that volume.  If the volume name is NIL, finds the default directory stream.
;;; Opens the directory if it is not already open.  If there is no appropriate directory stream, returns NIL.

    (SETQ VOL (\LFFindDirectoryVol VOL))
    (AND VOL (\LFDirectoryP VOL])

(\LFFindDirectoryVol
  [LAMBDA (VOL)                                                   (* amd "10-Feb-86 16:04")

;;; Maps a volume name, descriptor, or number into the descriptor for that volume provided the volume has a proper Lisp directory on it.  If VOL is NIL,
;;; finds the descriptor of the volume containing the default Lisp directory.  If there is no appropriate volume, returns NIL.

    (if VOL
        then ;; Normalize argument
             (COND
                ((type? LogicalVolumeDescriptor VOL))
                ((FIXP VOL)
                 (SETQ VOL (\PFGetVol VOL)))
                ((OR (ATOM VOL)
                     (STRINGP VOL))
                 (SETQ VOL (\PFGetLVPage VOL)))
                (T (SHOULDNT)))
             ;; Tell whether the specified volume has a proper Lisp directory on it.
             (AND VOL (\LFDirectoryP VOL)
                  VOL)
      else ;; Find the descriptor for the volume with the default Lisp directory on it.
           (PROG ((volumes (\PFGetVols))
                  (currentVol (\PFCurrentVol))
                  nextVolumes defaultVol)
                 [SETQ nextVolumes (for vols on volumes do (if (EQ currentVol (CAR vols))
                                                               then (RETURN (APPEND vols volumes]
                 (RETURN (for vol in nextVolumes thereis (\LFDirectoryP vol])

(\LFParseFileName
  [LAMBDA (FULLNAME)                                              ; Edited 22-Oct-87 16:06 by amd
```

```
    ;; Returns the parse of a filename
    (PROG (DIRECTORY NAME EXT VERSION ENDVOLNAME)
          (if (for TAIL on (UNPACKFILENAME.STRING FULLNAME) by (CDDR TAIL)
                 do (SELECTQ (CAR TAIL)
                             (HOST NIL)
                             (DIRECTORY (SETQ DIRECTORY (CADR TAIL)))
                             (NAME (SETQ NAME (CADR TAIL)))
                             (EXTENSION (SETQ EXT (CADR TAIL)))
                             (VERSION (SETQ VERSION (CADR TAIL)))
                             (RETURN T)))
              then (RETURN))
          (SETQ ENDVOLNAME (STRPOS ">" DIRECTORY))
          (RETURN (create PARSEDFILENAME
                          VOL _ [AND DIRECTORY (SUBSTRING DIRECTORY 1 (AND ENDVOLNAME (SUB1 ENDVOLNAME]
                          NAME _ (CONDCONCAT (AND ENDVOLNAME (SUBSTRING DIRECTORY (ADD1 ENDVOLNAME)))
                                             (AND ENDVOLNAME ">")
                                             NAME "." EXT)
                          VERSION _ (if (CL:ZEROP (NCHARS VERSION))
                                        then NIL
                                      else (MKATOM VERSION])

)

;; Creating and opening directories

(DEFINEQ

(\LFMakeVolumeDirectory
  [LAMBDA (vol DONTOPEN)                                    ; Edited 9-Jan-87 19:01 by amd
    ;; Creates a Lisp directory for vol
    (UNINTERRUPTABLY
        (PROG ((directoryID (\LFGenFileID vol))
               file)
          ;; Allocate and record pages for the directory file
              (SETQ file (create FileDescriptor
                                 fileID _ directoryID
                                 volNum _ (\PFVolumeNumber vol)
                                 type _ tLispDirectory
                                 size _ 0))
              (OR (\PFNewPages vol file (create PageGroup
                                                filePage _ 0
                                                volumePage _ 0
                                                nextFilePage _ directorySize))
                  (DiskError "FILE SYSTEM RESOURCES EXCEEDED"))
              (\PFSaveBuffers vol)
          ;; Make and put a leader page for the directory file; dlionstream created here is just a throwaway
              (\LFMakeLeaderPage file (PACKFILENAME.STRING 'NAME 'DIRECTORY 'VERSION 1)
                    NIL)
          ;; Put pointer to this directory in the volume root directory
              (\PFInsertDirectoryID vol tLispDirectory directoryID))
        ;; Open up the new directory
        (if DONTOPEN
            then NIL
          else (\LFDirectoryP vol)))])

(\LFDirectoryP
  [LAMBDA (vol)                                             ; Edited 22-Oct-87 16:07 by amd
    ;; If there is a valid Lisp directory on volume vol, opens it (if it isn't already open) and returns it; otherwise returns NIL. For there to be a valid
    ;; directory, the volume must be a Pilot volume, there must be a root directory on it with a Lisp directory entry, there must be an openable Lisp
    ;; directory file, and the leader page of that file must have the correct file system version number on it.
    (PROG (directoryID stream)
          (RETURN (OR (AND (type? DLIONSTREAM (\LFGetDirectory vol))
                           (\LFGetDirectory vol))
                      (AND (\PFPilotVolumeP vol)
                           (SETQ directoryID (\PFFindDirectoryID vol tLispDirectory))
                           (SETQ stream (\LFOpenOldFile (create FileDescriptor
                                                                fileID _ (\PFFindDirectoryID vol tLispDirectory)
                                                                volNum _ (\PFVolumeNumber vol)
                                                                type _ tLispDirectory)
                                                        (PACKFILENAME 'NAME 'DIRECTORY 'VERSION 1)
                                                        NIL))
                           (EQL (fetch (LeaderPage version) of (fetch (DLIONSTREAM LEADERPAGE) of stream))
                                lispFileVersion)
                           (PROGN (replace ACCESS of stream with 'BOTH)
                                  (replace MAXBUFFERS of stream with MAX.SMALLP)
                                  (\OPENFILE stream)
                                  (\LFPutDirectory vol stream])
```

(\LFPurgeDirectory

```
[LAMBDA (vol)                                                    (* amd "10-Feb-86 16:04")
    ;; CLose the directory if it is open
    (\LFCloseDirectory vol)
    ;; Take directory off disk if it is there
    (PROG ((directoryID (\PFFindDirectoryID vol tLispDirectory))
           file)
          (if directoryID
              then (\PFRemoveDirectoryID vol tLispDirectory)
                   (SETQ file (create FileDescriptor
                                      fileID _ directoryID
                                      volNum _ (\PFVolumeNumber vol)
                                      type _ tLispDirectory))
                   (replace (FileDescriptor size) of file with (\PFFindFileSize file))
                   (\PFTrimHelper vol file 0])
```

(\LFCloseDirectory

```
[LAMBDA (vol)                                                    (* amd "10-Feb-86 16:04")
;;; Remove internal record of directory
    (if (\LFGetDirectory vol)
        then (FORGETPAGES (\LFGetDirectory vol))
             (\LFPutDirectory vol NIL])
)
;; Functions for making, deleting, and finding entries in a directory.
(DEFINEQ
```

(\LFMakeDirEntry

```
[LAMBDA (stream UNAME DirStream POS)                              ; Edited 22-Oct-87 16:08 by amd
    ;; Makes a directory entry for a new file
    (PROG ((NC (LENGTH (fetch (ExpandedName CHARLIST) of UNAME)))
           SIZE)
     ;; SIZE is how big the directory entry must be.  The 10 is 1 byte !, 1 byte type, 1 byte entry length, 4 bytes fileID, 2 bytes version, 1 byte string
     ;; length (for filename)
          (SETQ SIZE (IPLUS NC 10))
     ;; Check entry and move to fileID field
          (\SETFILEPTR DirStream POS)
          (\LFCheckBang DirStream)
          (OR (CL:ZEROP (\BIN DirStream))
              (\LFDirectoryScrambled DirStream))
          (OR (IGEQ (\BIN DirStream)
                    SIZE)
              (\LFDirectoryScrambled DirStream))
          (UNINTERRUPTABLY
              ;; Write out fileID
              (\LFDWOUT DirStream (fetch (FileDescriptor fileID) of (fetch (DLIONSTREAM FILEDESC) of stream)))
              ;; Write out version number
              (\WOUT DirStream (fetch (ExpandedName VERSION) of UNAME))
              ;; Write out filename preceded by number of chars in it (ie, as a bcpl string)
              (\BOUT DirStream NC)
              (for C in (fetch (ExpandedName CHARLIST) of UNAME) do (\BOUT DirStream C))
              ;; When everything is ready, finally change the type from hole to file
              (\SETFILEPTR DirStream (ADD1 POS))
              (\BOUT DirStream 1))
     ;; Remember where file is in directory
          (replace (DLIONSTREAM DIRINFO) of stream with POS)
     ;; Write changes to directory file out to disk
          (FORCEOUTPUT DirStream])
```

(\LFRemoveDirEntry

```
[LAMBDA (stream dirStream)                                        ; Edited 22-Oct-87 16:09 by amd
    ;; Change type of dir entry to hole and write changed directory pages out to disk
    (UNINTERRUPTABLY
        (\SETFILEPTR dirStream (fetch (DLIONSTREAM DIRINFO) of stream))
        (\LFCheckBang dirStream)
        (\BOUT dirStream 0))
    ;; Merge with following hole, if there is one
    (UNINTERRUPTABLY
```

```
        [PROG ((ENTRYSIZE (\BIN dirStream))
               NEWENTRYSIZE)
              (\SETFILEPTR dirStream (PLUS (fetch (DLIONSTREAM DIRINFO) of stream)
                                           ENTRYSIZE))
              (if (NOT (\EOFP dirStream))
                  then (\LFCheckBang dirStream)
                       (if (CL:ZEROP (\BIN dirStream))
                           then (SETQ NEWENTRYSIZE (PLUS ENTRYSIZE (\BIN dirStream)))
                                (\SETFILEPTR dirStream (IPLUS (fetch (DLIONSTREAM DIRINFO) of stream)
                                                              2))
                                (if (ILESSP NEWENTRYSIZE 256)
                                    then (\BOUT dirStream NEWENTRYSIZE])
```

;; Force the altered directory out to disk

```
(FORCEOUTPUT dirStream])
```

(\LFReadFileID

```
[LAMBDA (directory position)                                    (* hts%: "11-Jan-85 02:05")
```

;;; Returns the file ID recorded in the entry beginning at position

```
(\SETFILEPTR directory position)
```

;; bang

```
(\LFCheckBang directory)
```

;; Make sure its not a hole

```
(if (NEQ (BIN directory)
         1)
    then (\LFDirectoryScrambled))
```

;; Entry length

```
(\BIN directory)
```

;; Finally read in the file id

```
(\LFDWIN directory])
```

(\LFFindDirHole

```
[LAMBDA (STREAM UNAME DIRSTREAM)                                ; Edited 22-Oct-87 16:18 by amd
```

;; Finds or creates a hole in the directory large enough to fit the entry represented by UNAME. Returns the byte address of the hole if sucessful, ;; NIL otherwise. BYTES is how big the entry must be. The 10 is 1 byte !, 1 byte type, 1 byte entry length, 4 bytes fileID, 2 bytes version, 1 byte ;; string length (for filename)

```
(bind [BYTES _ (IPLUS 10 (LENGTH (fetch (ExpandedName CHARLIST) of UNAME]
      (PTR _ (OR (fetch (DLIONSTREAM DIRHOLEPTR) of DIRSTREAM)
                 0))
      ENTRYLENGTH TYPE do (\SETFILEPTR DIRSTREAM PTR)
                          (if (\EOFP DIRSTREAM)
                              then ;; Make a new entry at the end of the file
                                   (RETURN (if (\LFMakeDirHole DIRSTREAM PTR BYTES)
                                               then PTR
                                               else NIL))
                            else (\LFCheckBang DIRSTREAM)
                                 (SETQ TYPE (\BIN DIRSTREAM))
                                 (SETQ ENTRYLENGTH (\BIN DIRSTREAM))
                                 (if (AND (CL:ZEROP TYPE)
                                          (ILEQ BYTES ENTRYLENGTH))
                                     then ;; Entry big enough
                                          (if (IGEQ ENTRYLENGTH (PLUS BYTES 14))
                                              then ;; Too large, so break it apart. (Too large if there is room for another entry with
                                                   ;; filename of 3 or more chars.)
                                                   (UNINTERRUPTABLY
                                                       (\LFMakeDirHole DIRSTREAM (PLUS PTR BYTES)
                                                                       (DIFFERENCE ENTRYLENGTH BYTES))
                                                       (\LFMakeDirHole DIRSTREAM PTR BYTES)))
                                          (RETURN PTR)))
                          (SETQ PTR (IPLUS PTR ENTRYLENGTH])
```

(\LFMakeDirHole

```
[LAMBDA (DIRSTREAM WHERE HOLESIZE)                              ; Edited 22-Oct-87 16:20 by amd
```

;; Makes an empty slot in the directory; this slot will soon be used to hold a directory entry. Returns DIRSTREAM if successful, NIL otherwise.

```
(PROG [(DIRSIZE (fetch (FileDescriptor size) of (fetch (DLIONSTREAM FILEDESC) of DIRSTREAM]
```

;; Extends the directory if necessary.

```
      (if (ILEQ (TIMES BYTESPERPAGE (SUB1 DIRSIZE))
                (IPLUS WHERE HOLESIZE))
          then (if (NULL (\LFExtendFile DIRSTREAM (ADD1 DIRSIZE)))
                   then (RETURN NIL)))
      (UNINTERRUPTABLY
          (\SETFILEPTR DIRSTREAM WHERE)
```

```
            ;; Mark beginning of entry
            (\BOUT DIRSTREAM (CHARCODE !))
            ;; Mark as hole
            (\BOUT DIRSTREAM 0)
            ;; Note size of hole
            (\BOUT DIRSTREAM HOLESIZE)
            ;; Pad rest with nulls.
            (to (IDIFFERENCE HOLESIZE 3) do (\BOUT DIRSTREAM 0)))
    ;; Flush to disk.
        (FORCEOUTPUT DIRSTREAM)
        (RETURN DIRSTREAM])
```

(\LFCheckBang

```
  [LAMBDA (DIRSTREAM)                                    (* amd "10-Feb-86 16:04")
          (* * comment)
    (OR (EQ (BIN DIRSTREAM)
            (CHARCODE !))
        (\LFDirectoryScrambled DIRSTREAM])
)

(DEFINEQ
```

(\LFDirectorySearch

```
  [LAMBDA (DIRSTREAM TLIST HMIN KINDOFMATCH)             ; Edited 22-Oct-87 16:21 by amd
    ;; Finds next directory entry for which (CDR TLIST) is a prefix of the filename.  Returns NIL if no entry found, else the length of the remaining chars
    ;; in the entry.  Leaves the directory positioned after the char matching the last char of TLIST::1 --- DIRSTREAM is the ofd of the directory file ---
    ;; TLIST is a list of the form (POS . CHARPAIRS), where POS at entry is a fileptr in the directory file at which to start searching and CHARPAIRS
    ;; is like the characters pairs of a uname.  At exit, TLIST is smashed so that POS is the fileptr just beyond the found entry.  --- if HMIN~=NIL, sets
    ;; STREAM's DIRHOLEPTR to NIL or the fileptr of the first hole of at least HMIN words.
    (bind (MATCH _ NIL)
          (NEXT _ (fetch (DIRSEARCHSTATE DIRPTR) of TLIST))
          (CHARLIST _ (fetch (DIRSEARCHSTATE CHARLIST) of TLIST))
          THISNAMELENGTH TARGETLENGTH PTR TYP ENTRYLENGTH FILEID VERSION
       first (if HMIN
                 then (replace (DLIONSTREAM DIRHOLEPTR) of DIRSTREAM with NIL))
             (SETQ TARGETLENGTH (LENGTH CHARLIST))
       until MATCH do (\SETFILEPTR DIRSTREAM (SETQ PTR NEXT))
                      (if (\EOFP DIRSTREAM)
                          then (RETURN))
                      ;; Format of a directory entry is --- bang (check ;  should always contain !) --- type (0 = hole, 1 = file) --- entryLength --- fileID
                      ;; (4 bytes) --- version# (2 bytes) --- filenameLength --- filename (filenameLength bytes)
                      (\LFCheckBang DIRSTREAM)
                      (SETQ TYP (\BIN DIRSTREAM))
                      (SETQ ENTRYLENGTH (\BIN DIRSTREAM))
                      (SETQ NEXT (IPLUS PTR ENTRYLENGTH))
                      [if (CL:ZEROP TYP)
                          then ;; Not a file;  if hole is of right length etc., cache its position
                               (if (AND HMIN (ILEQ HMIN ENTRYLENGTH))
                                   then (replace (DLIONSTREAM DIRHOLEPTR) of DIRSTREAM with PTR)
                                        (SETQ HMIN NIL))
                        else (SETQ FILEID (\LFDWIN DIRSTREAM))
                             (SETQ VERSION (\WIN DIRSTREAM))
                             (SETQ THISNAMELENGTH (\BIN DIRSTREAM))
                             (if (OR (AND (EQ KINDOFMATCH 'EXACT)
                                          (EQL THISNAMELENGTH TARGETLENGTH))
                                     (AND (EQ KINDOFMATCH 'PARTIAL)
                                          (IGEQ THISNAMELENGTH TARGETLENGTH)))
                                 then (SETQ MATCH (for C in CHARLIST always (EQ C (\LFCASEARRAYFETCH (\BIN
                                                                                                      DIRSTREAM
                                                                                                      ]
       finally ;; Leave directory file pointer at beginning of entry
               (\SETFILEPTR DIRSTREAM PTR)
               ;; Remember where next entry is
               (replace (DIRSEARCHSTATE DIRPTR) of TLIST with NEXT)
               ;; Return the number of unmatched chars
               (RETURN (IDIFFERENCE THISNAMELENGTH TARGETLENGTH])
```

(\LFVersions

```
  [LAMBDA (UNPACKEDNAME STREAM HMIN)                     ; Edited 22-Oct-87 16:23 by amd
    ;; UNPACKEDNAME is a value of \UNPACKFILENAME.  STREAM is the directory ofd.  HMIN=T means look for a hole big enough for UNAME, a
    ;; number N means look for that size hole, NIL means don't look.  Returns a list of (version . fileptr) pairs sorted by increasing version.  Ptr is a
```

```
    ;; pointer to the beginning of the directory slot for the file.
    (bind (TLIST _ (create DIRSEARCHSTATE
                           DIRPTR _ 0
                           CHARLIST _ (fetch (ExpandedName CHARLIST) of UNPACKEDNAME)))
          (FIXEDVERSION _ (FIXP (fetch (ExpandedName VERSION) of UNPACKEDNAME)))
          PTR RESULT version first (OR (NULL FIXEDVERSION)
                                       (GREATERP FIXEDVERSION 0)
                                       (SETQ FIXEDVERSION NIL))
                                   (if (EQ HMIN T)
                                       then (SETQ HMIN 20))
       do [if (NULL (\LFDirectorySearch STREAM TLIST HMIN 'EXACT))
              then (RETURN (SORT RESULT (FUNCTION (LAMBDA (A B)
                                                    (LESSP (CAR A)
                                                           (CAR B]

          ;; DirectorySearch leaves directory file ptr at beginning of entry.  Record beginning of entry
          (SETQ PTR (\GETFILEPTR STREAM))

          ;; Read up to version number
          (\LFCheckBang STREAM)                                    ; Bang!
          (OR (EQL (\BIN STREAM)
                   1)
              (\LFDirectoryScrambled))                             ; type = file
          (\BIN STREAM)                                            ; Entry length
          (\LFDWIN STREAM)                                         ; file ID

          ;; Read version number
          (SETQ version (\WIN STREAM))

          ;; Name matches.  version is the version number.  Cons up a piece of the result.  If UNPACKEDNAME has an explicit version, insist on it
          ;; now
          (if FIXEDVERSION
              then [if (EQL version FIXEDVERSION)
                       then (RETURN (LIST (CONS version PTR]
            else ;; Merge new element into RESULT
                 (push RESULT (CONS version PTR)))

          ;; Stop looking if found a hole
          (if (AND HMIN (fetch (DLIONSTREAM DIRHOLEPTR) of STREAM))
              then (SETQ HMIN NIL])

)

(DEFINEQ
```

(\LFFileSpec

```
  [LAMBDA (NAME RECOG)                                             ; Edited 20-Oct-87 12:34 by amd
    ;; This returns a full file specification, with all the information needed to do open, delete, etc.  A filespec is a (packedname unpackedname dirptr)
    ;; triple, with the true version number smashed into the uname.  The dirptr is NIL if the file does not currently exist in the directory.
    (PROG (dirPtr version versionList (UNPACKEDNAME (\LFUnpackName NAME))
                DIRSTREAM)
     ;; If name didn't unpack properly, return NIL
          (OR UNPACKEDNAME (RETURN))
     ;; If there is no directory for the specified name, return NIL
          (OR DIRSTREAM (SETQ DIRSTREAM (\LFFindDirectory (fetch (ExpandedName VOLNUM) of UNPACKEDNAME)))
              (RETURN))
     ;; Build file specification
          [COND
             ([AND (SETQ versionList (\LFVersions UNPACKEDNAME DIRSTREAM (SELECTQ RECOG
                                                                                  ((NEW OLD/NEW)
                                                                                       T)
                                                                                  NIL)))
                   (SETQ version (SELECTQ (OR (fetch (ExpandedName VERSION) of UNPACKEDNAME)
                                              RECOG)
                                          ((OLD OLD/NEW)
                                               (CAR (LAST versionList)))
                                          (NEW                     ; A new version, so the DIRPTR is NIL
                                               [LIST (ADD1 (CAAR (LAST versionList])
                                          (OLDEST (CAR versionList))
                                          (ASSOC (fetch (ExpandedName VERSION) of UNPACKEDNAME)
                                                 versionList]
              (SETQ dirPtr (CDR version))
              (SETQ version (CAR version)))
             (T (SETQ dirPtr NIL)                                  ; Since file doesnt exist, recognition mode takes precedence
                                                                   ; over version number
                (SETQ version (SELECTQ (OR RECOG (fetch (ExpandedName VERSION) of UNPACKEDNAME))
                                       ((NEW OLD/NEW)
                                            (OR (FIXP (fetch (ExpandedName VERSION) of UNPACKEDNAME))
                                                1))
                                       ((OLD OLDEST)
                                          NIL)
                                       (FIXP (fetch (ExpandedName VERSION) of UNPACKEDNAME]
```

```
                                                                    ; We may have to zap a version number that was specified but
                                                                    ; not found
        (replace (ExpandedName VERSION) of UNPACKEDNAME with version)
        (RETURN (create DFSFileSpec
                        EXPANDEDNAME _ UNPACKEDNAME
                        FSDIRPTR _ dirPtr])
```

(\LFUnpackName

```
  [LAMBDA (name)                                                    ; Edited 20-Oct-87 12:34 by amd
```

;; Unpacks file name into a UNAME of the form ((VERSION . VOLNUM) . CHARLIST) where VERSION is the version indicator (either a positive ;; integer or one of OLD, OLDEST, NEW) VOLNUM is the logical volume number, and the CHARLIST is a list of characters in the name. Returns ;; NIL if the given name is not valid.

```
    (PROG ((PARSEDNAME (\LFParseFileName name))
           VOL charList version)
          (OR PARSEDNAME (RETURN))
          (SETQ VOL (\LFFindDirectoryVol (fetch (PARSEDFILENAME VOL) of PARSEDNAME)))
          (OR VOL (RETURN))
          (SETQ charList (for char instring (fetch (PARSEDFILENAME NAME) of PARSEDNAME)
                            collect                                 ; check for illegal chars
                                   (SETQ char (\LFCASEARRAYFETCH char))
                                   (if [FMEMB char (LIST 0 (\LFCASEARRAYFETCH (CHARCODE *))
                                                           (\LFCASEARRAYFETCH (CHARCODE ?]
                                       then (RETURN NIL))
                                   char))
          (OR charList (RETURN))
          (SETQ version (fetch (PARSEDFILENAME VERSION) of PARSEDNAME))
          (SETQ version (OR (FIXP version)
                            (SELECTQ version
                                 (H 'OLD)
                                 (L 'OLDEST)
                                 (N 'NEW)
                                NIL)))
          (RETURN (create ExpandedName
                          VOLNUM _ (\PFVolumeNumber VOL)
                          CHARLIST _ charList
                          VERSION _ version])
```

(\LFFullFileName

```
  [LAMBDA (UNPACKEDNAME)                                            (* amd "10-Feb-86 16:04")
```

;;; Puts together a full file name (including host, directory, subdirectory, name, and version) from a uname

```
    (AND (fetch (ExpandedName VERSION) of UNPACKEDNAME)
         (PACKFILENAME 'HOST 'DSK 'DIRECTORY [U-CASE (fetch (LogicalVolumeDescriptor LVlabel)
                                                            of (\PFGetVol (fetch (ExpandedName VOLNUM) of UNPACKEDNAME
                                                                                                                  ]
               'NAME
               (\LFFileName UNPACKEDNAME])
```

(\LFFileName

```
  [LAMBDA (UNPACKEDNAME)                                            (* amd "10-Feb-86 16:04")
```

;;; Puts together the subdirectory, filename, and version of a file from its uname

```
    (PROG ((CHARLIST (fetch (ExpandedName CHARLIST) of UNPACKEDNAME))
           (VERSION (CHCON (OR (FIXP (fetch (ExpandedName VERSION) of UNPACKEDNAME))
                               1)))
           CHARLISTLENGTH NAME)
          (SETQ CHARLISTLENGTH (LENGTH CHARLIST))
          [SETQ NAME (ALLOCSTRING (PLUS CHARLISTLENGTH 1 (LENGTH VERSION]
          (for I from 1 as CHAR in CHARLIST do (RPLCHARCODE NAME I CHAR))
          (RPLCHARCODE NAME (ADD1 CHARLISTLENGTH)
                 (CHARCODE ;))
          (for I from (PLUS CHARLISTLENGTH 2) as CHAR in VERSION do (RPLCHARCODE NAME I CHAR))
          (RETURN NAME])

)

(DEFINEQ
```

(\LFDirectoryScrambled

```
  [LAMBDA (DIRSTREAM)                                               (* hts%: "16-Jan-85 17:01")

          (* * comment)

    (printout PROMPTWINDOW "Local directory scrambled: " T [PACKFILENAME.STRING
                                                            'HOST
                                                            'DSK
                                                            'DIRECTORY
                                                            (U-CASE (fetch (LogicalVolumeDescriptor LVlabel)
                                                                       of (fetch (DLIONSTREAM VOLUME)
                                                                             of DIRSTREAM]
           T "Try scavenging the directory.")
```

```
    (DiskError "HARD DISK ERROR"])

)

(DEFINEQ
```

(\LFDWIN

```
  [LAMBDA (FILE)                                                    (* jds " 3-JAN-83 16:08")
    (IPLUS (LLSH (\BIN FILE)
                 24)
           (LLSH (\BIN FILE)
                 16)
           (LLSH (\BIN FILE)
                 8)
           (\BIN FILE])
```

(\LFDWOUT

```
  [LAMBDA (FILE NUMBER)                                             (* jds " 3-JAN-83 15:30")
    (\BOUT FILE (LOGAND 255 (LRSH NUMBER 24)))
    (\BOUT FILE (LOGAND 255 (LRSH NUMBER 16)))
    (\BOUT FILE (LOGAND 255 (LRSH NUMBER 8)))
    (\BOUT FILE (LOGAND 255 NUMBER])

)
```

;; Directory enumeration

```
(DEFINEQ
```

(\LFGenerateFiles

```
  [LAMBDA (FDEV PATTERN DESIREDPROPS)                               ; Edited 22-Oct-87 16:25 by amd
    ;; Returns a file-generator object that will generate exactly those files in the sys-dir of FDEV whose names match PATTERN.
    (WITH.MONITOR \LFtopMonitor
        [PROG (PARSED DIRECTORYSTREAM SEARCHSTATE GENFILTER HOST&DIRNAME NEXTFILE FILELIST)
              [SETQ PARSED (OR (\LFParseFileName PATTERN)
                               (RETURN (\NULLFILEGENERATOR]
              [SETQ DIRECTORYSTREAM (OR (\LFFindDirectory (fetch (PARSEDFILENAME VOL) of PARSED))
                                        (RETURN (\NULLFILEGENERATOR]
              (SETQ SEARCHSTATE (create DIRSEARCHSTATE
                                       DIRPTR _ 0
                                       CHARLIST _ (for C instring (fetch (PARSEDFILENAME NAME) of PARSED)
                                                     until (SELCHARQ (SETQ C (\LFCASEARRAYFETCH C))
                                                                     ((%# *)
                                                                      ;; \LFDirectorySearch currently only checks prefixes, so we truncate at the first * or
                                                                      ;; escape.  Also ignore version specifications,
                                                                      T)
                                                                     NIL)
                                                     collect C)))
              [SETQ GENFILTER (DIRECTORY.MATCH.SETUP (CONDCONCAT (fetch (PARSEDFILENAME NAME) of PARSED)
                                                                 ";"
                                                                 (fetch (PARSEDFILENAME VERSION) of PARSED]
              [SETQ HOST&DIRNAME (PACKFILENAME.STRING 'HOST 'DSK 'DIRECTORY (U-CASE (fetch (
                                                                                    LogicalVolumeDescriptor
                                                                                      LVlabel)
                                                                                 of (fetch (DLIONSTREAM
                                                                                            VOLUME)
                                                                                      of DIRECTORYSTREAM]
        ;; Generate a list of all the files that match the spec.
              (while (SETQ NEXTFILE (\LFFindNextFile DIRECTORYSTREAM SEARCHSTATE GENFILTER HOST&DIRNAME))
                 do (push FILELIST NEXTFILE))
        ;; Sort the list of files.  Not all directory enumeration requests require sorting, but almost all do, so I just sort them all for simplicity.
              (\LFSortFiles FILELIST)
        ;; Highest version enumeration: if the pattern does not have a version, then should return only the highest version of each file.
        ;; \LFHighestVersions requires that the file list be sorted first.
              (if (OR (CL:ZEROP (NCHARS (fetch (PARSEDFILENAME VERSION) of PARSED)))
                      (NULL (fetch (PARSEDFILENAME VERSION) of PARSED)))
                  then (SETQ FILELIST (\LFHighestVersions FILELIST)))
        ;; Dig up any file info that the caller has indicated he will request.  (During the enumeration, the user can ask for any of the file properties in
        ;; DESIREDPROPS.) This is done here and stored (rather than later when it is actually requested) to avoid the problem of a file having been
        ;; deleted by another process before its properties could be dug up.  Here that is safe, since this is being done under the top-level file system
        ;; monitorlock.
              (\LFFindInfo FILELIST DESIREDPROPS DIRECTORYSTREAM)
        ;; Finally return the file generator object.
              (RETURN (create FILEGENOBJ
                              NEXTFILEFN _ (FUNCTION \LFReturnNextFile)
                              FILEINFOFN _ (FUNCTION \LFReturnInfo)
                              GENFILESTATE _ (create GenerateFileState
                                                     CURRENTFILE _ NIL
                                                     RESTOFFILES _ FILELIST
```

```
                                                    ATTRIBUTES _ DESIREDPROPS])])
```

(\LFFindNextFile

```
[LAMBDA (directory SEARCHSTATE FILTER HOST&DIRNAME)                    (* amd "10-Feb-86 16:04")
```

;;; Finds the next file in directory that matches the specified filter, and returns its name, version, directory position, etc., if there is one.

```
(bind (ANOTHERENTRY _ NIL)
      ENTRYSTART VERSION FILENAME CHARS NAMELEN
   do (SETQ ANOTHERENTRY (\LFDirectorySearch directory SEARCHSTATE NIL 'PARTIAL))
      [if ANOTHERENTRY
          then ;; \LFDirectorySearch leaves directory file ptr at beginning of entry.  Read name and version.
               (SETQ ENTRYSTART (\GETFILEPTR directory))
               (\LFCheckBang directory)                                ; bang
               (OR (EQ (\BIN directory)
                       1)
                   (\LFDirectoryScrambled))                            ; type
               (\BIN directory)                                        ; entry length
               (\LFDWIN directory)                                     ; file ID
               (SETQ VERSION (\WIN directory))                         ; version
               (SETQ NAMELEN (\BIN directory))
               (SETQ CHARS (to NAMELEN collect (\BIN directory)))
                                                                       ; name
               ;; Construct the name of the file
               (SETQ FILENAME (\LFFileName (create ExpandedName
                                                   CHARLIST _ CHARS
                                                   VERSION _ VERSION]
   repeatuntil (OR (NOT ANOTHERENTRY)
                   (NOT FILTER)
                   (DIRECTORY.MATCH FILTER FILENAME))
   finally (RETURN (if ANOTHERENTRY
                       then (create GeneratedFile
                                    FULLNAME _ (CONCAT HOST&DIRNAME FILENAME)
                                    NAME _ (SUBSTRING FILENAME 1 NAMELEN)
                                    VERSION _ VERSION
                                    INFO _ ENTRYSTART)
                     else NIL])
```

(\LFSortFiles

```
[LAMBDA (FILES)                                                        (* amd "10-Feb-86 18:52")
```

;;; Sorts the list of generated files. Not all requests for directory enumeration require that the files be sorted, but most do, so I just sort them all. Note
;;; that in comparing names, you must not compare the version part of the name (hence the SUBSTRING stuff), since ALPHORDER does not get
;;; versions in the right order.

```
[SORT FILES (FUNCTION (LAMBDA (A B)
                        (SELECTQ (UALPHORDER (fetch (GeneratedFile NAME) of A)
                                             (fetch (GeneratedFile NAME) of B))
                             (LESSP T)
                             (EQUAL (LESSP (fetch (GeneratedFile VERSION) of A)
                                           (fetch (GeneratedFile VERSION) of B)))
                             NIL]
NIL])
```

(\LFHighestVersions

```
[LAMBDA (FILELIST)                                                     (* amd "10-Feb-86 16:04")
```

;;; Extracts the highest version files from a list of sorted files.

```
(for FILES on FILELIST when [NOT (AND (LISTP (CDR FILES))
                                      (type? GeneratedFile (CADR FILES))
                                      (STREQUAL (fetch (GeneratedFile NAME) of (CAR FILES))
                                                (fetch (GeneratedFile NAME) of (CADR FILES]
   collect (CAR FILES])
```

(\LFFindInfo

```
[LAMBDA (FILES PROPS DIRECTORY)                                        (* amd "10-Feb-86 16:04")
```

;;; Digs up any file info that the caller has indicated he will request. (During the enumeration, the user can ask for any of the file properties in
;;; DESIREDPROPS.) This is done here and stored (rather than later when it is actually requested) to avoid the problem of a file having been deleted by
;;; another process before its properties could be dug up. Here that is safe, since this is being done under the top-level file system monitorlock. This info
;;; is later read and returned to the user by \LFReturnInfo.

```
(if (LISTP PROPS)
    then (bind ENTRYSTART STREAM (BACKWARDPROPS _ (REVERSE PROPS)) for FILE in FILES
            do ;; Build a stream for the current file;  this stream will be used and reused for getting the file attributes.  Kind of a weird entry to
               ;; the OpenFile stuff, but that's because you already have your finger on the directory entry and don't have to bother looking it
               ;; up again.
               (SETQ ENTRYSTART (fetch (GeneratedFile INFO) of FILE))
               (replace (GeneratedFile INFO) of FILE with NIL)
```

```
                    (SETQ STREAM (\LFOpenOldFile (create FileDescriptor
                                                         fileID _ (\LFReadFileID DIRECTORY ENTRYSTART)
                                                         volNum _ (fetch (FileDescriptor volNum)
                                                                     of (fetch (DLIONSTREAM FILEDESC) of DIRECTORY))
                                                         type _ tLispFile)
                                         NIL ENTRYSTART))
                    (replace ACCESS of STREAM with 'INPUT)
                    ;; Now get all the info and save it.
                    (for ATTRIBUTE in BACKWARDPROPS do (push (fetch (GeneratedFile INFO) of FILE)
                                                             (GETFILEINFO STREAM ATTRIBUTE])
```

(\LFReturnNextFile

```
  [LAMBDA (GENERATED)                                                    (* amd "10-Feb-86 16:04")

          (* * comment)

    (if (NULL (fetch (GenerateFileState RESTOFFILES) of GENERATED))
        then NIL
      else (replace (GenerateFileState CURRENTFILE) of GENERATED with (pop (fetch (GenerateFileState RESTOFFILES)
                                                                              of GENERATED)))
           (fetch (GeneratedFile FULLNAME) of (fetch (GenerateFileState CURRENTFILE) of GENERATED])
```

(\LFReturnInfo

```
  [LAMBDA (GENERATED PROP)                                               ; Edited 20-Aug-88 17:23 by bvm

          (* * comment)

    (for ATTRIB in (fetch (GenerateFileState ATTRIBUTES) of GENERATED) as INFOVAL
       in (fetch (GeneratedFile INFO) of (fetch (GenerateFileState CURRENTFILE) of GENERATED))
       do (if (EQ ATTRIB PROP)
              then (RETURN INFOVAL])

)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \LFtopMonitor)
)

;; Holding onto directory streams

(DEFINEQ
```

(\LFGetDirectory

```
  [LAMBDA (vol)                                                          (* hts%: " 5-Jan-85 15:49")
    (ELT \LFdirectories (OR (FIXP vol)
                            (\PFVolumeNumber vol])
```

(\LFPutDirectory

```
  [LAMBDA (vol directory)                                                (* amd "10-Feb-86 16:04")
    (SETA \LFdirectories (OR (FIXP vol)
                             (\PFVolumeNumber vol))
          directory])
```

(\LFCreateDirectories

```
  [LAMBDA NIL                                                            ; Edited 22-Oct-87 16:26 by amd
    (if [NOT (AND (BOUNDP '\LFdirectories)
                  (type? ARRAYP \LFdirectories)
                  (ZEROP (ARRAYORIG \LFdirectories))
                  (EQL maxLogicalVolumes (ARRAYSIZE \LFdirectories]
        then (SETQ \LFdirectories (ARRAY maxLogicalVolumes NIL NIL 0))
             (SETQ \PFInitialized NIL))
    NIL])

)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \LFdirectories)
)

(\LFCreateDirectories)

;; Case array manipulation

(DEFINEQ
```

(\LFINITCASEARRAY

```
  [LAMBDA NIL                                                            ; Edited 20-Aug-88 18:06 by bvm
    ;; \DISKNAMECASEARRAY is a case array set up by mod44io.  Unfortunately,it counts > as an illegal filename char, so we need to make a copy
    ;; with that fixed.
```

```
    (PROG ((CASEARRAY (COPYARRAY \DISKNAMECASEARRAY)))
          (\PUTBASEBYTE (fetch (ARRAYP BASE) of CASEARRAY)
                 27 0)                                          ; ESC
          (for C from (CHARCODE "!") to (CHARCODE "9") do (\PUTBASEBYTE (fetch (ARRAYP BASE) of CASEARRAY)
                                                                 C C))
          (for C from (CHARCODE "<") to (CHARCODE "`") do (\PUTBASEBYTE (fetch (ARRAYP BASE) of CASEARRAY)
                                                                 C C))
          (\PUTBASEBYTE (fetch (ARRAYP BASE) of CASEARRAY)
                 (CHARCODE "|")
                 (CHARCODE "|"))
          (\PUTBASEBYTE (fetch (ARRAYP BASE) of CASEARRAY)
                 (CHARCODE "~")
                 (CHARCODE "~"))
          (RETURN CASEARRAY])
```

(**LFCASEARRAYFETCH**

```
  [LAMBDA (CHARCODE)                                            ; Edited 20-Oct-87 12:24 by amd
    (\GETBASEBYTE (fetch (ARRAYP BASE) of \LFCASEARRAY)
           CHARCODE])

)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \LFCASEARRAY \DISKNAMECASEARRAY)
)

(RPAQ? \LFCASEARRAY (\LFINITCASEARRAY))

(RPAQQ SCAVENGEDSKDIRECTORYCOMS (
```

;;; This module contains routines for scavenging the Lisp directory in the event that it should become smashed. It used to be in the file
;;; SCAVENGEDSKDIRECTORY.

```
                                        ;; Directory (LFDIRECTORY) level stuff

(FNS FILENAMEFROMID SCAVENGEDSKDIRECTORY SCAVENGEVOLUME \LFScavFileName
     \LFScavVersion)
(GLOBALVARS \LFtopMonitor)
                                        ;; Volume file map (LFFILEMAP) level stuff

(FNS \VFMGenerateFileIDs)))
```

;;; This module contains routines for scavenging the Lisp directory in the event that it should become smashed. It used to be in the file
;;; SCAVENGEDSKDIRECTORY.

;; Directory (LFDIRECTORY) level stuff

```
(DEFINEQ
```

(**FILENAMEFROMID**

```
  [LAMBDA (lowhalf highhalf volumename)                         ; Edited  5-Feb-88 19:38 by amd
    (LET ((stream (\LFFindDirectory volumename))
          name)
         (SETFILEPTR stream 0)
         (bind start length until (OR name (EOFP stream))
            do (SETQ start (GETFILEPTR stream))
               (\LFCheckBang stream)
               [if (AND (EQL (PROG1 (BIN stream)
                                    (SETQ length (BIN stream)))
                             1)
                        (EQL (BIN16 stream)
                             highhalf)
                        (EQL (BIN16 stream)
                             lowhalf))
                   then (LET ((version (BIN16 stream)))
                             (SETQ name (CONCAT (PACKC (to (BIN stream) collect (BIN stream)))
                                                ";" version]
               (SETFILEPTR stream (PLUS start length)))
         name])
```

(**SCAVENGEDSKDIRECTORY**

```
  [LAMBDA (volName SILENT)                                      ; Edited  8-Jan-87 17:55 by amd
    ;; If your BTree is intact but your directory is smashed, this routine will scavenge your volume by building a new directory which associates all
    ;; fileIDs in the BTree with a gensym filename
    (WITH.MONITOR \LFtopMonitor
        [PROG ((vol (\LFEntryPoint volName))
               DIRECTORY LISPDIRECTORY LISPFILES)
              (if (NOT (\PFPilotVolumeP vol))
                  then (ERROR "Non-pilot volume"))
         ;; Find the file ID's of the Lisp directory and all the Lisp files on the specified volume.
              (SETQ LISPDIRECTORY (\VFMGenerateFileIDs vol tLispDirectory))
```

```
        (SETQ LISPFILES (\VFMGenerateFileIDs vol tLispFile))
    ;; If there are no Lisp files of any sort on the volume, abort
        (if (AND (NULL LISPDIRECTORY)
                 (NULL LISPFILES))
            then (RETURN NIL))
    ;; This block throws away the old directory and builds a new one.  It must be atomic.
        (UNINTERRUPTABLY
            ;; If there is an old directory, get rid of it.
            (\LFPurgeDirectory vol)
            (if (NOT SILENT)
                then (printout NIL "Deleted old directory." T))
            ;; Create a fresh directory
            (if (type? LFDEV (\GETDEVICEFROMNAME 'DSK))
                then (\LFMakeVolumeDirectory vol)
              else (\LFMakeVolumeDirectory vol T)
                   (\LFOpenDevice))
            (\PFDsplyVolumes)
            (if (NOT SILENT)
                then (printout NIL "Created new directory." T))
            ;; For each file in volume file map, enter this fileID into the new directory
            (for fileID in LISPFILES
               do (PROCEED-CASE (PROG ((stream (\LFOpenOldFile (create FileDescriptor
                                                                       fileID _ fileID
                                                                       volNum _ (\PFVolumeNumber vol)
                                                                       type _ tLispFile)
                                                               NIL NIL))
                                       DIRINDEX UNAME NAME&VERSION NAME VERSION)
                                      (SETQ NAME&VERSION (fetch (LeaderPage fileName)
                                                                of (fetch (DLIONSTREAM LEADERPAGE) of stream)))
                                      (SETQ NAME (\LFScavFileName NAME&VERSION))
                                      (SETQ VERSION (\LFScavVersion NAME&VERSION fileID))
                                      (SETQ UNAME (create ExpandedName
                                                          VOLNUM _ (\PFVolumeNumber vol)
                                                          CHARLIST _ NAME
                                                          VERSION _ VERSION))
                                      (SETQ DIRINDEX (\LFFindDirHole stream UNAME (\LFGetDirectory vol)))
                                      (if (NULL DIRINDEX)
                                          then (LISPERROR "HARD DISK ERROR" "Can't rebuild directory"))
                                      (\LFMakeDirEntry stream UNAME (\LFGetDirectory vol)
                                            DIRINDEX)
                                      (if (NOT SILENT)
                                          then (PRINTOUT NIL "Added " (PACKC NAME)
                                                         ";" VERSION " to directory." T)))
                                (NIL NIL :REPORT "Skip this file"))))
    ;; Return the name of the new directory
        (RETURN (PACKFILENAME.STRING 'HOST 'DSK 'DIRECTORY (U-CASE (fetch (LogicalVolumeDescriptor LVlabel)
                                                                      of vol))))
```

(**SCAVENGEVOLUME**

```
  [LAMBDA (volName)                                          (* hts%: " 4-Jul-85 18:30")
```

;;; for backward compatibility

```
    (SCAVENGEDSKDIRECTORY volName])
```

(\LFScavFileName

```
  [LAMBDA (NAME&VERSION)                                     ; Edited 22-Oct-87 16:28 by amd
    ;; Extract the filename part of NAME&VERSION (ignore version number) and return it as a list of charcode
    (PROG ((NAME (for C instring (MKSTRING NAME&VERSION) until (EQL C (CHARCODE ;)) collect C)))
          (RETURN (if [OR (NULL NAME)
                          (for C in NAME thereis (ZEROP (\LFCASEARRAYFETCH C]
                      then ;; If there is an illegal char in the filename, or the filename is the empty string, gin up a random filename
                           (CHCON (CONCAT (GENSYM 'TRASHEDFILENAME)
                                          "."))
                    else ;; Otherwise return the filename found
                         NAME])
```

(\LFScavVersion

```
  [LAMBDA (NAME&VERSION FILEID)                              (* amd "10-Feb-86 16:04")
```

;;; Fetch the version number from NAME&VERSION. If it's garbled (ie, isn't a fixp) use the fileID as a version number instead (the fileID will at least give
;;; the file a unique version number and so avoid version number clashes)

```
    (OR (SMALLP (FILENAMEFIELD NAME&VERSION 'VERSION))
        (SMALLP FILEID)
```

```
        (RAND 1 MAX.SMALLP])

)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \LFtopMonitor)
)
```

;; Volume file map (LFFILEMAP) level stuff

```
(DEFINEQ
```

(\VFMGenerateFileIDs

```
  [LAMBDA (vol desiredType)                                   ; Edited 22-Oct-87 16:28 by amd
    ;; Returns a list of the fileIDs of all the keys in the BTree with type = desiredType
    (WITH.MONITOR \VFMmonitor
        (UNINTERRUPTABLY
            (\VFMContextSet vol)
            (bind (currentKey _ (create Key)) until (PROGN (replace (Key filePage) of currentKey with MAX.FIXP)
                                                           (MESASETQ currentKey (fetch (Interval nextKey)
                                                                                   of (\VFMGet currentKey 0))
                                                                  Key)
                                                           (EQL (fetch (Key fileID) of currentKey)
                                                                \VFMmaxID))
               when (EQL (fetch (Key type) of currentKey)
                         desiredType)
               collect (fetch (Key fileID) of currentKey)))))])

)

(RPAQQ LFPILOTFILECOMS
       (
```

;;; This module (together with its two sub-modules, FILEMAP and ALLOCATIONMAP) define the necessary subset of the Pilot file system. This used to ;;; be contained in the file LFPILOTFILE.

```
        ;; These functions transfer pages to and from the disk
        (FNS \PFGetPhysicalVolumePage)
        (FNS \PFGetLogicalVolumePage \PFPutLogicalVolumePage)
        (FNS \PFGetMarkerPage \PFPutMarkerPage)
        (FNS \PFGetFreePage \PFCreateFreePage)
        (FNS \PFGetAllocationMapPage \PFPutAllocationMapPage)
        (FNS \PFGetFileMapPage \PFPutFileMapPage)
        (FNS \PFGetPage \PFPutPage \PFCreatePage)
        (FNS \PFTransferFilePage)
        (FNS \PFTransferPage)
        [DECLARE%: DONTEVAL@LOAD (P (\LOCKFN '\PFTransferPage]
        (RESOURCES label)
        ;; File Descriptor pool for system files
        (FNS \PFCreateFileDescriptors \PFInitFileDescriptors)
        (GLOBALVARS \PFLogicalVolumeFileD \PFMarkerFileD \PFFreeFileD \PFAllocationMapFileD \PFFileMapFileD)
        (P (\PFCreateFileDescriptors))
        ;; Physical volume interface
        (FNS \PFCreatePhysicalVolume)
        (GLOBALVARS \PhysVolumePage)
        (P (\PFCreatePhysicalVolume))
        ;; Interface to logical volumes,
        (FNS \PFCreateVols \PFInitializeVols \PFGetVols \PFGetVol \PFVolumeNumber)
        (GLOBALVARS \DFSLogicalVolumes \DFSLogicalVolumeHash)
        (P (\PFCreateVols))
        (FNS \PFGetLVPage)
        ;; Pilot integrity
        (FNS \PFVersionOK \PFPilotVolumeP)
        ;; Pilot initialization
        (FNS \PFEnsureInitialized)
        (GLOBALVARS \PFInitialized)
        (INITVARS (\PFInitialized NIL)
               \PFDebugFlag)
        (P (ADDTOVAR \SYSTEMCACHEVARS \PFInitialized))
        (P (\PFEnsureInitialized))
        ;; Root directory management
        (FNS \PFFindDirectoryID \PFInsertDirectoryID \PFRemoveDirectoryID)
        (FNS \PFFindRootDirEntry \PFAddRootDirEntry \PFRemoveRootDirEntry \PFFindRootDirEntryNum
             \PFPatchRootDirEntries)
        (FNS \PFGetRootDirectory \PFPutRootDirectory \PFCreateRootDirectory \PFPurgeRootDirectory)
        (FNS \GetRootDirectoryType \PFPutRootDirectoryType)
        ;; Pilot file management
```

```
        (FNS \PFNewPages \PFTrimHelper \PFFindPageAddr \PFFindFileSize \PFFreeDiskPages \PFRoomForFile
             \PFSaveBuffers)

        ;; Lisp vmem

        (FNS \PFCurrentVol)

        ;; Display stub;  real volume display stuff is implemented on a library package called VOLUMEDISPLAY.

        (FNS \PFDsplyVolumes)))
```

;;; This module (together with its two sub-modules, FILEMAP and ALLOCATIONMAP) define the necessary subset of the Pilot file system. This used to ;;; be contained in the file LFPILOTFILE.

;; These functions transfer pages to and from the disk

```
(DEFINEQ

(\PFGetPhysicalVolumePage
  [LAMBDA (buffer)                                                    (* hts%: " 5-Jan-85 16:14")
    (\PFTransferPage 0 buffer 'VRR (create Label])

)

(DEFINEQ

(\PFGetLogicalVolumePage
  [LAMBDA (vol frame)                                                 (* hts%: "28-Nov-84 16:41")

          (* * comment)

    (\PFGetPage (ELT \PFLogicalVolumeFileD (OR (FIXP vol)
                                               (\PFVolumeNumber vol)))
          0 0 frame])

(\PFPutLogicalVolumePage
  [LAMBDA (vol frame)                                                 (* hts%: "28-Nov-84 16:41")

          (* * comment)

    (\PFPutPage (ELT \PFLogicalVolumeFileD (OR (FIXP vol)
                                               (\PFVolumeNumber vol)))
          0 0 frame])

)

(DEFINEQ

(\PFGetMarkerPage
  [LAMBDA (vol frame)                                                 (* hts%: "29-Nov-84 12:26")

          (* * comment)

    (OR (FIXP vol)
        (SETQ vol (\PFVolumeNumber vol)))
    (\PFGetPage (ELT \PFMarkerFileD vol)
          (IPLUS (LvBasePageAddr vol)
                 (MarkerPageAddr vol))
          (MarkerPageAddr vol)
          frame])

(\PFPutMarkerPage
  [LAMBDA (vol frame)                                                 (* hts%: "29-Nov-84 12:27")

          (* * comment)

    (OR (FIXP vol)
        (SETQ vol (\PFVolumeNumber vol)))
    (\PFPutPage (ELT \PFMarkerFileD vol)
          (IPLUS (LvBasePageAddr vol)
                 (MarkerPageAddr vol))
          (MarkerPageAddr vol)
          frame])

)

(DEFINEQ

(\PFGetFreePage
  [LAMBDA (vol volumePageNumber frame runLength noBreak)              (* edited%: " 4-Jul-85 04:34")

;;; Read a free page (or bunch of them) presumably to check their labels.

    (\PFGetPage (ELT \PFFreeFileD (OR (FIXP vol)
                                      (\PFVolumeNumber vol)))
          volumePageNumber volumePageNumber frame runLength noBreak])
```

(\PFCreateFreePage

```
  [LAMBDA (vol volumePageNumber frame runLength noBreak)        (* edited%: " 3-Jul-85 22:10")
```

;;; Write a label on a page that says its free

```
    (\PFCreatePage (ELT \PFFreeFileD (OR (FIXP vol)
                                         (\PFVolumeNumber vol)))
           volumePageNumber volumePageNumber frame runLength noBreak])

)

(DEFINEQ
```

(\PFGetAllocationMapPage

```
  [LAMBDA (vol volumePageNumber frame)                          (* hts%: "29-Nov-84 12:39")

          (* * comment)

    (\PFGetPage (ELT \PFAllocationMapFileD (OR (FIXP vol)
                                               (\PFVolumeNumber vol)))
           volumePageNumber volumePageNumber frame])
```

(\PFPutAllocationMapPage

```
  [LAMBDA (vol volumePageNumber frame)                          (* hts%: "29-Nov-84 12:29")

          (* * comment)

    (OR (FIXP vol)
        (SETQ vol (\PFVolumeNumber vol)))
    (\PFPutPage (ELT \PFAllocationMapFileD vol)
           volumePageNumber volumePageNumber frame])

)

(DEFINEQ
```

(\PFGetFileMapPage

```
  [LAMBDA (vol volumePageNumber frame)                          (* hts%: "29-Nov-84 12:32")

          (* * comment)

    (OR (FIXP vol)
        (SETQ vol (\PFVolumeNumber vol)))
    (\PFGetPage (ELT \PFFileMapFileD vol)
           volumePageNumber volumePageNumber frame])
```

(\PFPutFileMapPage

```
  [LAMBDA (vol volumePageNumber frame)                          (* hts%: "29-Nov-84 12:32")

          (* * comment)

    (OR (FIXP vol)
        (SETQ vol (\PFVolumeNumber vol)))
    (\PFPutPage (ELT \PFFileMapFileD vol)
           volumePageNumber volumePageNumber frame])

)

(DEFINEQ
```

(\PFGetPage

```
  [LAMBDA (file filePageNumber volumePageNumber frame runLength noBreak)
                                                                (* edited%: " 4-Jul-85 03:45")
```

;;; file: FileDescriptor, filePageNumber: FIXP, volumePageNumber: FIXP, frame: Page

;;; Reads a page from the disk into frame

```
    (\PFTransferFilePage file filePageNumber volumePageNumber frame 'VVR runLength noBreak])
```

(\PFPutPage

```
  [LAMBDA (file filePageNumber volumePageNumber frame)          (* hts%: "28-Nov-84 15:10")
```

;;; file: FileDescriptor, filePageNumber: FIXP, volumePageNumber: FIXP, frame: Page

;;; Writes the page in frame onto the disk and checks the label of the disk page

```
    (\PFTransferFilePage file filePageNumber volumePageNumber frame 'VVW])
```

(\PFCreatePage

```
  [LAMBDA (file filePageNumber volumePageNumber frame runLength noBreak)
                                                                (* edited%: " 3-Jul-85 22:04")
```

```
;;; file: FileDescriptor, filePageNumber: FIXP, volumePageNumber: FIXP, frame: Page

;;; Writes the page in frame onto the disk and writes a new label for it

    (\PFTransferFilePage file filePageNumber volumePageNumber frame 'VWW runLength noBreak])

)

(DEFINEQ
```

(\PFTransferFilePage

```
  [LAMBDA (file filePageNumber volumePageNumber frame operation runLength noBreak)
                                                                    ; Edited 16-Apr-87 19:53 by amd
    ;; file: FileDescriptor, filePageNumber: FIXP, volumePageNumber: FIXP, frame: Page, operation: (VVR VVW VWW)
    ;;
    ;; Transfers a page to or from the disk. This function, unlike \PFTransferPage, deals in file- and volume-relative page numbers. It builds the correct
    ;; label to be used for the transfer. NB: The only multi-page transfers occur during file allocation and deallocation. In these cases, FRAME is a junk
    ;; page that will get written to every page being processed.
    (SETQ runLength (OR runLength 1))
    ;; Break up the run into chunks of at most 128 pages for the Daybreak. DiskHeadDove$InitIOCB will not create an IOCB with a run length longer
    ;; than that for some reason. This is the most convenient place to put this patch for now -- ideally the driver should be changed to handle this. The
    ;; DLion is not adversely affected, since it does cylinder-crossing runs one page at a time anyway.
    (for PAGE-OFFSET from 0 by 128 as PAGES-LEFT from runLength by -128 while (IGREATERP PAGES-LEFT 0)
       do (WITH-RESOURCE label (if (FIXP (fetch (FileDescriptor fileID) of file))
                                   then (replace (Label fileID) of label with (fetch (FileDescriptor fileID)
                                                                                 of file))
                                   else ;; Logical volume pages, marker pages, and physical volume pages have a 5-word volume ID for their
                                        ;; fileID in a label.  This is essentially a loophole to get around the normal declaration of the Label
                                        ;; datatype, which expects a 2-word ID
                                        (MESASETQ label (fetch (FileDescriptor fileID) of file)
                                                VolumeID))
                           (replace (Label attributesInAllPages) of label with (fetch (FileDescriptor type) of file))
                           (replace (Label filePage) of label with (IPLUS filePageNumber PAGE-OFFSET))
                           (\PFTransferPage (IPLUS (LvBasePageAddr (fetch (FileDescriptor volNum) of file))
                                                   volumePageNumber PAGE-OFFSET)
                                  frame operation label (MIN PAGES-LEFT 128)
                                  noBreak)))
    NIL])

)

(DEFINEQ
```

(\PFTransferPage

```
  [LAMBDA (absoluteDiskAddress buffer mode label runLength noBreak)
                                                                    ; Edited 16-Apr-87 19:48 by amd
    ;; Transfers a run of pages to or from the disk. This routine, unlike \PFTransferFilePage, deals in virtual disk addresses and expects the label to be
    ;; set up correctly.
    (if (NULL runLength)
        then (SETQ runLength 1))
    ;; Make sure everything is swapped in to prevent page faulting in low-level disk routines.  In addition, buffer must be dirty for disk microcode to treat
    ;; it right.
    (SwapIn&Dirty buffer)
    (SwapIn&Dirty label)
    ;; Do the transfer
    (LET (DOB STATUS)
         (UNINTERRUPTABLY
             (SETQ DOB (\DL.OBTAINNEWDOB))
             (with DLION.DOB DOB (SETQ DISKADDRESS absoluteDiskAddress)
                   (SETQ BUFFER buffer)
                   (SETQ RUNLENGTH runLength)
                   (SETQ LABEL label)
                   (SETQ MODE mode))
             (\MISCAPPLY* (FUNCTION \DLDISK.EXECUTE)
                    DOB)
             (SETQ STATUS (fetch (DLION.DOB STATUS) of DOB))
             (SETQ DOB (\DL.RELEASEDOB DOB)))
         (if (AND (NOT noBreak)
                  (NEQ STATUS 'OK))
             then (DiskError "HARD DISK ERROR" STATUS))
         STATUS])

)

(DECLARE%: DONTEVAL@LOAD

(\LOCKFN '\PFTransferPage)
)
```

```
(DECLARE%: EVAL@COMPILE

[PUTDEF 'label 'RESOURCES '(NEW (create Label)
                                GET
                                (CL:LOCALLY (DECLARE (GLOBALVARS \label.GLOBALRESOURCE))
                                       (if \label.GLOBALRESOURCE then (PROG1 \label.GLOBALRESOURCE
                                                                             (\CLEARWORDS \label.GLOBALRESOURCE
                                                                                    (MESASIZE Label))
                                                                             (SETQ \label.GLOBALRESOURCE NIL))
                                          else
                                          (NEWRESOURCE label]
)
```

;; File Descriptor pool for system files

```
(DEFINEQ
```

(\PFCreateFileDescriptors

```
  [LAMBDA NIL                                                    (* hts%: " 7-Jan-85 15:15")
```

;;; Sets up the file descriptors for system files. Should be run at load time (or at least the first time you wake up on a dlion, and before running ;;; \PFInitFileDescriptors)

```
    (if [NOT (AND (BOUNDP '\PFLogicalVolumeFileD)
                  (BOUNDP '\PFMarkerFileD)
                  (BOUNDP '\PFFreeFileD)
                  (BOUNDP '\PFAllocationMapFileD)
                  (BOUNDP '\PFFileMapFileD]
        then (SETQ \PFInitialized NIL)
             ;; Logical volume descriptors
             (SETQ \PFLogicalVolumeFileD (ARRAY maxLogicalVolumes NIL NIL 0))
             (for volNum from 0 to (SUB1 maxLogicalVolumes)
                do (SETA \PFLogicalVolumeFileD volNum (create FileDescriptor
                                                              volNum _ volNum
                                                              type _ tLogicalVolumeRootPage
                                                              size _ 1)))
             ;; Marker pages
             (SETQ \PFMarkerFileD (ARRAY maxLogicalVolumes NIL NIL 0))
             (for volNum from 0 to (SUB1 maxLogicalVolumes)
                do (SETA \PFMarkerFileD volNum (create FileDescriptor
                                                       volNum _ volNum
                                                       type _ tSubVolumeMarkerPage
                                                       size _ 1)))
             ;; Free pages
             (SETQ \PFFreeFileD (ARRAY maxLogicalVolumes NIL NIL 0))
             (for volNum from 0 to (SUB1 maxLogicalVolumes)
                do (SETA \PFFreeFileD volNum (create FileDescriptor
                                                     fileID _ tFreePage
                                                     volNum _ volNum
                                                     type _ tFreePage)))
             ;; Volume allocation map pages
             (SETQ \PFAllocationMapFileD (ARRAY maxLogicalVolumes NIL NIL 0))
             (for volNum from 0 to (SUB1 maxLogicalVolumes)
                do (SETA \PFAllocationMapFileD volNum (create FileDescriptor
                                                              fileID _ tVolumeAllocationMap
                                                              volNum _ volNum
                                                              type _ tVolumeAllocationMap)))
             ;; Volume file map pages
             (SETQ \PFFileMapFileD (ARRAY maxLogicalVolumes NIL NIL 0))
             (for volNum from 0 to (SUB1 maxLogicalVolumes)
                do (SETA \PFFileMapFileD volNum (create FileDescriptor
                                                        fileID _ tVolumeFileMap
                                                        volNum _ volNum
                                                        type _ tVolumeFileMap])
```

(\PFInitFileDescriptors

```
  [LAMBDA NIL                                                    (* hts%: "30-Nov-84 13:44")
```

;;; Fills in the fileID for the system file descriptors whose fileID changes depending on what disk you're running on. This routine should be run every time ;;; you wake up on a DLion, but run after you've read in the physical volume page.

```
    (PROG [(lastVolNum (SUB1 (fetch (PhysicalVolumeDescriptor subVolumeCount) of \PhysVolumePage]
     ;; Logical volume descriptors
          (for volNum from 0 to lastVolNum do (replace (FileDescriptor fileID) of (ELT \PFLogicalVolumeFileD volNum
                                                                                       )
                                                      with (MESASETQ (create VolumeID)
                                                                     (fetch (SubVolumeDesc lvID)
                                                                        of (FMESAELT (fetch (PhysicalVolumeDescriptor
                                                                                                 subVolumes)
                                                                                        of \PhysVolumePage)
```

```
                                                                    SubVolumeArray volNum))
                                                            VolumeID)))
      ;; Marker pages
            (for volNum from 0 to lastVolNum do (replace (FileDescriptor fileID) of (ELT \PFMarkerFileD volNum)
                                                   with (MESASETQ (create VolumeID)
                                                                  (fetch (PhysicalVolumeDescriptor subVolumeMarkerID)
                                                                     of \PhysVolumePage)
                                                                  VolumeID])

)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \PFLogicalVolumeFileD \PFMarkerFileD \PFFreeFileD \PFAllocationMapFileD \PFFileMapFileD)
)

(\PFCreateFileDescriptors)

;; Physical volume interface

(DEFINEQ

(\PFCreatePhysicalVolume
  [LAMBDA NIL                                                       (* hts%: " 7-Jan-85 15:15")
    (if (NOT (AND (BOUNDP '\PhysVolumePage)
                  (type? PhysicalVolumeDescriptor \PhysVolumePage)))
        then (SETQ \PFInitialized NIL)
             (SETQ \PhysVolumePage (create PhysicalVolumeDescriptor)))
    NIL])

)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \PhysVolumePage)
)

(\PFCreatePhysicalVolume)

;; Interface to logical volumes,

(DEFINEQ

(\PFCreateVols
  [LAMBDA NIL                                                       ; Edited 22-Oct-87 16:30 by amd
    ;; Creates an array of logical volume page frames.  Also creates a hash table which maps logical volumes descriptors onto volume numbers.  Both
    ;; these data structures share logical volume page frames, so only one (the array) need be updated.  The conditional ensures that loading a new
    ;; version of the file system will not smash the logical volume information, unless the data structures are incompatible.
    (if (NOT (AND (BOUNDP '\DFSLogicalVolumes)
                  (type? ARRAYP \DFSLogicalVolumes)
                  (ZEROP (ARRAYORIG \DFSLogicalVolumes))
                  (EQL maxLogicalVolumes (ARRAYSIZE \DFSLogicalVolumes))
                  (BOUNDP '\DFSLogicalVolumeHash)
                  (HASHARRAYP \DFSLogicalVolumeHash)))
        then (SETQ \DFSLogicalVolumes (ARRAY maxLogicalVolumes NIL NIL 0))
             (SETQ \DFSLogicalVolumeHash (HASHARRAY maxLogicalVolumes))
             (bind vol for volNum from 0 to (SUB1 maxLogicalVolumes) do (SETQ vol (create LogicalVolumeDescriptor))
                                                                        (SETA \DFSLogicalVolumes volNum vol)
                                                                        (PUTHASH vol volNum \DFSLogicalVolumeHash)
                   )
             (SETQ \PFInitialized NIL))
    NIL])

(\PFInitializeVols
  [LAMBDA NIL                                                       (* hts%: "29-Nov-84 12:19")
    (for volNum from 0 to (SUB1 (fetch (PhysicalVolumeDescriptor subVolumeCount) of \PhysVolumePage))
       do (\PFGetLogicalVolumePage volNum (\PFGetVol volNum])

(\PFGetVols
  [LAMBDA NIL                                                       (* hts%: "11-Oct-84 17:19")
    (for volNum from 0 to (SUB1 (fetch (PhysicalVolumeDescriptor subVolumeCount) of \PhysVolumePage))
       collect (\PFGetVol volNum])

(\PFGetVol
  [LAMBDA (volNum)                                                  (* hts%: "11-Oct-84 15:12")
    (ELT \DFSLogicalVolumes volNum])

(\PFVolumeNumber
  [LAMBDA (vol)                                                     (* hts%: "26-Nov-84 11:52")
```

```
;;; vol: LogicalVolumeDescriptor;  RETURNS: FIXP in 0..9
    ;; Converts vol into a logical volume number, becuase the page reading and writing routines expect a logical volume number rather than the logical
    ;; volume itself.
    (GETHASH vol \DFSLogicalVolumeHash])

)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \DFSLogicalVolumes \DFSLogicalVolumeHash)
)

(\PFCreateVols)

(DEFINEQ

(\PFGetLVPage
  [LAMBDA (lvName)                                                    ; Edited  8-Jan-87 17:49 by amd
    ;; Returns the logical volume page for the volume whose name is lvName.  Returns NIL if there is no such volume.
    (for vol in (\PFGetVols) thereis (STRING-EQUAL lvName (fetch (LogicalVolumeDescriptor LVlabel) of vol])

)

;; Pilot integrity

(DEFINEQ

(\PFVersionOK
  [LAMBDA NIL                                                         (* hts%: " 6-Jan-85 18:49")

;;; Checks to see that the disk you are attempting to run on is partitioned in a way the file system can understand

    (for vol in (\PFGetVols) always (EQ pilotVersion (fetch (LogicalVolumeDescriptor version) of vol])


(\PFPilotVolumeP
  [LAMBDA (vol)                                                       (* amd "10-Feb-86 16:04")

;;; Tells whether the volume in question is a pilot or non-pilot volume.

;;; any volume which is not of type non-Pilot is considered a Pilot volume <normal, debugger, debuggerdebugger, etc.>

    (NEQ (fetch (LogicalVolumeDescriptor type) of vol)
         nonPilotVolume])

)

;; Pilot initialization

(DEFINEQ

(\PFEnsureInitialized
  [LAMBDA (FORCEINITIALIZATION)                                       (* amd "10-Feb-86 16:04")

;;; Caches enough of the state of the disk so that the file system can run.  Doesn't access the disk unless necessary.

    (SELECTQ (MACHINETYPE)
        ((DANDELION DOVE)
            (if (OR FORCEINITIALIZATION (NOT \PFInitialized))
                then ;; initialize physical volume page cache
                     (\PFGetPhysicalVolumePage \PhysVolumePage)
                     ;; Use physical volume page to set up disk-specific system file descriptors (for logical volume pages and marker pages)
                     (\PFInitFileDescriptors)
                     ;; initialize logical volume page cache;
                     (\PFInitializeVols)
                     (if (\PFVersionOK)
                         then ;; Initialize volume file map and volume allocation map
                              (\VAMInit)
                              (\VFMInit)
                              ;; Note that this routine has been run
                              (SETQ \PFInitialized T)
                              (\PFDsplyVolumes)
                              T
                       else (SETQ \PFInitialized NIL))
              else (SETQ \PFInitialized T)))
        (SETQ \PFInitialized T])

)
```

```
(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \PFInitialized)
)

(RPAQ? \PFInitialized NIL)

(RPAQ? \PFDebugFlag NIL)

(ADDTOVAR \SYSTEMCACHEVARS \PFInitialized)

(\PFEnsureInitialized)

;; Root directory management

(DEFINEQ

(\PFFindDirectoryID
  [LAMBDA (vol type)                                                    (* hts%: "18-Dec-84 16:45")

;;; If on vol there is a directory associated with the specified file type, returns the fileID associated with that directory;  else returns NIL

    (PROG ((rootDir (create RootDirectory)))
          (RETURN (if (\PFGetRootDirectory vol rootDir)
                      then (\PFFindRootDirEntry type rootDir)
                    else NIL])

(\PFInsertDirectoryID
  [LAMBDA (vol type directoryID)                                        ; Edited  9-Jan-87 19:05 by amd
    (PROG ((rootDir (create RootDirectory)))
          (if (NOT (\PFGetRootDirectory vol rootDir))
              then (\PFCreateRootDirectory vol rootDir))
          (\PFAddRootDirEntry type directoryID rootDir)
          (\PFPutRootDirectory vol rootDir))
    ;; Make sure it gets written out!
    (\PFSaveBuffers vol])

(\PFRemoveDirectoryID
  [LAMBDA (vol type)                                                    ; Edited 22-Oct-87 16:31 by amd
    (PROG ((rootDir (create RootDirectory)))
          (if (\PFGetRootDirectory vol rootDir)
              then (if (ILEQ (fetch (RootDirectory countEntries) of rootDir)
                             1)
                       then (\PFPurgeRootDirectory vol rootDir)
                     else (\PFRemoveRootDirEntry type rootDir)
                          (\PFPutRootDirectory vol rootDir])

)

(DEFINEQ

(\PFFindRootDirEntry
  [LAMBDA (type rootDir)                                                (* hts%: " 4-Jul-85 18:58")

;;; look through registered directories to find the desired one.  Stored as an array of (type directoryFileID) pairs.

    (\PFPatchRootDirEntries type rootDir)
    (LET ((entryNum (\PFFindRootDirEntryNum type rootDir)))
         (AND entryNum (fetch (RootDirEntry file) of (MESAELT (fetch (RootDirectory entries) of rootDir)
                                                              RootDirEntryArray entryNum])

(\PFAddRootDirEntry
  [LAMBDA (type directoryID rootDir)                                    (* hts%: " 4-Jul-85 18:41")

;;; Add specified (type directoryID) pair

    (UNINTERRUPTABLY
        (PROG ((entryNum (fetch (RootDirectory countEntries) of rootDir)))
              (MESASETA (fetch (RootDirectory entries) of rootDir)
                     RootDirEntryArray entryNum (create RootDirEntry
                                                        type _ type
                                                        file _ directoryID))
              (replace (RootDirectory countEntries) of rootDir with (ADD1 entryNum))))])

(\PFRemoveRootDirEntry
  [LAMBDA (type rootDir)                                                (* hts%: " 4-Jul-85 18:58")

          (* * comment)

    (UNINTERRUPTABLY
        (PROG ((nuke (\PFFindRootDirEntryNum type rootDir)))
              (if nuke
```

```
                then (bind (directories _ (fetch (RootDirectory entries) of rootDir)) for entryNum
                        from (ADD1 nuke) to (fetch (RootDirectory countEntries) of rootDir)
                        do (MESASETA directories RootDirEntryArray (SUB1 entryNum)
                                  (MESAELT directories RootDirEntryArray entryNum)))
                     (add (fetch (RootDirectory countEntries) of rootDir)
                          -1)))])
```

(\PFFindRootDirEntryNum

```
  [LAMBDA (type rootDir)                                    ; Edited 22-Oct-87 16:32 by amd
```

;; look through registered directories to find the desired one. Stored as an array of (type directoryFileID) pairs.

```
    (bind (directories _ (fetch (RootDirectory entries) of rootDir)) for entryNum from 0
       to (SUB1 (fetch (RootDirectory countEntries) of rootDir)) thereis (EQL (fetch (RootDirEntry type)
                                                                                 of (MESAELT directories
                                                                                           RootDirEntryArray
                                                                                           entryNum))
                                                                               type])
```

(\PFPatchRootDirEntries

```
  [LAMBDA (type rootDir)                                    (* hts%: " 4-Jul-85 18:58")
```

;;; Quietly patch up an off-by-one that was in Intermezzo.

```
    (\PFRemoveRootDirEntry 0 rootDir)
    (add (fetch (RootDirectory countEntries) of rootDir)
         1])

)

(DEFINEQ
```

(\PFGetRootDirectory

```
  [LAMBDA (vol rootDir)                                     (* hts%: " 5-Jan-85 16:26")
```

;;; Reads in and returns the root directory for the specified volume, provided that it is there; else returns NIL

```
    (if (NEQ (\GetRootDirectoryType vol)
             tRootDirectory)
        then NIL
      else (PROG ((fileD (create FileDescriptor
                                 fileID _ tRootDirectory
                                 volNum _ (\PFVolumeNumber vol)
                                 type _ tRootDirectory
                                 size _ 1))
                  where)
```

;; find location of root directory page

```
           (SETQ where (\VFMGetPageGroup vol fileD 0))
           (OR where (RETURN NIL))
```

;; read in root directory page

```
           (\PFGetPage fileD 0 (fetch (PageGroup volumePage) of where)
                  rootDir)
           (RETURN T])
```

(\PFPutRootDirectory

```
  [LAMBDA (vol rootDir)                                     (* edited%: "20-Jan-85 16:01")
```

(* * comment)

```
    (PROG ((fileD (create FileDescriptor
                          fileID _ tRootDirectory
                          volNum _ (\PFVolumeNumber vol)
                          type _ tRootDirectory
                          size _ 1))
           where)
```

;; find location of root directory page

```
          (SETQ where (\VFMGetPageGroup vol fileD 0))
          (OR where (DiskError "HARD DISK ERROR" "Can't find volume root directory"))
```

;; read in root directory page

```
          (\PFPutPage fileD 0 (fetch (PageGroup volumePage) of where)
                 rootDir])
```

(\PFCreateRootDirectory

```
  [LAMBDA (vol rootDir)                                     (* hts%: " 9-Aug-85 12:25")
```

(* * comment)

```
    (UNINTERRUPTABLY
        (PROG ((fileD (create FileDescriptor
                              fileID _ tRootDirectory
```

```
                              volNum _ (\PFVolumeNumber vol)
                              type _ tRootDirectory
                              size _ 0)))
                    (OR (\PFNewPages vol fileD (create PageGroup
                                                       filePage _ 0
                                                       nextFilePage _ 1))
                        (DiskError "FILE SYSTEM RESOURCES EXCEEDED"))
                    (\PFPutRootDirectory vol rootDir)
                    (\PFPutRootDirectoryType vol tRootDirectory))])
```

(\PFPurgeRootDirectory

```
  [LAMBDA (vol rootDir)                                    (* hts%: " 5-Jan-85 16:15")

          (* * comment)

    (UNINTERRUPTABLY
        (PROG ((fileD (create FileDescriptor
                              fileID _ tRootDirectory
                              volNum _ (\PFVolumeNumber vol)
                              type _ tRootDirectory
                              size _ 1)))
              (\PFPutRootDirectoryType vol tUnassigned)
              (\PFTrimHelper vol fileD 0)))])

)

(DEFINEQ
```

(\GetRootDirectoryType

```
  [LAMBDA (vol)                                            (* hts%: "18-Dec-84 21:55")

          (* * comment)

    (fetch (LogicalVolumeDescriptor volumeRootDirectory) of vol])
```

(\PFPutRootDirectoryType

```
  [LAMBDA (vol directoryID)                                (* hts%: "18-Dec-84 19:16")

          (* * comment)

    (replace (LogicalVolumeDescriptor volumeRootDirectory) of vol with directoryID)
    (\PFPutLogicalVolumePage vol vol)
    (PROG ((markerPage (create SubVolumeMarkerPage)))
          (\PFGetMarkerPage vol markerPage)
          (replace (LogicalSubVolumeMarker volumeRootDirectory) of markerPage with directoryID)
          (\PFPutMarkerPage vol markerPage])

)

;; Pilot file management

(DEFINEQ
```

(\PFNewPages

```
  [LAMBDA (vol file group)                                 ; Edited 22-Oct-87 16:32 by amd
    ;; Allocates the specified group of pages for file and records them in the volume file map.  Returns file if successful, NIL otherwise.
    (bind (startSize _ (fetch (FileDescriptor size) of file))
          (currentGroup _ (create PageGroup)) until (EQL (fetch (FileDescriptor size) of file)
                                                         (fetch (PageGroup nextFilePage) of group))
      do
        ;; Build the group to attempt to allocate next
        (replace (PageGroup filePage) of currentGroup with (fetch (FileDescriptor size) of file))
        (replace (PageGroup volumePage) of currentGroup with 0)
        (replace (PageGroup nextFilePage) of currentGroup with (fetch (PageGroup nextFilePage) of group))
        ;; Allocate as many pages of the desired group as possible
        (if (NOT (\VAMAllocPageGroup vol file currentGroup))
            then (\PFTrimHelper vol file startSize)
                 (RETURN NIL))
        ;; Stick the newly allocated group into the volume file map BTree
        (\VFMInsertPageGroup vol file currentGroup)
        ;; Record the newly-increased size of the file
        (replace (FileDescriptor size) of file with (fetch (PageGroup nextFilePage) of currentGroup))
        (BLOCK)
      finally (\PFDsplyVolumes)
              (RETURN file])
```

(\PFTrimHelper

```
  [LAMBDA (vol filePtr targetFileSize)                     ; Edited 22-Oct-87 16:33 by amd
    ;; Shortens or deletes a file by taking entries out of the BTree and out of the allocation map Removes the pages of the file between targetFileSize &
    ;; actualFileSize
```

```
    (if (NOT (EQL targetFileSize (fetch (FileDescriptor size) of filePtr)))
        then ;; Bear trap:
             (if (AND \PFDebugFlag (IGREATERP targetFileSize (fetch (FileDescriptor size) of filePtr)))
                 then (LET ((\INTERRUPTABLE T))
                           (HELP "\PFTrimHelper asked to grow file")))
             (bind (group _ (create PageGroup
                                    filePage _ targetFileSize
                                    volumePage _ nullVolumePage
                                    nextFilePage _ (fetch (FileDescriptor size) of filePtr)))
                until (PROGN (\VFMDeletePageGroup vol filePtr group)
                             (\VAMFreePageGroup vol filePtr group)
                             (replace (FileDescriptor size) of filePtr with (fetch (PageGroup filePage) of group))
                             (if (ZEROP (fetch (PageGroup filePage) of group))
                                 then (replace (PageGroup nextFilePage) of group with 0)
                                      (\VFMDeletePageGroup vol filePtr group)
                                      (\VAMFreePageGroup vol filePtr group)
                                      T
                               else (EQL (fetch (PageGroup filePage) of group)
                                         targetFileSize)))
                do (replace (PageGroup nextFilePage) of group with (fetch (PageGroup filePage) of group))
                   (replace (PageGroup filePage) of group with targetFileSize)
                   (BLOCK))
             (\PFDsplyVolumes])
```

(\PFFindPageAddr

```
  [LAMBDA (file filePage)                                   ; Edited 22-Oct-87 16:34 by amd
```

```
    ;; Tells where page filePage of file is located on the disk.  Caches the last pageGroup for the file
    (PROG ((PAGEGROUP (fetch (FileDescriptor PAGEGROUP) of file)))
          (if (OR (NOT (FIXP PAGEGROUP))
                  (ILESSP filePage (fetch (PageGroup filePage) of PAGEGROUP))
                  (IGEQ filePage (fetch (PageGroup nextFilePage) of PAGEGROUP)))
              then ;; Page group we are after is not in cache;  we will have to look it up in the volume file map
                   (SETQ PAGEGROUP (\VFMGetPageGroup (\PFGetVol (fetch (FileDescriptor volNum) of file))
                                                     file filePage))
                   (OR [AND PAGEGROUP (NOT (ZEROP (fetch (PageGroup volumePage) of PAGEGROUP]
                       (DiskError "HARD DISK ERROR" "Can't find file page"))
                   (replace (FileDescriptor PAGEGROUP) of file with PAGEGROUP))
          (RETURN (IPLUS (fetch (PageGroup volumePage) of PAGEGROUP)
                         filePage
                         (IMINUS (fetch (PageGroup filePage) of PAGEGROUP])
```

(\PFFindFileSize

```
  [LAMBDA (file)                                            (* amd "10-Feb-86 16:04")
```

;;; Finds the number of pages in the specified file, as recorded in the volume file map.

```
    (fetch (PageGroup filePage) of (\VFMGetPageGroup (\PFGetVol (fetch (FileDescriptor volNum) of file))
                                                     file MAX.FIXP])
```

(\PFFreeDiskPages

```
  [LAMBDA (vol recompute)                                   (* amd "10-Feb-86 16:04")
```

;;; Returns the free page count for the specified volume.

```
    (if recompute
        then (\VAMRecomputeFreePageCount vol)
             (\PFDsplyVolumes))
    (fetch (LogicalVolumeDescriptor freePageCount) of vol])
```

(\PFRoomForFile

```
  [LAMBDA (vol filePtr groupPtr)                            ; Edited 22-Oct-87 16:35 by amd
```

```
    ;; Returns T iff there is room for the specified file on the specified volume.  Formula is the same as Pilot uses;  it is a little more conservative than
    ;; necessary.  The -5 is the maximum number of file map pages that could split;  I don't know what the 15/16th's is about.
    (LEQ (DIFFERENCE (fetch (PageGroup nextFilePage) of groupPtr)
                     (fetch (PageGroup filePage) of groupPtr))
         (if (EQL tVolumeFileMap (fetch (FileDescriptor type) of filePtr))
             then (\PFFreeDiskPages vol)
           else (IDIFFERENCE (IQUOTIENT (ITIMES (\PFFreeDiskPages vol)
                                                15)
                                        16)
                             5])
```

(\PFSaveBuffers

```
  [LAMBDA (VOL)                                             (* amd "10-Feb-86 16:04")
```

;;; Saves out dirty buffers.

```
    (\PFPutLogicalVolumePage VOL VOL)
    (\VAMBufferSave)
    (\VFMSaveBuffer])

)

;; Lisp vmem

(DEFINEQ

(\PFCurrentVol
  [LAMBDA NIL                                                    ; Edited 22-Oct-87 16:36 by amd
    ;; Returns the logical volume page of the volume which contains the currently running virtual memory.  Depends on booting from physical volume
    ;; boot pointers.
    (for vol in (\PFGetVols) thereis (EQL [fetch (DiskFileID da) of (FMESAELT (fetch (PhysicalVolumeDescriptor
                                                                                       bootingInfo)
                                                                                of \PhysVolumePage)
                                                                        PVBootFiles
                                                                        (SELECTQ (MACHINETYPE)
                                                                             (DANDELION hardMicrocode)
                                                                             (DOVE bftGerm)
                                                                             (\NOMACHINETYPE]
                                                          (fetch (DiskFileID da) of (FMESAELT (fetch (LogicalVolumeDescriptor
                                                                                                       bootingInfo)
                                                                                                of vol)
                                                                                        LVBootFiles
                                                                                        (SELECTQ (MACHINETYPE)
                                                                                             (DANDELION hardMicrocode)
                                                                                             (DOVE bftGerm)
                                                                                             (\NOMACHINETYPE])

)

;; Display stub;  real volume display stuff is implemented on a library package called VOLUMEDISPLAY.

(DEFINEQ

(\PFDsplyVolumes
  [LAMBDA NIL                                                    (* edited%: " 4-Jul-85 03:14")

;;; Updates the volume display window as necessary.

    (if (DEFINEDP '\DSKDISPLAY.UPDATE)
        then (\DSKDISPLAY.UPDATE])

)

(RPAQQ LFALLOCATIONMAPCOMS
       (

;;; Implements the 1108 file system volume file map.  Very roughly translates the Pilot file VolAllocMapImpl.mesa.  Used to be contained in the separate
;;; file LFALLOCATIONMAP.  Must be loaded after the PILOTFILE module.

        ;; Needed improvement : Restructure interface with FILEIO so that a page can be allocated and written in one fell swoop.  MFile/Pilot have a
        ;; special interface for this.
        (DECLARE%: EVAL@COMPILE DONTCOPY (CONSTANTS (BITSPERPAGE 4096)))
        ;; Public routines
        (FNS \VAMAllocPageGroup \VAMFreePageGroup \VAMInit \VAMRecomputeFreePageCount)
        ;; Private routines:
        (FNS \VAMFilePageNumber \VAMEnoughSpace \VAMFindFreePages \VAMCheckEndOfVol \VAMUpdateVAM
             \VAMAdjustGroup)
        (RESOURCES \DFSVAMpage \DFSVAMjunkPage)
        (GLOBALVARS \VAMmonitor)
        [INITVARS (\VAMmonitor (CREATE.MONITORLOCK 'VAMmonitor]
        ;; buffer management
        (FNS \VAMGetVAMPageFor \VAMBufferInit \VAMBufferSave \VAMMarkBufferDirty)
        (GLOBALVARS \VAMbuffer \VAMbufferVolume \VAMbufferVolumePage \VAMbufferDirty)
        ;; Initialize VAM
        (P (\VAMInit))))

;;; Implements the 1108 file system volume file map.  Very roughly translates the Pilot file VolAllocMapImpl.mesa.  Used to be contained in the separate
;;; file LFALLOCATIONMAP.  Must be loaded after the PILOTFILE module.

;; Needed improvement : Restructure interface with FILEIO so that a page can be allocated and written in one fell swoop.  MFile/Pilot have a special
;; interface for this.

(DECLARE%: EVAL@COMPILE DONTCOPY

(DECLARE%: EVAL@COMPILE
```

```
(RPAQQ BITSPERPAGE 4096)

(CONSTANTS (BITSPERPAGE 4096))
)
)
```

;; Public routines

```
(DEFINEQ
```

(\VAMAllocPageGroup

```
[LAMBDA (vol filePtr groupPtr)                                    ; Edited 22-Oct-87 16:38 by amd
```

;; Allocates as many of the pages in groupPtr as it can in a contiguous run. Modifies groupPtr so the caller can know what pages and how many ;; were allocated

```
(WITH.MONITOR \VAMmonitor
    (UNINTERRUPTABLY
        (LET [(RUNLENGTH (IDIFFERENCE (fetch (PageGroup nextFilePage) of groupPtr)
                                      (fetch (PageGroup filePage) of groupPtr]
             (if (\VAMEnoughSpace vol filePtr RUNLENGTH)
                 then
```

;; Look in the free page bitmap to find a contiguous bunch of free pages and mark them taken in the bitmap.

```
                      (\VAMFindFreePages vol filePtr groupPtr)
                      (SETQ RUNLENGTH (IDIFFERENCE (fetch (PageGroup nextFilePage) of groupPtr)
                                                   (fetch (PageGroup filePage) of groupPtr)))
```

;; Update free page count and lower bound on the logical volume page

```
                      (add (fetch (LogicalVolumeDescriptor freePageCount) of vol)
                           (IMINUS RUNLENGTH))
                      (replace (LogicalVolumeDescriptor lowerBound) of vol with (IPLUS (fetch (PageGroup
                                                                                                volumePage)
                                                                                           of groupPtr)
                                                                                       RUNLENGTH))
```

;; Check all these pages to make sure they are indeed free.

```
                      (WITH-RESOURCE \DFSVAMjunkPage (\PFGetFreePage vol (fetch (PageGroup volumePage)
                                                                            of groupPtr)
                                                                 \DFSVAMjunkPage RUNLENGTH))
```

;; Finally, clear each page and give it a free page label.

```
                      (WITH-RESOURCE \DFSVAMpage (\PFCreatePage filePtr (\VAMFilePageNumber
                                                                           (fetch (FileDescriptor type)
                                                                              of filePtr)
                                                                           (fetch (PageGroup volumePage)
                                                                              of groupPtr)
                                                                           (fetch (PageGroup filePage)
                                                                              of groupPtr))
                                                                (fetch (PageGroup volumePage) of groupPtr)
                                                                \DFSVAMpage RUNLENGTH))
```

;; Return T indicating success.

```
                      T
                 else
```

;; Not enough space on the volume: return NIL to indicate failure.

```
                      (replace (PageGroup nextFilePage) of groupPtr with (fetch (PageGroup filePage) of groupPtr))
                      (replace (PageGroup volumePage) of groupPtr with 0)
                     NIL))))])
```

(\VAMFreePageGroup

```
[LAMBDA (vol filePtr groupPtr)                                    ; Edited 22-Oct-87 16:39 by amd
```

;; Frees each page in groupPtr

```
(WITH.MONITOR \VAMmonitor
    (UNINTERRUPTABLY
        [PROG ((group (\VAMAdjustGroup groupPtr)))                ; Adjust to coincide with Pilot's silly '[0, 0)' convention
```

;; If no pages to free, just return (runlength <= 0 might upset later code)

```
              (if (IGEQ (fetch (PageGroup filePage) of group)
                        (fetch (PageGroup nextFilePage) of group))
                  then (RETURN))
              (LET [(RUNLENGTH (IDIFFERENCE (fetch (PageGroup nextFilePage) of group)
                                            (fetch (PageGroup filePage) of group]
```

;; First check the page labels to make sure all the pages really do belong to the file we are shortening.

```
                   (WITH-RESOURCE \DFSVAMjunkPage (\PFGetPage filePtr (\VAMFilePageNumber (fetch (
                                                                                                FileDescriptor
                                                                                                  type)
                                                                                             of filePtr)
                                                                                      (fetch (PageGroup volumePage)
                                                                                         of group)
                                                                                      (fetch (PageGroup filePage)
                                                                                         of group))
                                                                           (fetch (PageGroup volumePage) of group)
                                                                           \DFSVAMjunkPage RUNLENGTH))
```

;; Then clear each page on the disk and give it a new label saying it is a free page.

```
                    (WITH-RESOURCE \DFSVAMpage (\PFCreateFreePage vol (fetch (PageGroup volumePage) of group)
                                                    \DFSVAMpage RUNLENGTH))
                    ;; Finally mark the pages as free in the free page bitmap.
                    (to RUNLENGTH as volumePageNumber from (fetch (PageGroup volumePage) of group)
                       do (\VAMUpdateVAM vol filePtr volumePageNumber 'free))
                    ;; Update free page count and lower bound on the logical volume page
                    (add (fetch (LogicalVolumeDescriptor freePageCount) of vol)
                         RUNLENGTH))
                (replace (LogicalVolumeDescriptor lowerBound) of vol with (IMIN (fetch (PageGroup volumePage)
                                                                                   of group)
                                                                                (fetch (LogicalVolumeDescriptor
                                                                                        lowerBound)
                                                                                   of vol)))])
```

(\VAMInit

```
  [LAMBDA NIL                                                    (* hts%: " 5-Jan-85 16:18")
```

;;; Initializes or reinitializes the volume allocation map

```
    (WITH.MONITOR \VAMmonitor (\VAMBufferInit])
```

(\VAMRecomputeFreePageCount

```
  [LAMBDA (vol)                                                  (* amd "10-Feb-86 16:04")
```

;;; Recomputes the free page count for each volume from scratch; also resets the lower bound pointer

```
    (WITH.MONITOR \VAMmonitor
        [replace (LogicalVolumeDescriptor freePageCount) of vol
           with (bind (firstFree _ T) for page from 1 to (fetch (LogicalVolumeDescriptor volumeSize) of vol)
                   count (PROG [(free (ZEROP (\VAMUpdateVAM vol NIL page 'read]
                               (if (AND free firstFree)
                                   then (replace (LogicalVolumeDescriptor lowerBound) of vol with page)
                                        (SETQ firstFree NIL))
                               (RETURN free]
        (\PFPutLogicalVolumePage vol vol)
        (fetch (LogicalVolumeDescriptor freePageCount) of vol))])

)

;; Private routines:

(DEFINEQ
```

(\VAMFilePageNumber

```
  [LAMBDA (fileType volumePageNumber filePageNumber)             (* amd "10-Feb-86 16:04")
```

;;; Returns the real file page number

```
    (SELECTC fileType
        (tLispFile filePageNumber)
        (tLispDirectory
            filePageNumber)
        (tVolumeFileMap
            volumePageNumber)
        (tRootDirectory
            0)
        (tDiagnosticMicrocode
            filePageNumber)
        (SHOULDNT])
```

(\VAMEnoughSpace

```
  [LAMBDA (vol filePtr RUNLENGTH)                                ; Edited 22-Oct-87 16:40 by amd
```

;; Tells whether there's enough space left on the specified volume to allocate RUNLENGTH pages. There should always be room for new volume
;; file map pages. For other kinds of files, the '15/16th's - 5' is the criterion the Pilot people chose. It is a little over-conservative. The -5 is the
;; maximum number of btree splits you can have in the file map; I don't know what the '15/16th's' is for.

```
    (OR (EQL tVolumeFileMap (fetch (FileDescriptor type) of filePtr))
        (ILEQ RUNLENGTH (IDIFFERENCE (IQUOTIENT (ITIMES (fetch (LogicalVolumeDescriptor freePageCount)
                                                           of vol)
                                                        15)
                                                16)
                                     5])
```

(\VAMFindFreePages

```
  [LAMBDA (vol filePtr groupPtr)                                 ; Edited 22-Oct-87 16:41 by amd
```

;; Scans page allocation bitmap till it finds a chunk of contiguous free pages to partially satisfy the request. Modifies groupPtr accordingly.

```
    (UNINTERRUPTABLY
        (PROG ((volPage# (fetch (LogicalVolumeDescriptor lowerBound) of vol))
               (filePage# (fetch (PageGroup filePage) of groupPtr)))
```

```
        ;; Find first free page and allocate it.  lowerBound is supposed to be the first free page on the volume
            (until [PROGN (if (IGEQ volPage# (SUB1 (fetch (LogicalVolumeDescriptor volumeSize) of vol)))
                              then (DiskError "HARD DISK ERROR" "FREE PAGE COUNT WRONG"))
                          (ZEROP (\VAMUpdateVAM vol filePtr volPage# 'alloc]
              do (add volPage# 1))
        ;; Note in groupPtr the beginning page of the run we will allocate to this file
            (replace (PageGroup volumePage) of groupPtr with volPage#)
        ;; Keep allocating until either you've allocated enough or you run out of consecutive free pages
            [repeatuntil (PROGN (add volPage# 1)
                                (add filePage# 1)
                                (if (IGEQ volPage# (SUB1 (fetch (LogicalVolumeDescriptor volumeSize) of vol)))
                                    then (DiskError "HARD DISK ERROR" "FREE PAGE COUNT WRONG"))
                                (OR (EQL filePage# (fetch (PageGroup nextFilePage) of groupPtr))
                                    (NEQ 0 (\VAMUpdateVAM vol filePtr volPage# 'alloc]
;;; Note in the PageGroup what the last page allocated actually was, so the caller will know
            (replace (PageGroup nextFilePage) of groupPtr with filePage#)))])
```

(\VAMCheckEndOfVol

```
  [LAMBDA (vol volPage#)                                        (* amd "10-Feb-86 16:04")
```

;;; Checks to make sure you are not about to allocate off the end of the volume.

```
    (if (IGEQ volPage# (SUB1 (fetch (LogicalVolumeDescriptor volumeSize) of vol)))
        then (DiskError "HARD DISK ERROR" "FREE PAGE COUNT WRONG"))
    NIL])
```

(\VAMUpdateVAM

```
  [LAMBDA (vol filePtr page allocOrFree)                        (* hts%: "16-Jan-85 21:08")
```

;;; vol: LogicalVolumeDescriptor, filePtr: FileDescriptor, page: FIXP, allocOrFree: {alloc, free}

;;; RETURNS previous value of allocation map for specified page

;;; Sets (if allocOrFree = alloc) or clears (if allocOrFree = free) the map bit for the specified page

```
    (PROG ((VAMPage# (IQUOTIENT page BITSPERPAGE))
           (VAMWord# (IQUOTIENT (IREMAINDER page BITSPERPAGE)
                                BITSPERWORD))
           (VAMBit# (IREMAINDER page BITSPERWORD))
           VAMPage VAMWord VAMBit result)
          (SETQ VAMPage (\VAMGetVAMPageFor vol VAMPage#))
          (SETQ VAMWord (\GETBASE VAMPage VAMWord#))
          (SETQ VAMBit (MASK.1'S (DIFFERENCE 15 VAMBit#)
                                 1))
          (SETQ result (if (BITTEST VAMWord VAMBit)
                           then 1
                           else 0))
          (SELECTQ allocOrFree
              (alloc (SETQ VAMWord (BITSET VAMWord VAMBit))
                     (\VAMMarkBufferDirty))
              (free (SETQ VAMWord (BITCLEAR VAMWord VAMBit))
                    (\VAMMarkBufferDirty))
              (read)
              (SHOULDNT))
          (\PUTBASE VAMPage VAMWord# VAMWord)
          (RETURN result])
```

(\VAMAdjustGroup

```
  [LAMBDA (groupPtr)                                            ; Edited 22-Oct-87 16:42 by amd
```

```
    ;; Adjust groupPtr to not delete the last page of the file unless it is a separate request for that specific purpose.  This was a silly Pilot convention
    ;; (now obsolete).
    (PROG ((group (create PageGroup using groupPtr)))
          [if (ZEROP (fetch (PageGroup filePage) of group))
              then (if (ZEROP (fetch (PageGroup nextFilePage) of group))
                       then (replace (PageGroup nextFilePage) of group with 1)
                   else (replace (PageGroup filePage) of group with 1)
                        (replace (PageGroup volumePage) of group with (ADD1 (fetch (PageGroup volumePage)
                                                                                   of group]
          (RETURN group])

)

(DECLARE%: EVAL@COMPILE

[PUTDEF '\DFSVAMpage 'RESOURCES '(NEW (create Page]

[PUTDEF '\DFSVAMjunkPage 'RESOURCES '(NEW (create Page]
)
```

```
(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \VAMmonitor)
)

(RPAQ? \VAMmonitor (CREATE.MONITORLOCK 'VAMmonitor))

;; buffer management

(DEFINEQ

(\VAMGetVAMPageFor
  [LAMBDA (vol VAMPage#)                                    ; Edited 22-Oct-87 16:42 by amd
    (PROG ((volumePage (IPLUS (fetch (LogicalVolumeDescriptor vamStart) of vol)
                              VAMPage#)))
          (if (AND (FIXP \VAMbufferVolumePage)
                   (EQ \VAMbufferVolume vol)
                   (EQL \VAMbufferVolumePage volumePage))
              then ;; If the desired VAM page is already read in, just return it
                   (RETURN \VAMbuffer)
            else ;; Otherwise write out the old VAM page if there is one
                 (\VAMBufferSave)
                 (UNINTERRUPTABLY
                     ;; Record what the new page is
                     (SETQ \VAMbufferVolume vol)
                     (SETQ \VAMbufferVolumePage volumePage)
                     ;; and read it in
                     (\PFGetAllocationMapPage \VAMbufferVolume \VAMbufferVolumePage \VAMbuffer))
                 (RETURN \VAMbuffer])

(\VAMBufferInit
  [LAMBDA NIL                                               (* hts%: "16-Jan-85 21:04")

;;; if bufferVolumePage is NIL, GetVAMPageFor will not try to flush an old version of it

    (SETQ \VAMbuffer (create Page))
    (SETQ \VAMbufferVolume)
    (SETQ \VAMbufferVolumePage)
    (SETQ \VAMbufferDirty NIL])

(\VAMBufferSave
  [LAMBDA NIL                                               (* amd "10-Feb-86 18:13")

;;; Flush last VAM page used

    (if (AND (FIXP \VAMbufferVolumePage)
             \VAMbufferDirty)
        then (\PFPutAllocationMapPage \VAMbufferVolume \VAMbufferVolumePage \VAMbuffer)
             (SETQ \VAMbufferDirty NIL])

(\VAMMarkBufferDirty
  [LAMBDA NIL                                               (* hts%: "16-Jan-85 21:02")

;;; Indicate that the buffer VAM page is dirty and will have to be written out.

    (SETQ \VAMbufferDirty T)
    NIL])

)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \VAMbuffer \VAMbufferVolume \VAMbufferVolumePage \VAMbufferDirty)
)

;; Initialize VAM

(\VAMInit)

(RPAQQ LFFILEMAPCOMS
       (

;;; Implements the volume file map, which maps Pilot file ID numbers onto runs of disk pages.  Roughly equivalent to the Pilot file VolFileMapImpl.mesa.
;;; Must be loaded after the PILOTFILE module.  Used to be contained in a separate file called LFFILEMAP.

        (DECLARE%: EVAL@COMPILE DONTCOPY (RECORDS Key Interval Index BufferArray Buffer)
               (RECORDS \BTREEBUF)
               (CONSTANTS (maxReadPtr (DIFFERENCE (MESASIZE Buffer)
                                                  (MESASIZE Index)))
```

```
                    (treeDepth 5))
                (FNS ShowIntervals))
        (INITRECORDS \BTREEBUF)

        ;; Initialization routines

        (FNS \VFMInit)

        ;; The following are public entry points to the volume file map module

        (FNS \VFMDeletePageGroup \VFMGetPageGroup \VFMInsertPageGroup)

        ;; The following are routines internal to the volume file map module.

        (FNS \VFMContextSet \VFMCreateVPage \VFMDelete \VFMDelete1 \VFMDelete2 \VFMFind \VFMFreeVPage \VFMGet
             \VFMGet1 \VFMInsert \VFMInsert1 \VFMLower \VFMMerge \VFMMerge1 \VFMPutNext \VFMReadNext \VFMSplit
             \VFMSplit1)
        (GLOBALVARS \VFMmaxID \VFMmaxKey \VFMnullKey \VFMvolumeHandle \VFMinterval \VFMold \VFMlow \VFMhigh
               \VFMoldPtr \VFMlowPtr \VFMhighPtr \VFMmonitor)

        ;; Buffer management

        (FNS \VFMGetBufferFor \VFMSaveBuffer \VFMClearBuffers \VFMKillBuffer \VFMCorrectBufferP
             \VFMMarkBufferDirty)
        (GLOBALVARS \VFMbufferPool \VFMbufferSize \VFMbuffer \VFMxtraBuffer)
        (INITVARS (\VFMbufferSize 10))

        ;; Interval cache interface

        (FNS \VFMCreateIntervals \VFMClearIntervals \VFMGetInterval \VFMBlankInterval)
        (GLOBALVARS \VFMintervals)

        ;; BLT routine that doesn't stomp on itself for overlapping intervals

        (FNS \VFMSmartBLT)

        ;; Loading initialization

        (FNS \VFMAtLoad)
        (P (\VFMAtLoad))))


;;; Implements the volume file map, which maps Pilot file ID numbers onto runs of disk pages.  Roughly equivalent to the Pilot file VolFileMapImpl.mesa.
;;; Must be loaded after the PILOTFILE module.  Used to be contained in a separate file called LFFILEMAP.

(DECLARE%: EVAL@COMPILE DONTCOPY

(DECLARE%: EVAL@COMPILE

(MESARECORD Key ((fileID SWAPPEDFIXP)
                 (filePage SWAPPEDFIXP)
                 (type WORD)))

(MESARECORD Interval ((key Key)
                      (volumePage SWAPPEDFIXP)
                      (nextKey Key)))

(MESARECORD Index ((key Key)
                   (volumePage SWAPPEDFIXP)))

(MESAARRAY BufferArray [(0 (SUB1 (IQUOTIENT WORDSPERPAGE (MESASIZE Index]
       Index)

(MESARECORD Buffer ((data BufferArray)
                    (used WORD))                                    (* This is the structure for a BTree page)
       (CREATE (create Page))
       (TYPE? (type? Page DATUM)))
)

(DECLARE%: EVAL@COMPILE

(DATATYPE \BTREEBUF ((VOLUME POINTER)
                     (VOLPAGENUM FIXP)
                     (PAGE POINTER)
                     (DIRTY FLAG)))
)

(/DECLAREDATATYPE '\BTREEBUF '(POINTER FIXP POINTER FLAG)
       ;; ---field descriptor list elided by lister---
       '6)

(DECLARE%: EVAL@COMPILE

(RPAQ maxReadPtr (DIFFERENCE (MESASIZE Buffer)
                      (MESASIZE Index)))

(RPAQQ treeDepth 5)

(CONSTANTS (maxReadPtr (DIFFERENCE (MESASIZE Buffer)
                           (MESASIZE Index)))
       (treeDepth 5))
)
```

```
(DEFINEQ

(ShowIntervals
  [LAMBDA (vol)                                                    ; Edited 22-Oct-87 12:04 by amd
    (bind (intervalCache _ (PROGN (\VFMContextSet vol)
                                  (\VFMGetInterval)))
          interval for level from 0 to 4 do (printout T level ":" T "key: ")
                                            (SETQ interval (ELT intervalCache level))
                                            (DISPLAYWORDS (fetch (Interval key) of interval)
                                                   (MESASIZE Key))
                                            (printout T "volumePage: " (fetch (Interval volumePage) of interval)
                                                   T)
                                            (printout T "nextKey: ")
                                            (DISPLAYWORDS (fetch (Interval nextKey) of interval)
                                                   (MESASIZE Key])

)
)

(/DECLAREDATATYPE '\BTREEBUF '(POINTER FIXP POINTER FLAG)
        ;; ---field descriptor list elided by lister---
        '6)

;; Initialization routines

(DEFINEQ

(\VFMInit
  [LAMBDA NIL                                                      (* hts%: " 5-Jan-85 16:29")

;;; Minimally reinitialize the volume file map state variables

    (WITH.MONITOR \VFMmonitor
        (UNINTERRUPTABLY
            ;; Clear out the BTree interval cache
            (\VFMClearIntervals)
            ;; Clear the btree node cache
            (\VFMClearBuffers)))])

)

;; The following are public entry points to the volume file map module

(DEFINEQ

(\VFMDeletePageGroup
  [LAMBDA (vol filePtr groupPtr)                                   ; Edited 22-Oct-87 16:46 by amd
    ;; Deletes all or part of a single page group from the volume file map.  The page group requested to be deleted need not correspond to a single run
    ;; of pages on the disk.  It can be part of a single run of pages or stretch over several runs of pages.  In particular it is possible to delete a page or
    ;; pages out of the middle of a run of pages (the scavenger uses this capability).  The actual page group deleted is returned in the group pointed to
    ;; by GroupPtr.  Thus GroupPtr points to a modifiable hint.  Care must be taken by the caller to insure that the page group to be deleted exists.  If it
    ;; doesn't, Bug (pageGroupNotFound) is raised.  This procedure implements the following funny features:

    ;; 1.0 If the page group to be deleted includes parts of more than one run of pages on the disk, only the last run (or that part of the last run
    ;; requested to be deleted) will be deleted.

    ;; 2.0 If the page group to be deleted is the last page group left for the file and includes page zero of the file and at least one following page, page
    ;; zero will not be deleted.  This is a special case that facilitates shrinking a file to a zero-length file.  VolAllocMapImpl has special case code in
    ;; FreePageGroup for this also.  You can delete this last page of the file by specifying page group '[0..0)' .

    ;; 3.0 A hole at the beginning of a file is represented as follows: if file F is missing pages (0..n) and the preceding file in the lexicographic ordering is
    ;; file E of size m, then the interval in the file map representing the hole looks like this: (key: (E, m), volumePage: nullVolumePage, nextKey: (F, n)).

    ;; 4.0 A hole in the middle of the file (e.g.  missing pages (m..n)) looks like this: (key: (F, m), volumePage: nullVolumePage, nextKey: (F, n)).

    ;; 5.0 This procedure does not care whether the page group being deleted corresponds to a hole in a file or to a real run of pages on the volume,
    ;; with the exception of a hole at the beginning of a file.  If the page group to be deleted is fully contained in a hole at the beginning of the file, Bug
    ;; (pageGroupNotFound) is raised.

    (WITH.MONITOR \VFMmonitor
        (UNINTERRUPTABLY
            (PROG ((key (create Key
                                fileID _ (fetch (FileDescriptor fileID) of filePtr)
                                filePage _ (IDIFFERENCE (fetch (PageGroup nextFilePage) of groupPtr)
                                                        (if (ZEROP (fetch (PageGroup nextFilePage) of groupPtr))
                                                            then 0
                                                          else 1))
                                type _ (fetch (FileDescriptor type) of filePtr)))
                   (interval (create Interval))
                   (fileSize (fetch (FileDescriptor size) of filePtr)))
                                                                   ; (ASSERT (LEQ (fetch (PageGroup nextFilePage) of groupPtr)
                                                                   ; fileSize))
                  (\VFMContextSet vol)
                  (MESASETQ interval (\VFMGet key 0)
                         Interval)                                 ; get interval containing last page of group
                  (if (OR (NOT (EQL (fetch (Key fileID) of (fetch (Interval key) of interval))
```

```
                                (fetch (FileDescriptor fileID) of filePtr)))
                        (AND (NOT (EQL (fetch (Key fileID) of (fetch (Interval nextKey) of interval))
                                       (fetch (FileDescriptor fileID) of filePtr)))
                             (EQL (fetch (Interval volumePage) of interval)
                                  nullVolumePage)))
                    then (DiskError "HARD DISK ERROR" "Page group not found"))
            ;; for a zero-length file, interval.nextKey.fileID # filePtr.fileID BUT interval.volumePage # nullVolumePage
                [replace (PageGroup filePage) of groupPtr with (replace (Key filePage) of key
                                                                        with (IMAX (fetch (Key filePage)
                                                                                          of (fetch (Interval key)
                                                                                                    of interval))
                                                                                   (fetch (PageGroup filePage)
                                                                                          of groupPtr]
                [replace (PageGroup volumePage) of groupPtr
                   with (if (EQL (fetch (Interval volumePage) of interval)
                                 nullVolumePage)
                            then nullVolumePage
                         else (IPLUS (fetch (Interval volumePage) of interval)
                                     (IDIFFERENCE (fetch (PageGroup filePage) of groupPtr)
                                                  (fetch (Key filePage) of (fetch (Interval key) of interval]
                (replace (PageGroup nextFilePage) of groupPtr with (IMIN (fetch (Key filePage)
                                                                                of (fetch (Interval nextKey)
                                                                                          of interval))
                                                                         (fetch (PageGroup nextFilePage)
                                                                                of groupPtr)))
                                                                        ; deal with the starting page of the page group first
                (if [AND (MESAEQUAL key (fetch (Interval key) of interval)
                                    Key)
                         (OR (NOT (EQL (fetch (PageGroup nextFilePage) of groupPtr)
                                       fileSize))
                             (NOT (ZEROP (fetch (Key filePage) of key]
                    then (\VFMDelete key 0))
                [if (NOT (ZEROP (fetch (Key filePage) of key)))
                    then (PROG [(previousKey (create Key
                                                     fileID _ (fetch (FileDescriptor fileID) of filePtr)
                                                     filePage _ (SUB1 (fetch (Key filePage) of key]
                               (if (EQL (fetch (Key fileID) of (fetch (Interval key) of (\VFMGet previousKey 0)))
                                        (fetch (FileDescriptor fileID) of filePtr))
                                   then                 ; key.filePage is not the first (existing) page of the file
                                        (\VFMInsert key nullVolumePage 0]
                                                                        ; now deal with the ending page of the page group
                (replace (Key filePage) of key with (fetch (PageGroup nextFilePage) of groupPtr))
                (if (EQL (fetch (Key filePage) of key)
                         fileSize)
                    then (\VFMDelete key 0))
                (if [AND [NOT (EQL (fetch (Key filePage) of key)
                                   (fetch (Key filePage) of (fetch (Interval nextKey) of interval]
                         (EQL (fetch (Key fileID) of key)
                              (fetch (Key fileID) of (fetch (Interval nextKey) of interval]
                    then (\VFMInsert key [if (EQL (fetch (PageGroup volumePage) of groupPtr)
                                                  nullVolumePage)
                                             then nullVolumePage
                                          else (IPLUS (fetch (PageGroup volumePage) of groupPtr)
                                                      (IDIFFERENCE (fetch (PageGroup nextFilePage) of groupPtr)
                                                                   (fetch (PageGroup filePage) of groupPtr]
                                         0)))))])
```

(\VFMGetPageGroup

```
[LAMBDA (vol filePtr filePage)                          ; Edited 22-Oct-87 16:47 by amd
    ;; finds page group containing key (filePage = nextFilePage = size when off end of file)
    (WITH.MONITOR \VFMmonitor
        (UNINTERRUPTABLY
            (PROG ((key (create Key
                                fileID _ (fetch (FileDescriptor fileID) of filePtr)
                                filePage _ filePage
                                type _ (fetch (FileDescriptor type) of filePtr)))
                   (interval (create Interval)))
                  (\VFMContextSet vol)
                  (MESASETQ interval (\VFMGet key 0)
                            Interval)
                  [RETURN (AND (EQL (fetch (Key fileID) of (fetch (Interval key) of interval))
                                    (fetch (FileDescriptor fileID) of filePtr))
                               (create PageGroup
                                       filePage _ (fetch (Key filePage) of (fetch (Interval key) of interval))
                                       volumePage _ (fetch (Interval volumePage) of interval)
                                       nextFilePage _ (fetch (Key filePage)
                                                             of (if (EQL (fetch (Key fileID) of (fetch (Interval nextKey)
                                                                                                       of interval))
                                                                         (fetch (FileDescriptor fileID) of filePtr))
                                                                    then (fetch (Interval nextKey) of interval)
                                                                  else (fetch (Interval key) of interval]
                                                                         ; covers page zero and size requests

            ))])
```

(\VFMInsertPageGroup

```
[LAMBDA (vol filePtr groupPtr)                                  ; Edited 22-Oct-87 16:47 by amd
    ;; inserts a pageGroup into B-tree (unordered inserts are merged for rebuild)
    (WITH.MONITOR \VFMmonitor
        (UNINTERRUPTABLY
            (PROG ((key (create Key
                                fileID _ (fetch (FileDescriptor fileID) of filePtr)
                                filePage _ (fetch (PageGroup filePage) of groupPtr)
                                type _ (fetch (FileDescriptor type) of filePtr)))
                   (interval (create Interval)))
                  (\VFMContextSet vol)
                  (MESASETQ interval (\VFMGet key 0)
                        Interval)
                  (if (MESAEQUAL (fetch (Interval key) of interval)
                            key Key)
                      then (\VFMDelete key 0)
                           (MESASETQ interval (\VFMGet key 0)
                                 Interval))
                  (if [OR [NOT (EQL (IDIFFERENCE (fetch (Key filePage) of key)
                                                 (fetch (Key filePage) of (fetch (Interval key) of interval)))
                                    (IDIFFERENCE (fetch (PageGroup volumePage) of groupPtr)
                                                 (fetch (Interval volumePage) of interval]
                          (NOT (EQL (fetch (Key fileID) of key)
                                    (fetch (Key fileID) of (fetch (Interval key) of interval]
                      then                                      ; don't merge with previous
                           (\VFMInsert key (fetch (PageGroup volumePage) of groupPtr)
                                  0)
                           (MESASETQ interval (\VFMGet key 0)
                                  Interval))
                  (replace (Key filePage) of key with (fetch (PageGroup nextFilePage) of groupPtr))
                  (if [AND (NOT (MESAEQUAL (fetch (Interval nextKey) of interval)
                                           key Key))
                           (NOT (EQL (fetch (PageGroup filePage) of groupPtr)
                                     (fetch (PageGroup nextFilePage) of groupPtr]
                      then (\VFMInsert key nullVolumePage 0))
                  (if [AND (MESAEQUAL (fetch (Interval nextKey) of interval)
                                      key Key)
                           (EQL (fetch (Interval volumePage) of (\VFMGet key 0))
                                (IPLUS (fetch (Interval volumePage) of interval)
                                       (IDIFFERENCE (fetch (Key filePage) of (fetch (Interval nextKey) of interval))
                                                    (fetch (Key filePage) of (fetch (Interval key) of interval]
                      then (\VFMDelete key 0)                   ; merge with following
                           )))))])

)

;; The following are routines internal to the volume file map module.

(DEFINEQ

(\VFMContextSet
  [LAMBDA (vol)                                                 ; Edited 22-Oct-87 12:12 by amd
    ;; vol: LogicalVolumeDescriptor
    (SETQ \VFMvolumeHandle vol])

(\VFMCreateVPage
  [LAMBDA NIL                                                   (* hts%: " 6-Aug-85 12:44")

;;; Returns SWAPPEDFIXP
                                                                ; Internal

;;; Calls VolAllocMap.AllocPageGroup to get a new page for the vfm B-tree.  Returns its volume-relative page number.

    (with LogicalVolumeDescriptor \VFMvolumeHandle
          (PROG [(group (create PageGroup
                                filePage _ 0
                                volumePage _ 0
                                nextFilePage _ 1))
                 (vfmFileD (ELT \PFFileMapFileD (\PFVolumeNumber \VFMvolumeHandle]
                (OR (\VAMAllocPageGroup \VFMvolumeHandle vfmFileD group)
                    (DiskError "HARD DISK ERROR" "File map Btree split failed."))
                (RETURN (fetch (PageGroup volumePage) of group])

(\VFMDelete
  [LAMBDA (deleteKey deleteLevel)                               (* hts%: "24-Jan-85 16:23")

;;; key: Key, level: SMALLP
                                                                ; Internal

;;; Deletes the index corresponding to key.  Error if no such index.  No merging is done here explicitly;  it happens as a side-effect of (Find ...)
```

```
    (DECLARE (SPECVARS deleteKey deleteLevel))
    (PROG (firstFlag lastFlag volumePage (nextKey (create Key)))
          (DECLARE (SPECVARS firstFlag lastFlag volumePage nextKey))
```

;; volumePage is the page holding the key (delete if firstFlag AND lastFlag) --- nextKey is the following key; must be slid down over deleted key

```
          (\VFMFind deleteKey deleteLevel (FUNCTION \VFMDelete1))
          [if firstFlag
              then
```

;; Since this is the first entry in a page, there is a reference to it in the next higher level. If the current page will become empty ;; due to the delete, we simply delete the reference in the higher page. Otherwise we must replace the reference with the new ;; first entry of the current page.

```
                   (\VFMDelete deleteKey (ADD1 deleteLevel))
                   (if lastFlag
                       then (\VFMFreeVPage volumePage)
                     else (\VFMInsert nextKey volumePage (ADD1 deleteLevel]
          (\VFMFind deleteKey deleteLevel (FUNCTION \VFMDelete2))
                                                                  ; Get the preceding index

      ])
```

(\VFMDelete1

```
  [LAMBDA NIL                                                     (* amd "10-Feb-86 18:16")
                                                                  ; Internal
```

;;; Save the following Index in nextKey; set firstFlag, lastFlag, and volumePage. Shift entries if at beginning of page.

```
    (SETQ firstFlag (EQP \VFMlowPtr 0))
    (SETQ lastFlag (EQP \VFMhighPtr (fetch (Buffer used) of \VFMbuffer)))
    (SETQ volumePage (fetch (Interval volumePage) of \VFMinterval))
    (MESASETQ nextKey (fetch (Index key) of \VFMhigh)
           Key)                                                   (* (ASSERT (MESAEQUAL (fetch
                                                                  (Index key) of \VFMlow) deleteKey Key)))
    (if (AND firstFlag (NOT lastFlag))
        then (\VFMSmartBLT \VFMbuffer (\ADDBASE \VFMbuffer \VFMhighPtr)
                     (replace (Buffer used) of \VFMbuffer with (IDIFFERENCE (fetch (Buffer used) of \VFMbuffer)
                                                                  \VFMhighPtr)))
             (\VFMMarkBufferDirty \VFMbuffer])
```

(\VFMDelete2

```
  [LAMBDA NIL                                                     (* hts%: "29-Jan-85 20:50")
                                                                  ; Internal
```

;;; Slide the entries down over nextKey, and then reinsert nextKey in place of the deleted entry. Be careful to preserve the correct volumePage.

```
    (replace (Index key) of \VFMhigh with nextKey)
    (replace (Index volumePage) of \VFMhigh with (fetch (Index volumePage) of \VFMlow))
    (MESASETQ \VFMlow \VFMold Index)
    (SETQ \VFMlowPtr \VFMoldPtr)
    (\VFMSmartBLT (\ADDBASE \VFMbuffer (IPLUS \VFMlowPtr (MESASIZE Index)))
           (\ADDBASE \VFMbuffer \VFMhighPtr)
           (IDIFFERENCE (fetch (Buffer used) of \VFMbuffer)
                  \VFMhighPtr))
    (\VFMPutNext (fetch (Index key) of \VFMhigh)
           (fetch (Index volumePage) of \VFMhigh)
           deleteLevel)
    (replace (Buffer used) of \VFMbuffer with (IDIFFERENCE (IPLUS \VFMlowPtr (fetch (Buffer used) of \VFMbuffer))
                                                  \VFMhighPtr))
    (\VFMMarkBufferDirty \VFMbuffer])
```

(\VFMFind

```
  [LAMBDA (key level proc)                                        ; Edited 22-Oct-87 12:30 by amd
```

;; key: Key, level: SMALLP, proc: FUNCTION

;; executes proc with context (buffer, low, \VFMhigh) surrounding key (merges too)

```
    (MESASETQ \VFMinterval (\VFMGet key (ADD1 level))
           Interval)
    (SETQ \VFMbuffer (\VFMGetBufferFor (fetch (Interval volumePage) of \VFMinterval)))
```

;; Initialize reader

```
    (replace (Index key) of \VFMhigh with (fetch (Interval key) of \VFMinterval))
    (replace (Index volumePage) of \VFMhigh with nullVolumePage)
    (MESASETQ \VFMold (MESASETQ \VFMlow \VFMhigh Index)
           Index)
    (SETQ \VFMoldPtr (SETQ \VFMlowPtr (SETQ \VFMhighPtr 0)))
```

;; Scan this page till key is passed

```
    (repeatuntil (\VFMLower key (fetch (Index key) of \VFMhigh)) do (\VFMReadNext))
    (APPLY proc)
    (if (AND (ILEQ (fetch (Buffer used) of \VFMbuffer)
                   (IQUOTIENT (MESASIZE Buffer)
                          3))
             (NOT (MESAEQUAL (fetch (Interval nextKey) of \VFMinterval)
                         \VFMmaxKey Key)))
        then (\VFMMerge (fetch (Index key) of \VFMold)
```

```
                    level])
```

(\VFMFreeVPage

```
  [LAMBDA (volumePage)                                              (* hts%: " 9-Jan-85 17:31")

;;; volumePage: SWAPPEDFIXP
                                                                    ; Internal

;;; calls VolAllocMap.FreePageGroup to free a page of the vfm BTree

    (with LogicalVolumeDescriptor \VFMvolumeHandle
          (PROG [(group (create PageGroup
                                filePage _ volumePage
                                volumePage _ volumePage
                                nextFilePage _ (ADD1 volumePage)))
                 (vfmFileD (ELT \PFFileMapFileD (\PFVolumeNumber \VFMvolumeHandle]
                (\VAMFreePageGroup \VFMvolumeHandle vfmFileD group)))
    (\VFMKillBuffer volumePage])
```

(\VFMGet

```
  [LAMBDA (getKey getLevel)                                         ; Edited 22-Oct-87 16:49 by amd
    ;; key: Key, level: SMALLP;  returns Interval
    (DECLARE (SPECVARS getKey getLevel))
    (if (GREATERP getLevel treeDepth)
        then (DiskError "HARD DISK ERROR" "Can't find BTree entry"))
    (if (EQL getLevel treeDepth)
        then ;; If you've run out of interval cache to check, just return the widest possible interval
             (create Interval
                     key _ \VFMnullKey
                     volumePage _ (fetch (LogicalVolumeDescriptor vfmStart) of \VFMvolumeHandle)
                     nextKey _ \VFMmaxKey)
      else (MESASETQ \VFMinterval (ELT (\VFMGetInterval)
                                       getLevel)
                     Interval)
           (if [OR (\VFMLower getKey (fetch (Interval key) of \VFMinterval))
                   (NOT (\VFMLower getKey (fetch (Interval nextKey) of \VFMinterval]
               then ;; If the cached interval for the current level isn't the one you were looking for, then search one level closer to the root of the
                    ;; btree
                    (\VFMFind getKey getLevel '\VFMGet1))
           (ELT (\VFMGetInterval)
                getLevel])
```

(\VFMGet1

```
  [LAMBDA NIL                                                       ; Edited 22-Oct-87 12:31 by amd
    (PROG ((interval (ELT (\VFMGetInterval)
                          getLevel)))
          (if interval
              then (replace (Interval key) of interval with (fetch (Index key) of \VFMlow))
                   (replace (Interval volumePage) of interval with (fetch (Index volumePage) of \VFMhigh))
                   (replace (Interval nextKey) of interval with (fetch (Index key) of \VFMhigh])
```

(\VFMInsert

```
  [LAMBDA (insertKey insertVolumePage insertLevel)                  ; Edited 22-Oct-87 14:44 by amd
    ;; key: Key, volumePage: PageNumber, level: Level
    ;; Inserts an Index containing key and volumePage, calling Split if necessary.
    (DECLARE (SPECVARS insertKey insertVolumePage insertLevel))
    (PROG (splitFlag)
          (DECLARE (SPECVARS splitFlag))
     ;; Try the insert.
          (\VFMFind insertKey insertLevel '\VFMInsert1)
     ;; If there wasn't enough space to insert, split the page and retry the insertion.
          (if splitFlag
              then (\VFMSplit insertKey insertLevel)
                   (\VFMFind insertKey insertLevel '\VFMInsert1])
```

(\VFMInsert1

```
  [LAMBDA NIL                                                       ; Edited 22-Oct-87 14:44 by amd
    (PROG NIL
          (if (SETQ splitFlag (IGREATERP (fetch (Buffer used) of \VFMbuffer)
                                         maxReadPtr))
              then (RETURN))
          (if (ILESSP \VFMlowPtr (fetch (Buffer used) of \VFMbuffer))
              then (\VFMSmartBLT (\ADDBASE \VFMbuffer (IPLUS \VFMlowPtr (TIMES (MESASIZE Index)
                                                                               2)))
                                 (\ADDBASE \VFMbuffer \VFMhighPtr)
                                 (IDIFFERENCE (fetch (Buffer used) of \VFMbuffer)
```

```
                                \VFMhighPtr))
                    (\VFMPutNext insertKey (fetch (Index volumePage) of \VFMhigh)
                           insertLevel)
             else (\VFMSmartBLT (\ADDBASE \VFMbuffer (IPLUS \VFMlowPtr (MESASIZE Index)))
                           (\ADDBASE \VFMbuffer \VFMhighPtr)
                           (IDIFFERENCE (fetch (Buffer used) of \VFMbuffer)
                                  \VFMhighPtr)))
          (\VFMPutNext (fetch (Index key) of \VFMhigh)
                 insertVolumePage insertLevel)
          (replace (Buffer used) of \VFMbuffer with (IDIFFERENCE (IPLUS \VFMlowPtr (fetch (Buffer used)
                                                                                      of \VFMbuffer))
                                                                 \VFMhighPtr))
          (\VFMMarkBufferDirty \VFMbuffer])
```

(\VFMLower

```
[LAMBDA (A B)                                                    ; Edited 22-Oct-87 16:49 by amd
```

;; a: Key, b: Key; returns BOOLEAN

;; Compares two keys for ordering: a < b iff a.id < a.id or (a.id = b.id and a.page < b.page) ; any key < maxKey to close key space. Somewhat icky ;; because fileIDs are 32 bit #s where high bit set means high positive number, not negative.

```
(PROG ((AFILE (fetch (Key fileID) of A))
       (BFILE (fetch (Key fileID) of B))
       (APAGE (fetch (Key filePage) of A))
       (BPAGE (fetch (Key filePage) of B)))
      (RETURN (OR (if (GEQ AFILE 0)
                      then (if (LESSP BFILE 0)
                               then T
                             else (LESSP AFILE BFILE))
                    else (if (GEQ BFILE 0)
                             then NIL
                           else (GREATERP AFILE BFILE)))
                  (AND (EQL AFILE BFILE)
                       (OR (ILESSP APAGE BPAGE)
                           (MESAEQUAL B \VFMmaxKey Key])
```

(\VFMMerge

```
[LAMBDA (mergeKey mergeLevel)                                    (* hts%: "25-Jan-85 12:17")
```

;;; key: Key, level: SMALLP

; Internal

;;; Tries to merge page of oldInterval with next page at same mergeLevel or with root; cannot merge last page of any mergeLevel except rootlevel

```
(DECLARE (SPECVARS mergeKey mergeLevel))
(PROG (mergeFlag (leftInterval (create Interval))
                 (rightInterval (create Interval)))
      (DECLARE (SPECVARS mergeFlag leftInterval rightInterval))
```

;; get a valid volumePage

```
      (MESASETQ leftInterval (\VFMGet mergeKey (ADD1 mergeLevel))
             Interval)
      (\VFMFind (fetch (Interval nextKey) of leftInterval)
             mergeLevel
             (FUNCTION \VFMMerge1))                              ; beware the merging
```

;; Get rid of the old reference to the merging page.

```
      (\VFMDelete (fetch (Interval nextKey) of leftInterval)
             (ADD1 mergeLevel))
```

;; If the page was not actually merged, insert the new Index, else free up the merged page.

```
      (if mergeFlag
          then (\VFMFreeVPage (fetch (Interval volumePage) of rightInterval))
        else (\VFMInsert (fetch (Interval key) of rightInterval)
                         (fetch (Interval volumePage) of rightInterval)
                         (ADD1 mergeLevel])
```

(\VFMMerge1

```
[LAMBDA NIL                                                      ; Edited 22-Oct-87 16:51 by amd
  (PROG (xtraBufferUsed)
        (MESASETQ rightInterval \VFMinterval Interval)
        (SETQ \VFMxtraBuffer (\VFMGetBufferFor (fetch (Interval volumePage) of leftInterval)))
        (SETQ xtraBufferUsed (fetch (Buffer used) of \VFMxtraBuffer))
                                                                 ; xtraBufferUsed used to solve stack modeling error
        (if (EQL mergeLevel (SUB1 treeDepth))
            then (replace (Buffer used) of \VFMxtraBuffer with 0))
        (if (SETQ mergeFlag (ILESSP (IPLUS (fetch (Buffer used) of \VFMbuffer)
                                           (fetch (Buffer used) of \VFMxtraBuffer))
                                    (MESASIZE Buffer)))
            then ;; If merging possible then merge pages.  Merge buffer with aux buffer.
                 (\VFMSmartBLT (\ADDBASE \VFMxtraBuffer xtraBufferUsed)
                        \VFMbuffer
                        (fetch (Buffer used) of \VFMbuffer))
                 (replace (Buffer used) of \VFMxtraBuffer with (IPLUS (fetch (Buffer used) of \VFMxtraBuffer)
```

```
                                                    (fetch (Buffer used) of \VFMbuffer)))
                                                ; buffer.used remains to prevent Find from attempting a merge

        else ;; otherwise balance pages simply to provide hysteresis against futile merge attempts.  First find middle.
             (while (ILESSP \VFMlowPtr (IQUOTIENT (IDIFFERENCE (fetch (Buffer used) of \VFMbuffer)
                                                                (fetch (Buffer used) of \VFMxtraBuffer))
                                                   2))
                do (\VFMReadNext))
             ;; move first of \VFMbuffer to xtra
             (\VFMSmartBLT (\ADDBASE \VFMxtraBuffer xtraBufferUsed)
                    \VFMbuffer \VFMlowPtr)
             ;; slide down the rest of \VFMbuffer
             (\VFMSmartBLT \VFMbuffer (\ADDBASE \VFMbuffer \VFMlowPtr)
                    (IDIFFERENCE (fetch (Buffer used) of \VFMbuffer)
                           \VFMlowPtr))
             ;; Straighten out end-of-node info.
             (replace (Buffer used) of \VFMxtraBuffer with (IPLUS (fetch (Buffer used) of \VFMxtraBuffer)
                                                                  \VFMlowPtr))
             (replace (Buffer used) of \VFMbuffer with (IDIFFERENCE (fetch (Buffer used) of \VFMbuffer)
                                                                    \VFMlowPtr))
             ;; use \VFMlow to insert while it is still valid
             (replace (Interval key) of rightInterval with (fetch (Index key) of \VFMlow)))
    ;; Finish up.
        (\VFMMarkBufferDirty \VFMbuffer)
        (\VFMMarkBufferDirty \VFMxtraBuffer)
        (SETQ \VFMxtraBuffer NIL])
```

(\VFMPutNext

```
[LAMBDA (key volumePage level)                                  (* hts%: "25-Jan-85 15:25")
```

;; key: Key, volumePage: SWAPPEDFIXP, level: SMALLP ; Internal

;; Compresses item in the context of low. Note the side effect on \VFMlow but not on high!! No compression is implemented in this version, but
;; useful one would include: front compression (especially to shrink page groups back to 2 fields)

```
(MESASETQ \VFMold \VFMlow Index)
(SETQ \VFMoldPtr \VFMlowPtr)
(replace (Index key) of \VFMlow with key)
(replace (Index volumePage) of \VFMlow with volumePage)
(MESASETQ (\ADDBASE (fetch (Buffer data) of \VFMbuffer)
                    \VFMlowPtr)
       \VFMlow Index)
(SETQ \VFMlowPtr (IPLUS \VFMoldPtr (MESASIZE Index)))
```

;; keep cache up to date in the face of changes

```
(SETA (\VFMGetInterval)
      level
      (create Interval
              key _ (fetch (Index key) of \VFMold)
              volumePage _ (fetch (Index volumePage) of \VFMlow)
              nextKey _ (fetch (Index key) of \VFMlow)))
```

;; Mark buffer dirty

```
(\VFMMarkBufferDirty \VFMbuffer])
```

(\VFMReadNext

```
[LAMBDA NIL                                                     ; Edited 22-Oct-87 16:52 by amd
```

;; Decompresses item at \VFMhigh to become \VFMlow & bumps high. Note the side effect on \VFMlow and not high. No compression is
;; implemented in this version

```
(OR (ILEQ \VFMhighPtr (fetch (Buffer used) of \VFMbuffer))
    (DiskError "HARD DISK ERROR" "Read too far in ReadNext"))
(MESASETQ \VFMold \VFMlow Index)
(SETQ \VFMoldPtr \VFMlowPtr)
(MESASETQ \VFMlow \VFMhigh Index)
(SETQ \VFMlowPtr \VFMhighPtr)
(if (ILESSP \VFMhighPtr (fetch (Buffer used) of \VFMbuffer))
    then                                                        ; Loophole
         (MESASETQ \VFMhigh (\ADDBASE (fetch (Buffer data) of \VFMbuffer)
                                      \VFMhighPtr)
                   Index)
         (SETQ \VFMhighPtr (IPLUS \VFMhighPtr (MESASIZE Index)))
  else                                                          ; Leave ptr alone
       (replace (Index key) of \VFMhigh with \VFMmaxKey)
       (replace (Index volumePage) of \VFMhigh with nullVolumePage])
```

(\VFMSplit

```
[LAMBDA (splitKey splitLevel)                                   (* hts%: " 5-Jan-85 16:29")
```

;;; key: Key, level: SMALLP

; Internal

;;; moves half of \DFSVFMbuffer (or root) to xtraBuffer, creating new page of tree

```
    (DECLARE (SPECVARS splitKey splitLevel))
    (PROG ((keyStone (create Key))
           (page (\VFMCreateVPage)))                    ; keyStone is the half way mark
          (DECLARE (SPECVARS keyStone page))
          (\VFMFind splitKey splitLevel (FUNCTION \VFMSplit1))
          (\VFMInsert keyStone page (ADD1 splitLevel])
```

(\VFMSplit1

```
  [LAMBDA NIL                                           (* hts%: "25-Jan-85 12:01")
                                                        ; Internal
    ;; Read in an extra page into which to copy the second half of the current node
    (SETQ \VFMxtraBuffer (\VFMGetBufferFor page))
    ;; Find the middle of the page to split
    (SETQ \VFMhighPtr 0)
    (replace (Index key) of \VFMhigh with (fetch (Interval key) of \VFMinterval))
    (replace (Index volumePage) of \VFMhigh with nullVolumePage)
    (repeatuntil (IGREATERP \VFMhighPtr (IQUOTIENT (fetch (Buffer used) of \VFMbuffer)
                                                   2))
       do (\VFMReadNext))
    ;; Move the last half of buffer to extra buffer.
    (\BLT \VFMxtraBuffer (\ADDBASE (fetch (Buffer data) of \VFMbuffer)
                                   \VFMlowPtr)
          (replace (Buffer used) of \VFMxtraBuffer with (IDIFFERENCE (fetch (Buffer used) of \VFMbuffer)
                                                                     \VFMlowPtr)))
    (replace (Buffer used) of \VFMbuffer with \VFMlowPtr)
    (MESASETQ keyStone (fetch (Index key) of \VFMlow)
           Key)
    ;; Mark buffers dirty so that they will be flushed out to disk, and clear the extra buffer holder (just to prevent confusion)
    (\VFMMarkBufferDirty \VFMbuffer)
    (\VFMMarkBufferDirty \VFMxtraBuffer)
    (SETQ \VFMxtraBuffer NIL])

)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \VFMmaxID \VFMmaxKey \VFMnullKey \VFMvolumeHandle \VFMinterval \VFMold \VFMlow \VFMhigh \VFMoldPtr
       \VFMlowPtr \VFMhighPtr \VFMmonitor)
)

;; Buffer management

(DEFINEQ
```

(\VFMGetBufferFor

```
  [LAMBDA (VOLPAGENUM)                                  ; Edited 22-Oct-87 16:53 by amd
    ;; Try to find btree page in buffer pool.  If there, move to front of buffer pool.  Otherwise, read in the requred page and put it at the front of the pool.
    ;; If buffer pool is > maxbufferpoolsize then flush the last page in the pool
    (LET ((BUFFER (\VFMKillBuffer VOLPAGENUM))
          LAST FLUSH)
         (if BUFFER
             then ;; Move buffer to front of buffer list
                  (push \VFMbufferPool BUFFER)
           else ;; Create and read in new buffer
                (push \VFMbufferPool (SETQ BUFFER (create \BTREEBUF
                                                          VOLUME _ \VFMvolumeHandle
                                                          VOLPAGENUM _ VOLPAGENUM
                                                          PAGE _ (create Buffer)
                                                          DIRTY _ NIL)))
                (\PFGetFileMapPage \VFMvolumeHandle VOLPAGENUM (fetch (\BTREEBUF PAGE) of BUFFER))
                ;; Shorten buffer pool if necessary
                (if [SETQ FLUSH (CDR (SETQ LAST (FNTH \VFMbufferPool \VFMbufferSize]
                    then (RPLACD LAST NIL)
                         (\VFMSaveBuffer T FLUSH)))
         ;; Finally set the main buffer page to be the selected buffer page.
         (fetch (\BTREEBUF PAGE) of BUFFER])
```

(\VFMSaveBuffer

```
  [LAMBDA (notAll whichBuffers evenIfNotDirty)          ; Edited 22-Oct-87 16:54 by amd
    ;; Flushes dirty buffers.  If notAll is true, then it is to save only the specified buffers
    (OR notAll (SETQ whichBuffers \VFMbufferPool))
```

```
    (for BUF inside whichBuffers when (OR (fetch (\BTREEBUF DIRTY) of BUF)
                                          evenIfNotDirty)
       do (\PFPutFileMapPage (fetch (\BTREEBUF VOLUME) of BUF)
                             (fetch (\BTREEBUF VOLPAGENUM) of BUF)
                             (fetch (\BTREEBUF PAGE) of BUF))
          (replace (\BTREEBUF DIRTY) of BUF with NIL])
```

(\VFMClearBuffers

```
  [LAMBDA NIL                                                    (* hts%: "16-Nov-84 15:38")

;;; Clear the btree node cache

    (SETQ \VFMbufferPool NIL])
```

(\VFMKillBuffer

```
  [LAMBDA (VOLPAGENUM)                                           ; Edited 22-Oct-87 14:53 by amd
    ;; Remove the buffer for a btree node which is being decommissioned.  Return the removed buffer.
    (if (AND (LISTP \VFMbufferPool)
             (\VFMCorrectBufferP (CAR \VFMbufferPool)
                    VOLPAGENUM))
        then (CL:POP \VFMbufferPool)
      else (bind CURRENT for PREV on \VFMbufferPool do (if (AND (LISTP (SETQ CURRENT (CDR PREV)))
                                                                (\VFMCorrectBufferP (CAR CURRENT)
                                                                       VOLPAGENUM))
                                                           then (RETURN (PROG1 (CAR CURRENT)
                                                                               (RPLACD PREV (CDR CURRENT)))))])
```

(\VFMCorrectBufferP

```
  [LAMBDA (BUFFER VOLPAGENUM)                                    ; Edited 22-Oct-87 16:54 by amd
    ;; True iff BUFFER is the right buffer for VOLPAGENUM
    (AND (EQL (fetch (\BTREEBUF VOLUME) of BUFFER)
              \VFMvolumeHandle)
         (EQL (fetch (\BTREEBUF VOLPAGENUM) of BUFFER)
              VOLPAGENUM])
```

(\VFMMarkBufferDirty

```
  [LAMBDA (BUFFERPAGE)                                           ; Edited 22-Oct-87 15:13 by amd
    ;; Note that the specified buffer has been written into and will have to be flushed out to disk.
    (replace (\BTREEBUF DIRTY) of (for BUF in \VFMbufferPool thereis (EQL BUFFERPAGE (fetch (\BTREEBUF PAGE)
                                                                                           of BUF)))
       with T)
    NIL])

)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \VFMbufferPool \VFMbufferSize \VFMbuffer \VFMxtraBuffer)
)

(RPAQ? \VFMbufferSize 10)

;; Interval cache interface

(DEFINEQ
```

(\VFMCreateIntervals

```
  [LAMBDA NIL                                                    ; Edited 22-Oct-87 16:55 by amd
    ;; Conditionally create array to hold interval cache for each volume.  Conditional so that loadfroming this file will not destroy state.
    ;; Interval cache for each volume keeps a finger into the BTree: traces a correct path through the BTree, which need be only partially backtracked
    ;; (if at all) to find any given interval in the BTree.  Saves reading one page at each level of the BTree every time you want to look for an interval.
    (if [NOT (AND (BOUNDP '\VFMintervals)
                  (type? ARRAYP \VFMintervals)
                  (ZEROP (ARRAYORIG \VFMintervals))
                  (EQL maxLogicalVolumes (ARRAYSIZE \VFMintervals]
        then (SETQ \VFMintervals (ARRAY maxLogicalVolumes NIL NIL 0])
```

(\VFMClearIntervals

```
  [LAMBDA NIL                                                    (* hts%: " 5-Jan-85 16:25")

;;; Clears the BTree interval cache so that it will be correctly reinitialized should this lisp image wake up on an alien machine

    (for volume from 0 to (SUB1 maxLogicalVolumes) do (SETA \VFMintervals volume NIL])
```

(\VFMGetInterval

```
  [LAMBDA NIL                                                    ; Edited 22-Oct-87 12:09 by amd
```

```
;; Returns the interval cache for the current volume.  If this interval cache is empty, initializes with a leftmost path through the BTree for that
;; volume.

(PROG ((volNum (\PFVolumeNumber \VFMvolumeHandle)))
      (RETURN (OR (ELT \VFMintervals volNum)
                  (SETA \VFMintervals volNum
                        (bind (intervalArray _ (ARRAY treeDepth NIL NIL 0))
                              (BTreePageNum _ (fetch (LogicalVolumeDescriptor vfmStart) of \VFMvolumeHandle))
                           for level from (SUB1 treeDepth) to 0 by -1
                           do (SETQ \VFMbuffer (\VFMGetBufferFor BTreePageNum))
                              [SETQ BTreePageNum (fetch (Interval volumePage)
                                                        of (SETA intervalArray level
                                                                 (create Interval
                                                                         key _ \VFMnullKey
                                                                         volumePage _ (fetch (Index volumePage)
                                                                                             of \VFMbuffer)
                                                                         nextKey _ (fetch (Index key) of \VFMbuffer
                                                                                          ]
                           finally (RETURN intervalArray])
```

(\VFMBlankInterval

```
  [LAMBDA NIL                                                    (* hts%: "26-Jan-85 18:57")

;;; Returns the interval cache for the current volume.  If this interval cache is empty, initializes with a blank set of intervals with InitMap will fill with a
;;; leftmost path through the BTree for that volume.

;;; Should be called by InitMap only.

    (PROG ((volNum (\PFVolumeNumber \VFMvolumeHandle)))
          (RETURN (OR (ELT \VFMintervals volNum)
                      (SETA \VFMintervals volNum (PROG ((intervalCache (ARRAY treeDepth NIL NIL 0)))
                                                       (for level from 0 to (SUB1 treeDepth)
                                                          do (SETA intervalCache level (create Interval)))
                                                       (RETURN intervalCache])

)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \VFMintervals)
)

;; BLT routine that doesn't stomp on itself for overlapping intervals

(DEFINEQ
```

(\VFMSmartBLT

```
  [LAMBDA (DBASE SBASE NWORDS)                                   (* hts%: "24-Jun-84 15:57")
                                                                 ; This is necessary because \BLT will not copy overlapping
                                                                 ; intervals correctly in one direction.
    (if (AND (PTRGTP SBASE DBASE)
             (PTRGTP (\ADDBASE DBASE NWORDS)
                     SBASE))
        then (for i from 0 to (SUB1 NWORDS) do (\PUTBASE DBASE i (\GETBASE SBASE i)))
             DBASE
      else (\BLT DBASE SBASE NWORDS])

)

;; Loading initialization

(DEFINEQ
```

(\VFMAtLoad

```
  [LAMBDA NIL                                                    (* hts%: "25-Jan-85 11:50")

;;; Initialize global variables for the volume file map

    (SETQ \VFMmaxID -1)
    (SETQ \VFMmaxKey (create Key
                             fileID _ \VFMmaxID
                             filePage _ lastPageNumber))
    (SETQ \VFMnullKey (create Key))
    (SETQ \VFMvolumeHandle NIL)
    (SETQ \VFMinterval (create Interval))
    (SETQ \VFMold (create Index))
    (SETQ \VFMlow (create Index))
    (SETQ \VFMhigh (create Index))
    (SETQ \VFMoldPtr 0)
    (SETQ \VFMlowPtr 0)
    (SETQ \VFMhighPtr 0)
    (\VFMCreateIntervals)
    (SETQ \VFMmonitor (CREATE.MONITORLOCK '\VFMmonitor])

)
```

(**VFMAtLoad**)

```
(PUTPROPS LOCALFILE MAKEFILE-ENVIRONMENT (:READTABLE "INTERLISP" :PACKAGE "INTERLISP"))
```

FUNCTION INDEX

CONSTANT INDEX

RECORD INDEX

VARIABLE INDEX

MACRO INDEX

PROPERTY INDEX