```
DIRECTORY
    Mopcodes: FROM "Mopcodes",
    System: FROM "System";

File: DEFINITIONS =
BEGIN

    --File identifiers, capabilities, and types
ID: TYPE = System.FileID;   -- = RECORD[System.UniversalID]
Capability: TYPE = PRIVATE RECORD [fID: ID, permissions: Permissions];
Permissions: TYPE = [0..32);    --This declaration simulates SET OF {read, write, grow, shrink, delete}
    read: Permissions = 1;
    write: Permissions = 2;
    grow: Permissions = 4;
    shrink: Permissions = 8;
    delete: Permissions = 16;

nullID: ID = [System.nullID];
nullCapability: Capability = Capability[nullID, 0];

ShowCapability: PROCEDURE [file: Capability] RETURNS [fID: ID, permissions: Permissions] =
    INLINE BEGIN RETURN [file.fID, file.permissions] END;
LimitPermissions: PROCEDURE [originalFC: Capability, maxPermissions: Permissions] RETURNS [lesserFC: Capability] =
    INLINE BEGIN
    PermissionsAND: PROCEDURE [p1, p2: Permissions] RETURNS [Permissions] =
        MACHINE CODE BEGIN Mopcodes.zAND END;
    RETURN [Capability[originalFC.fID, LOOPHOLE[PermissionsAND[originalFC.permissions, maxPermissions]]]] END;
Type: TYPE = RECORD [CARDINAL];

    --Addressing within a file
maxPagesPerFile: LONG CARDINAL = 8388607;       -- = 2^23 - 1
PageNumber: TYPE = LONG CARDINAL;    -- Simulates TYPE = {0..maxPagesPerFile};
    firstPageNumber: PageNumber = 0;
    lastPageNumber: PageNumber = 8388606;    -- should be maxPagesPerFile - 1;
PageCount: TYPE = LONG CARDINAL; -- Simulates TYPE = {0..maxPagesPerFile};
    firstPageCount: PageCount = 0;
    lastPageCount: PageCount = maxPagesPerFile;

    --Creating, Deleting, and Moving Files
Create: PROCEDURE [volume: System.VolumeID, initialSize: PageCount, type: Type]
        RETURNS [file: Capability];
Delete: PROCEDURE [file: Capability];
Move: PROCEDURE [file: Capability, volume: System.VolumeID];

    --Immutable Files
MakeImmutable: PROCEDURE [file: Capability];
ReplicateImmutable: PROCEDURE [file: Capability, volume: System.VolumeID];
DeleteImmutable: PROCEDURE [file: Capability, volume: System.VolumeID];

    --Attributes of Files
GetSize: PROCEDURE [file: Capability] RETURNS [size: PageCount];
SetSize: PROCEDURE [file: Capability, size: PageCount];
GetAttributes: PROCEDURE [file: Capability]
        RETURNS [type: Type, immutable, temporary: BOOLEAN, volume: System.VolumeID];
MakePermanent: PROCEDURE [file: Capability];

    --Locating files
IsOnVolume: PROCEDURE [file: Capability, volume: System.VolumeID];

    --Signals and Errors generated by the File interface
Unknown: ERROR [file: Capability];
Error: ERROR [type: ErrorType];
    ErrorType: TYPE = {insufficientPermissions, immutable, nonuniqueID, notImmutable, reservedType };

END.
```

LOG

Time: May 5, 1978 9:16 AM By: Lauer Action: Created file from Pilot Functional Spec
Time: June 21, 1978 3:17 PM By: Lauer Action: Changed value of maxPagesPerFile to correct specification; changed name of "type" parameter to procedure Create
Time: August 3, 1978 12:51 PM By: Lauer Action: Added nullID
Time: August 18, 1978 11:54 AM By: Horsley Action: added reservedType to ErrorType
Time: March 7, 1979 10:42 AM By: Redell Action: Changed PageNumber and PageCount to LONG CARDINAL*, as allowed by Mesa 5.0*
Time: March 30, 1979 4:48 PM By: Redell Action: Converted ShowCapability and LimitPermissions to inline procedures.
Time: April 3, 1979 12:40 PM By: Redell Action: Converted LimitPermissions to use Mopcodes directly to avoid importing InlineDefs.

```
DIRECTORY
    File: FROM "File" USING [Type];

FileTypes: DEFINITIONS =
BEGIN

    -- This file is the authoritative definition of File Types for Pilot files for all purposes. In this module are defined the types
    -- peculiar to Pilot's own files as well as the ranges of the types defined by all clients, applications, and supporting
    -- software. Each such client, application, and supporting software project should maintain its own file of File Types for its
    -- own subrange.

    -- Style Note: The subranges for the different applications, etc., are declared as subrange types of CARDINAL. The
    -- constants are declared to be values of the type File.Type (the correct type to be passed to the File.Create operation of
    -- Pilot). The value of each constant is constructed from the appropriate subrange type in order that range checking can
    -- be applied by the compiler (when this feature becomes available). This style is recommended for client maintained files
    -- of subranges of File Types.

FileType: TYPE = File.Type;

    -- Allocation of subranges of File Types
PilotFileType: TYPE = CARDINAL[0..256);
MesaFileType: TYPE = CARDINAL[256..512);
DCSFileType: TYPE = CARDINAL[512..768);
TestFileType: TYPE = CARDINAL[768..896);
PioneerFileType: TYPE = CARDINAL[896..960);
StarFileType: TYPE = CARDINAL[1024..2048);
CommonSoftwareFileType: TYPE = CARDINAL[2048..3072);
DocProcFileType: TYPE = CARDINAL[3072..4096);

    -- The following is the type to use in the absence of any other file type. It is not particularly recommended.
tUntypedFile: File.Type = [LAST[CARDINAL]];

    -- Pilot File Types

PilotRootFileType: TYPE = PilotFileType[0..15]; -- registered in volume root page
PilotVFileType: TYPE = PilotFileType[1..5]; --accessed as "volume-files"
    tUnassigned: File.Type = [PilotRootFileType[0]];
    tFreePage: File.Type = [PilotVFileType[1]];
    tLogicalVolumeRootPage: File.Type = [PilotVFileType[2]];
    tPhysicalVolumeRootPage: File.Type = [PilotVFileType[3]];
    tVolumeAllocationMap: File.Type = [PilotVFileType[4]];
    tVolumeFileMap: File.Type = [PilotVFileType[5]];
    tAnonymousFile: File.Type = [PilotRootFileType[6]];
    tSpaceBTree: File.Type = [PilotRootFileType[7]];
    tMapLog: File.Type = [PilotRootFileType[8]];
    tCodeFile: File.Type = [PilotRootFileType[9]];
    tBeingMoved: File.Type = [PilotRootFileType[10]];
    tBeingReplicated: File.Type = [PilotRootFileType[11]];
    --... additional RootFileTypes go in here
    tClientRootFile: File.Type = [PilotRootFileType[15]];
    tBootFile: File.Type = [PilotFileType[16]];

END.
```

LOG

Time: July 28, 1978 3:19 PM By: Lauer Action: Created file
Time: August 1, 1978 3:23 PM By: Lauer Action: Permuted the assignment of Pilot file types 1 .. 5 (so that the File Manager can have 1..4 for its own use)
Time: August 29, 1978 8:22 PM By: Lauer Action: added "tMapLog"
Time: October 13, 1978 3:46 PM By: Lauer Action: CR 20.70: added "tCodeFile"
Time: October 17, 1978 1:18 PM By: Lauer Action: CR 20.86: added "DCSFileType"
Time: November 10, 1978 11:03 AM By: Lauer Action: CR (unassigned): added "StarFileType"
Time: December 28, 1978 10:50 AM By: Lauer Action: CR 149: added "TestFileType"
Time: February 9, 1979 5:01 PM By: Lauer Action: CR 156: added "PioneerFileType"
Time: March 12, 1979 4:04 PM By: Redell Action: Added tClientRootFile and subranges of PilotFileType
Time: March 19, 1979 8:57 AM By: Redell Action: Split tRootPage into tLogicalVolumeRootPage and tPhysicalVolumeRootPage

Time: March 20, 1979 10:55 AM By: Redell Action: Added tBeingMoved and tBeingRemapped
Time: May 11, 1979 2:43 PM By: Forrest Action: Added CommonSoftwareFileType
Time: July 30, 1979 9:38 PM By: Redell Action: Added tBootFile
Time: August 12, 1979 5:21 PM By: Forrest Action: Added DocProc File types as per CR ???

```
Forgot: DEFINITIONS =

BEGIN

    -- Signal to be raised if an unimplemented feature is encountered
Unimplemented: SIGNAL;

END.
```

LOG

Time: August 29, 1978 1:53 PM By: Lauer Action: Created file
Time: September 28, 1978 2:16 PM By: Sandman CR20.40: (WorryCallDebugger)
Time: March 2, 1979 12:03 PM By: McJones Action: Moved WorryCallDebugger to RuntimeInternal

```
Process: DEFINITIONS =
BEGIN

    --Initializing monitors and condition variables
InitializeMonitor: PROCEDURE [monitor: LONG POINTER TO MONITORLOCK];
InitializeCondition: PROCEDURE [condition: LONG POINTER TO CONDITION, ticks: Ticks];
Ticks: TYPE = CARDINAL;
Milliseconds: TYPE = CARDINAL;
Seconds: TYPE = CARDINAL;
MsecToTicks: PROCEDURE [Milliseconds] RETURNS [Ticks];
SecondsToTicks: PROCEDURE [Seconds] RETURNS [Ticks];
TicksToMsec: PROCEDURE [Ticks] RETURNS [Milliseconds];

    --Timeouts
SetTimeout: PROCEDURE [condition: LONG POINTER TO CONDITION, ticks: Ticks];
DisableTimeout: PROCEDURE [LONG POINTER TO CONDITION];

    --Detaching processes
Detach: PROCEDURE [PROCESS];

    --Identity of the currently executing process
GetCurrent: PROCEDURE RETURNS [PROCESS];

    --Priorities of processes
SetPriority: PROCEDURE [Priority];
GetPriority: PROCEDURE RETURNS [Priority];
Priority: TYPE = [0..7];

    --Aborting a process
Abort: PROCEDURE [UNSPECIFIED];   --parameter should be any process
DisableAborts: PROCEDURE [LONG POINTER TO CONDITION];

    --Control of Scheduling
Yield: PROCEDURE;

    --Process validation
ValidateProcess: PROCEDURE [UNSPECIFIED];
InvalidProcess: ERROR [process: UNSPECIFIED];

    --Signals and errors generated by the Process machinery
Aborted: ERROR;
TooManyProcesses: ERROR;
END.
```

LOG

Time: May 15, 1978 2:20 PM By: Lauer Action: Created file from AlphaMesa4.0 version of ProcessDefs
Time: June 7, 1978 1:00 PM By: Lauer Action: Changed ERROR declarations to be SIGNALs for compatibility with the Alto Process machinery.
Time: June 22, 1978 9:01 AM By: Lauer Action: changed POINTER's to LONG POINTER's; added "DisableAborts"
Time: July 21, 1978 11:39 AM By: Lauer Action: corrected type of second parameter of InitializeCondition to be Ticks
Time: August 3, 1978 12:56 PM By: Lauer Action: Corrected spelling of "Milliseconds"
Time: August 11, 1978 10:33 AM By: Lauer Action: Converted Aborted and InvalidProcess from SIGNALs to ERRORs; deleted TimedOut
Time: August 29, 1978 10:50 AM By: Lauer Action: Added "GetCurrent"
Time: September 14, 1978 10:35 AM By: Lauer Action: Added "Seconds" and "SecondsToTicks"
Time: May 16, 1979 12:22 PM By: Sandman Action: Added ValidateProcess, changed params of Detach and GetCurrent

```
DIRECTORY
    Environment: FROM "Environment" USING [PageCount, PageNumber, PageOffset, wordsPerPage],
    File: FROM "File" USING [Capability, nullCapability, PageNumber];

Space: DEFINITIONS =

BEGIN

    -- Spaces and space handles
wordsPerPage: CARDINAL = Environment.wordsPerPage;
PageCount: TYPE = Environment.PageCount;
PageNumber: TYPE = Environment.PageNumber;
PageOffset: TYPE = Environment.PageOffset;
Handle: TYPE = PRIVATE RECORD [UNSPECIFIED, UNSPECIFIED];
nullHandle: READONLY Handle;
mds: READONLY Handle;
virtualMemory: READONLY Handle;

    -- Creating and deleting spaces
defaultBase: CARDINAL = LAST[PageOffset];
Create: PROCEDURE [size: PageCount, parent: Handle, base: PageOffset←defaultBase] RETURNS [newSpace: Handle];
CreateUniformSwapUnits: PROCEDURE [size: PageCount←1, parent: Handle];
Delete: PROCEDURE [space: Handle];
DeleteSwapUnits: PROCEDURE [space: Handle];

    -- Mapping spaces
WindowOrigin: TYPE = RECORD [file: File.Capability, base: File.PageNumber];
defaultWindow: WindowOrigin = [File.nullCapability, 0];
MakeReadOnly: PROCEDURE [space: Handle];
Map: PROCEDURE [space: Handle, window: WindowOrigin←defaultWindow];
Remap: PROCEDURE [space: Handle, window: WindowOrigin←defaultWindow];
Unmap: PROCEDURE [space: Handle];

    -- Swapping commands
Activate: PROCEDURE [space: Handle];
Deactivate: PROCEDURE [space: Handle];
ForceOut: PROCEDURE [space: Handle];
Kill: PROCEDURE [space: Handle];

    -- Miscellaneous operations
GetAttributes: PROCEDURE [space: Handle]
        RETURNS [parent, lowestChild, nextSibling: Handle, base: PageOffset, size: PageCount, mapped: BOOLEAN];
GetHandle: PROCEDURE [page: PageNumber] RETURNS [Handle];
GetWindow: PROCEDURE [space: Handle] RETURNS [WindowOrigin];
LongPointer: PROCEDURE [space: Handle] RETURNS [LONG POINTER];
LongPointerFromPage: PROCEDURE [page: PageNumber] RETURNS [LONG POINTER];
MDS: PROCEDURE RETURNS [Handle] = INLINE BEGIN RETURN[mds] END;
PageFromLongPointer: PROCEDURE [lp: LONG POINTER] RETURNS [page: PageNumber];
Pointer: PROCEDURE [space: Handle] RETURNS [POINTER];
VMPageNumber: PROCEDURE [space: Handle] RETURNS [PageNumber];

    -- Signals and errors
Error: ERROR [type: ErrorType];
    ErrorType: TYPE = {invalidHandle, invalidMappingOperation, invalidParameters, invalidWindow, noWindow,
            notApplicableToSwapUnit, spaceTreeTooDeep};
InsufficientSpace: ERROR [available: PageCount];

END.
```

LOG

Time: March 1978 By: McJones Action: Created file
Time: September 14, 1978 10:46 AM By: Lauer Action: Added LongPointerFromPage, PageFromLongPointer
Time: March 7, 1979 10:16 AM By: McJones Action: Eliminated dependence on SpaceInternal; added MakeReadOnly, defaults
Time: July 19, 1979 11:03 AM By: McJones Action: Added uniform swap units; deleted AddressFault, InsufficientPermissions

```
DIRECTORY
    Environment: FROM "Environment" USING [Block, Byte, Word];

Stream: DEFINITIONS =
BEGIN

        --The following types are used by all clients of  Stream
Handle: TYPE = POINTER TO Object;
Byte: TYPE = Environment.Byte;
Word: TYPE = Environment.Word;
SubSequenceType: TYPE = [0..256);
Block: TYPE = Environment.Block;
InputOptions: TYPE = RECORD [
        terminateOnEndPhysicalRecord, signalLongBlock, signalShortBlock, signalSSTChange, signalEndOfStream:
        BOOLEAN];
defaultInputOptions: InputOptions = [FALSE, FALSE, FALSE, FALSE, FALSE];
CompletionCode: TYPE = {normal, endRecord, sstChange, endOfStream};

        --The following are the data transmission operations
GetBlock: PROCEDURE [sH: Handle, block: Block] RETURNS [bytesTransferred: CARDINAL, why: CompletionCode, sst:
        SubSequenceType] = INLINE
    -- Calls sH.get with the appropriate parameters
    BEGIN
    [bytesTransferred, why, sst] ← sH.get[sH, block, sH.options];
    END;            --GetBlock

SetInputOptions: PROCEDURE [sH: Handle, options: InputOptions] = INLINE
    -- Sets the input options in the Object pointed to by sH
    BEGIN
    sH.options ← options;
    END;            --SetInputOptions

PutBlock: PROCEDURE [sH: Handle, block: Block, endPhysicalRecord: BOOLEAN] = INLINE
    -- Calls sH.put with the appropriate parameters
    BEGIN
    sH.put[sH, block, endPhysicalRecord];
    END;            --PutBlock

GetByte: PROCEDURE [sH: Handle] RETURNS [byte: Byte];
GetChar: PROCEDURE [sH: Handle] RETURNS [char: CHARACTER];
GetWord: PROCEDURE [sH: Handle] RETURNS [word: Word];
PutByte: PROCEDURE [sH: Handle, byte: Byte];
PutChar: PROCEDURE [sH: Handle, char: CHARACTER];
PutWord: PROCEDURE [sH: Handle, word: Word];

SendNow: PROCEDURE [sH: Handle] = INLINE
    -- Calls sH.put with a block of no bytes to terminate the physical record
    BEGIN
    bl: Block = [NIL, 0, 0];
    sH.put[sH, bl, TRUE];
    END;            --SendNow

SetSST: PROCEDURE [sH: Handle, sst: SubSequenceType] = INLINE
    -- Changes the current SubSequenceType by calling sH.setSST
    BEGIN
    sH.setSST[sH, sst];
    END;            --SetSST

SendAttention: PROCEDURE [sH: Handle, byte: Byte] = INLINE
    -- Calls sH.sendAttention
    BEGIN
    sH.sendAttention[sH, byte];
    END;            --SendAttention

WaitForAttention: PROCEDURE [sH: Handle] RETURNS [Byte] = INLINE
    -- Waits for an attention by calling sH.waitAttention
    BEGIN
    RETURN[sH.waitAttention[sH]];
    END;            --WaitForAttention
```

```
Delete: PROCEDURE [sH: Handle] = INLINE
    -- Calls sH.delete
    BEGIN
    sH.delete[sH];
    END;            --Delete

        --The following are the signals which can be generated by a Stream
SSTChange: SIGNAL [sst: SubSequenceType, nextIndex: CARDINAL];
EndOfStream: SIGNAL [nextIndex: CARDINAL];
TimeOut: SIGNAL [nextIndex: CARDINAL];
LongBlock: SIGNAL [nextIndex: CARDINAL];
ShortBlock: ERROR;

        --The following are the types used by implementors of filters and transducers
Object: TYPE = RECORD [
        options: InputOptions,
        get: GetProcedure,
        put: PutProcedure,
        setSST: SetSSTProcedure,
        sendAttention: SendAttentionProcedure,
        waitAttention: WaitAttentionProcedure,
        delete: DeleteProcedure];
GetProcedure: TYPE = PROCEDURE [sH: Handle, block: Block, options: InputOptions] RETURNS [bytesTransferred: CARDINAL,
        why: CompletionCode, sst: SubSequenceType];
PutProcedure: TYPE = PROCEDURE [sH: Handle, block: Block, endPhysicalRecord: BOOLEAN];
SetSSTProcedure: TYPE = PROCEDURE [sH: Handle, sst: SubSequenceType];
SendAttentionProcedure: TYPE = PROCEDURE [sH: Handle, byte: Byte];
WaitAttentionProcedure: TYPE = PROCEDURE [sH: Handle] RETURNS [Byte];
DeleteProcedure: TYPE = PROCEDURE [sH: Handle];

END.        -- Stream
```

LOG

Time: March 17, 1978 11:39 AM By: HCL Action: Created file
Time: April 6, 1978 3:24 PM By: HCL Action: Updated to conform to revised Functional Specifications
Time: April 18, 1978 9:58 AM By: HCL Action: Updated for compilation by Alpha release of Mesa 4.0 (LONG POINTER is allowed).
Time: May 5, 1978 2:19 PM By: Lauer Action: Recompiled for storage on "official" directory on CoPilot; added ***GetWord*** *and* ***PutWord***
Time: June 22, 1978 8:57 AM By: Lauer Action: Added "EndOfStream" signal, input option, and completion code.
Time: July 21, 1978 11:31 AM By: HCL Action: moved type 'Block' from Stream to Environment; added GetChar and PutChar
Time: August 14, 1978 2:03 PM By: HCL Action: deleted "DeleteModule" (can be simulated by Runtime.UnNew[GlobalFrame[p]]; moved "DeleteModuleAfterReturn" to Runtime and renamed it "SelfDestruct"
Time: February 22, 1979 2:50 PM By: Dalal Action: moved a large number of procedures from StreamImpl.mesa to this module as INLINEs. Added the parameter sH to all the procedures of Stream.Object. Added ***Delete****.*
Time: March 13, 1979 10:17 AM By: Dalal Action: modified SendAttention and WaitForAttention to take and return a byte of data respectiveely.

```
DIRECTORY
    Mopcodes: FROM "Mopcodes" USING [zMISC];

System: DEFINITIONS =

BEGIN

    -- Universal identifiers
UniversalID: TYPE = PRIVATE RECORD [a, b, c, d: UNSPECIFIED];
nullID: UniversalID = [0, 0, 0, 0];
GetUniversalID: PROCEDURE RETURNS [uid: UniversalID];
FileID: TYPE = RECORD [UniversalID]; -- a useful special case of UniversalID
VolumeID: TYPE = RECORD [UniversalID]; -- a useful special case of UniversalID
    -- Network addresses
    NetworkAddress: TYPE = PRIVATE RECORD [a, b, c, d: UNSPECIFIED]; -- an "exported" type
nullNetworkAddress: NetworkAddress = [0, 0, 0, 0];

    -- Timekeeping facilities
GreenwichMeanTime: TYPE = LONG CARDINAL;
GetGreenwichMeanTime: PROCEDURE RETURNS [gmt: GreenwichMeanTime];

TimerHandle: TYPE = PRIVATE RECORD [LONG UNSPECIFIED]; -- change to (public) [Pulses]
Microseconds: TYPE = LONG CARDINAL;
CreateIntervalTimer: PROCEDURE RETURNS [t: TimerHandle]; -- add = INLINE BEGIN RETURN[[GetClockPulses[]]] END
GetIntervalTime: PROCEDURE [t: TimerHandle] RETURNS [m: Microseconds];

Pulses: TYPE = RECORD [LONG CARDINAL];
GetClockPulses: PROCEDURE RETURNS [p: Pulses] =
    MACHINE CODE BEGIN Mopcodes.zMISC, 11B END;
PulsesToMicroseconds: PROCEDURE [p: Pulses] RETURNS [m: Microseconds];
MicrosecondsToPulses: PROCEDURE [m: Microseconds] RETURNS [p: Pulses];

    -- System power control
PowerOff: PROCEDURE;
SetAutomaticPowerOn: PROCEDURE [time: GreenwichMeanTime, externalEvent: BOOLEAN];
ResetAutomaticPowerOn: PROCEDURE;

END.
```

LOG

Time: April 12, 1978 11:56 AM By: Lauer Action: Created file
Time: May 4, 1978 3:19 PM By: Lauer Action: Fixed for Mesa 4.0; established this file as the "official" version; added NetworkAddresses
Time: May 5, 1978 11:12 AM By: Lauer Action: Added nullID
Time: June 21, 1978 11:23 AM By: Lauer Action: Deleted AllocateSpecificSocket
Time: July 29, 1978 1:07 PM By: Horsley Action: Deleted DeleteIntervalTimer and InvalidTimerHandle
Time: August 11, 1978 10:22 AM By: Lauer Action: Moved TimerHandle from SystemInternal
Time: August 29, 1978 11:26 AM By: Lauer Action: Changed representation of UniversalID to be four words of UNSPECIFIED
Time: March 9, 1979 4:37 PM By: McJones Action: Moved body of NetworkAddress from SystemInternal; changed representation of GreenwichMeanTime and Microseconds to be LONG CARDINALs
Time: May 2, 1979 2:26 PM By: McJones Action: Added new interval timing (Pulses)
Time: May 22, 1979 3:30 PM By: Lauer Action: Replaced NetworkAddress with a record containing only two LONG UNSPECIFIED's

```
-- This is the interface for the UTxxx display and keyboard

DIRECTORY
    Space: FROM "Space" USING [Handle, nullHandle];

UserTerminal: DEFINITIONS
IMPORTS Space =
BEGIN


    -- Type definitions --
Coordinate: TYPE = RECORD [x, y: INTEGER];
CursorArray: TYPE = ARRAY [0..16) OF WORD;
BitmapHandle: TYPE = PRIVATE RECORD [UNSPECIFIED];


    -- Constants (modulo display hardware) --

screenWidth: READONLY CARDINAL[0..32767];
screenHeight: READONLY CARDINAL[0..32767];
pixelsPerInch: READONLY CARDINAL;
numberOfKeys: READONLY CARDINAL;


    -- Errors and signals --

InvalidHandle: ERROR;
BeingDisplayed: ERROR;


    -- The keyboard --

KeyArray: READONLY LONG DESCRIPTOR FOR READONLY ARRAY OF WORD;


    -- Mouse functions --

SetMousePosition: PROCEDURE [newMousePosition: Coordinate];
    -- Sets mouse to specified coordinates

GetMousePosition: PROCEDURE RETURNS [mousePosition: Coordinate];
    -- Returns current position of mouse


    -- Cursor functions --

GetCursorPosition: PROCEDURE RETURNS [cursorPosition: Coordinate];
    -- Returns current position of cursor

SetCursorPosition: PROCEDURE [newCursorPosition: Coordinate];
    -- Sets cursor to specified coordinates

SetCursorPattern: PROCEDURE [cursorPattern: CursorArray];
    -- Does the obvious thing, assumes 16x16 cursor

GetCursorPattern: PROCEDURE RETURNS [cursorPattern: CursorArray];


    -- Bitmap functions --

SetBorder: PROCEDURE [oddLines, evenLines: [0..377B]];
```

```
    -- Sets pattern for Dandelion's "border" registers

CreateBitmap: PROCEDURE [space: Space.Handle ← Space.nullHandle] RETURNS [bitmap: BitmapHandle];

DestroyBitmap: PROCEDURE [bitmap: BitmapHandle];

GetAttributes: PROCEDURE [bitmap: BitmapHandle] RETURNS [space: Space.Handle, isDisplayed: BOOLEAN];

GetCurrentBitmap: PROCEDURE RETURNS [h: BitmapHandle];

BlinkDisplay: PROCEDURE;

DisplayBitmap: PROCEDURE [bitmap: BitmapHandle];

RemoveBitmap: PROCEDURE [bitmap: BitmapHandle];

ActivateBitmap: PROCEDURE [bitmap: BitmapHandle];
    -- Analogous to Space.Activate

DeactivateBitmap: PROCEDURE [bitmap: BitmapHandle];
    -- Ditto comment above

KillBitmap: PROCEDURE [bitmap: BitmapHandle];
    -- Ditto comment above

WaitForScanLine: PROCEDURE [scanLine: INTEGER];
    -- Returns when display reaches specified place

END.
```

LOG
Time: March 1, 1979 2:00 PM By: Gobbel Action: created file
Time: May 29, 1979 3:58 PM By: Gobbel Action: Make version 2 - largely redone
Time: June 6, 1979 4:29 PM By: Gobbel Action: Fix some mistakes
Time: August 10, 1979 5:02 PM By: Forrest Action: Change ScreeHeight and ScreenWidth to subranges [0..32767]

```
DIRECTORY
    File: FROM "File",
    System: FROM "System";

Volume: DEFINITIONS =
BEGIN

    --Volume identifiers
ID: TYPE = System.VolumeID;     -- = RECORD[System.UniversalID]
nullID: ID = ID[System.nullID];

systemID: READONLY ID; -- ID of system volume
SystemID: PROCEDURE RETURNS [ID] ≔ INLINE BEGIN RETURN [systemID] END;

    --Volume addressing
maxPagesPerVolume: LONG CARDINAL ≔ 8388607;   -- = 2^23 - 1
PageCount: TYPE = LONG CARDINAL; -- Simulates TYPE = [0..maxPagesPerVolume];
    firstPageCount: PageCount = 0;
    lastPageCount: PageCount = maxPagesPerVolume;

    --Attributes of Volumes
GetAttributes: PROCEDURE [volume: ID] RETURNS [volumeSize, freePageCount: PageCount, rootFile: File.Capability];
GetLabelString: PROCEDURE [volume: ID, s: STRING];
SetRootFile: PROCEDURE [volume: ID, file: File.Capability];

    --Mounting and Unmounting volumes
GetNext: PROCEDURE [volume: ID]
        RETURNS [nextVolume: ID];
Open: PROCEDURE [volume: ID];
Close: PROCEDURE [volume: ID];

    --Locating volumes
IsOnServer: PROCEDURE [volume: ID, netAddress: System.NetworkAddress];

    --Signals and Errors generated by the Volume interface
InsufficientSpace: ERROR;
InvalidLabelString: ERROR [s: STRING];
Unknown: ERROR [volume: ID];
END.
```

LOG

Time: May 5, 1978 9:16 AM By: Lauer Action: Created file from Pilot Functional Spec
Time: June 21, 1978 3:22 PM By: Lauer Action: changed value of maxPagesPerVolume to specification; changed "nullVolume" to "nullID", "EnumerateNextVolume" to "GetNext"; added "SystemID"
Time: March 7, 1979 11:13 AM By: Redell Action: Changed PageCount to LONG CARDINAL as allowed by Mesa 5.0
Time: March 19, 1979 4:36 PM By: Redell Action: Added Open operation.
Time: March 30, 1979 5:34 PM By: Redell Action: Exported variable 'systemID'.
Time: April 2, 1979 3:43 PM By: Horsley Action: Removed return variable name from SystemID since it invokes a compiler bug.
Time: May 22, 1979 3:28 PM By: Lauer Action: Exported "systemID" as a READONLY
Time: August 10, 1979 4:53 PM By: Lauer Action: Fixed GetLabelString to atke volume

```
VMMapLog: DEFINITIONS =

BEGIN

-- Map log: Pilot-to-Debugger communication of virtual memory backing storage binding

Descriptor: TYPE = MACHINE DEPENDENT RECORD [ -- must be allocated in permanently resident memory
    self: Entry, -- description of virtual memory used by Pilot to access log
    writer: EntryPointer, -- next entry Pilot will write
    reader: EntryPointer, -- next entry debugger will examine
    limit: EntryPointer, -- entry following last word of ring buffer
    patchTable: LONG POINTER TO PatchTable];

-- Above writer, reader, and limit fields are relative to LOOPHOLE[LongPointerFromPage[self.page], EntryBasePointer]

EntryBasePointer: TYPE = LONG BASE POINTER TO RECORD [UNSPECIFIED];

EntryPointer: TYPE = EntryBasePointer RELATIVE POINTER [0..177777B] TO Entry;

Entry: TYPE = MACHINE DEPENDENT RECORD [ -- describe mapping of run of virtual pages with contiguous "backing storage"
    page: CARDINAL, -- starting virtual page number
    count: [1..4096], -- number of pages (12 bits)
    writeProtected: BOOLEAN,
    fill: [0..1], -- allow for expansion of variant tag
    filePoint: SELECT kind: * FROM -- corresponding "backing storage"
        nil => NULL, -- to nothing
        alto31 => [ -- to Alto file with given fp, starting at given filePage
            fp: AltoFP,
            filePage: CARDINAL],
        pilot31 => [ -- to contiguous disk addresses on Pilot-formatted model 31
            vda: CARDINAL, -- ((track*2) + head)*12 + sector
            fid: PilotFID, -- fid of every label in run
            filePage: CARDINAL], -- page number of first page in run
        disk => [ -- to contiguous pages on Pilot physical volume
            driveTag: CARDINAL, -- as returned by DiskChannel.GetDriveTag
            diskPage: LONG CARDINAL, -- i.e. DiskChannel.DiskPageNumber
            labelCheck: CARDINAL], -- as computed by FilePageLabel.LabelCheckSum
        ENDCASE];

Pilot31Label: TYPE = MACHINE DEPENDENT RECORD [ -- must be consistent with Pilot's FileInternal.Label
    word0: UNSPECIFIED, -- don't care
    fid: PilotFID, -- constant for each page in a run
    word3: UNSPECIFIED, -- don't care
    page: CARDINAL, -- increments for each page in a run
    word5, word6, word7: UNSPECIFIED]; -- don't care

PilotFID: TYPE = MACHINE DEPENDENT RECORD [UNSPECIFIED, UNSPECIFIED];

AltoFP: TYPE = MACHINE DEPENDENT RECORD [
    serial: RECORD [UNSPECIFIED, UNSPECIFIED],
    leaderDA: UNSPECIFIED];


-- Patch table: Debugger-to-Pilot communication of patches to (writeProtected) code segments

PatchTable: TYPE = MACHINE DEPENDENT RECORD [
    limit: PatchTableEntryPointer, -- entry following last entry in use
    maxLimit: PatchTableEntryPointer, -- entry following last allocated entry
    entries: ARRAY [0..0) OF PatchTableEntry]; -- last entry with given address overrides

-- Above limit, maxLimit fields are relative to LOOPHOLE[@entries[0], PatchTableEntryBasePointer]

PatchTableEntryBasePointer: TYPE = LONG BASE POINTER TO RECORD [UNSPECIFIED];

PatchTableEntryPointer: TYPE = PatchTableEntryBasePointer RELATIVE POINTER [0..177777B] TO PatchTableEntry;

PatchTableEntry: TYPE = MACHINE DEPENDENT RECORD [
    address: LONG POINTER,
    value: UNSPECIFIED];

END.
```

LOG

Time: June 23, 1978 8:43 AM By: McJones Action: Created file
Time: July 12, 1978 3:39 PM By: Johnsson Action: Counter proposal
Time: August 1, 1978 4:52 PM By: McJones Action: Refinements to BASE-RELATIVE *types*
Time: August 28, 1978 10:08 AM By: McJones Action: Added patch table
Time: August 16, 1979 10:42 PM By: McJones Action: Added writeProtected, disk variant to Entry

```
DIRECTORY
    Environment: FROM "Environment" USING [Base];

Zone: DEFINITIONS =
BEGIN

    -- Types known by clients of the zone allocator
Alignment: TYPE = {a1, a2, a4, a8, a16};
Base: TYPE = Environment.Base; -- for upward compatibility
Handle: TYPE = PRIVATE RECORD[LONG UNSPECIFIED];
SegmentHandle: TYPE = PRIVATE RECORD[UNSPECIFIED];
BlockSize: TYPE = CARDINAL [0..77777B--largest block is 2^15-1 words--];

    -- Useful constants
minimumNodeSize: READONLY BlockSize;
nullSegment: READONLY SegmentHandle;
nil: READONLY Base RELATIVE POINTER;

    -- Status returned by Zone operations
Status: TYPE = {okay, noRoomInZone, nonEmptySegment, storageOutOfRange, zoneTooSmall, segmentTooSmall,
        invalidNode, invalidZone, invalidSegment, nodeLoop};

    -- Node allocation and deallocation
MakeNode: PROCEDURE [zH: Handle, n: BlockSize, alignment: Alignment ← a1] RETURNS [node: Base RELATIVE POINTER, s:
        Status];
FreeNode: PROCEDURE [zH: Handle, p: LONG POINTER] RETURNS [s: Status];
SplitNode: PROCEDURE [zH: Handle, p: LONG POINTER, n: BlockSize] RETURNS [s: Status];
NodeSize: PROCEDURE [p: LONG POINTER] RETURNS [n: BlockSize];

    -- Zone and segment management
Create: PROCEDURE [storage: LONG POINTER, length: BlockSize, zoneBase: Base, threshold: BlockSize ← minimumNodeSize,
        checking: BOOLEAN ← FALSE] RETURNS [zH: Handle, s: Status];
AddSegment: PROCEDURE [zH: Handle, storage: LONG POINTER, length: BlockSize] RETURNS [sH: SegmentHandle, s:
        Status];
RemoveSegment: PROCEDURE [zH: Handle, sH: SegmentHandle] RETURNS [storage: LONG POINTER, s: Status];

    -- Zone and segment attributes
GetAttributes: PROCEDURE [zH: Handle] RETURNS [zoneBase: Base, threshold: BlockSize, checking: BOOLEAN, storage:
        LONG POINTER, length: BlockSize, next: SegmentHandle];
SetChecking: PROCEDURE [zH: Handle, checking: BOOLEAN] RETURNS [s: Status];
GetSegmentAttributes: PROCEDURE [zH: Handle, sH: SegmentHandle] RETURNS [storage: LONG POINTER, length: BlockSize,
        next: SegmentHandle];

END.
```

LOG

Time: February 21, 1979 10:50 AM By: Lauer Action: Created file
Time: July 18, 1979 4:19 PM By: Knutsen Action: Promoted Alignment *and* MakeAlignedNode *(renamed* MakeNode*) from* ZoneExtension.
Time: timeStamp By: yourName Action: shortDescription

```
-- Pilot.config (last edited by: Dalal on: August 9, 1979  4:11 PM)
PACK -- resident code; some modules correspond to PACK within a smaller configuration
    PilotControl, -- PilotKernel
    DebuggerMicrocode,
    DriverControl,
    UserTerminalImpl, -- UserTerminalDriver
    DiskDriverSharedImpl; -- Model31Driver

Pilot: CONFIGURATION
    IMPORTS PilotClient
    EXPORTS BitBlt, ByteBlt, Dialup, File, Forgot, Log, NetworkStream, Process, RS232C, RS232CManager, Runtime,
            Socket, Space, Stream, System, UserTerminal, Volume, Zone,
            -- Communication private -- OISCPDefs, StatsDefs, StatsPrivateDefs,
            -- for PUP package only -- BufferDefs, ByteBltDefs, CommUtilDefs, DriverDefs,
            PupDefs =

BEGIN

PilotKernel;

DebuggerMicrocode;

DriverControl;

UserTerminalDriver;

Model31Driver;

RS232CIO;

END.
```

LOG

Time: June 30, 1978 11:15 AM By: McJones Action: Created file
Time: July 14, 1978 1:46 PM By: McJones Action: VFSPrograms subsumes FPT, Swapper, VMM
Time: July 21, 1978 1:47 PM By: Horsley Action: Updated to real thing for Pilot 1.0d
Time: August 11, 1978 2:17 PM By: Horsley Action: Added DisplayPrivate to EXPORTs
Time: August 30, 1978 3:54 PM By: Horsley Action: Added VMMode
Time: August 30, 1978 4:12 PM By: Horsley Action: Added Forgot, Log to EXPORTs
Time: August 31, 1978 11:45 AM By: Lauer Action: Fixed syntax error on Forgot
Time: August 31, 1978 7:06 PM By: Horsley Action: Added Control
Time: September 8, 1978 4:02 PM By: Purcell Action: Added Communication
Time: September 15, 1978 11:29 AM By: Horsley Action: Added Communication exports
Time: September 28, 1978 10:25 AM By: Horsley Action: Added RS232C exports
Time: October 18, 1978 11:52 AM By: Dalal Action: Added BufferDefs exports
Time: February 27, 1979 2:48 PM By: Horsley Action: Pilot Kernel
Time: April 9, 1979 1:41 PM By: Gobbel Action: Added UserTerminalDriver, deleted DisplayPrivate
Time: April 9, 1979 7:01 PM By: Gobbel Action: Added DeviceControl
Time: April 17, 1979 1:40 PM By: Gobbel Action: DeviceControl renamed to DriverControl
Time: April 30, 1979 9:49 PM By: Lauer Action: Exported Zone
Time: May 31, 1979 2:38 PM By: Lauer Action: Exported ZoneExtension (temporary unitl this can be consolidated with Zone);
Time: June 6, 1979 3:26 PM By: Schwartz Action: Replaced RS232CConfig by RS232CIO.
Time: June 7, 1979 8:00 AM By: Schwartz Action: Added Dialup and RS232CManager exports
Time: June 15, 1979 5:30 PM By: Gobbel Action: Added BitBlt to EXPORTs
Time: July 9, 1979 2:12 PM By: Gobbel Action: Added ByteBlt to EXPORTs
Time: July 18, 1979 12:48 PM By: McJones Action: Moved DebuggerMicrocode here from MesaRuntime
Time: July 31, 1979 3:13 PM By: Forrest Action: Flush Zone Extension
Time: August 9, 1979 4:11 PM By: Dalal Action: Added Communication stuff

```
-- TestPilot.config (last edited by: McJones on: August 15, 1979  4:05 PM)
PACK -- resident code; some modules correspond to PACK within a smaller configuration
    PilotControl, -- TestPilotKernel
    DebuggerMicrocode,
    DriverControl,
    UserTerminalImpl, -- UserTerminalDriver
    DiskDriverSharedImpl; -- Model31Driver

TestPilot: CONFIGURATION
    IMPORTS PilotClient
    EXPORTS
        -- public -- BitBlt, ByteBlt, Dialup, File, Forgot, Log, NetworkStream, Process, RS232C, RS232CManager,
        Runtime, Socket, Space, Stream, System, UserTerminal, Volume, Zone,
        -- Communication private -- OISCPDefs, StatsDefs, StatsPrivateDefs,
        -- for PUP package only -- BufferDefs, ByteBltDefs, CommUtilDefs, DriverDefs,
        PupDefs,
        -- for test puposes only -- AltoDisk, DiskChannel, IOCS, KernelFile, PilotSwitches, ProcessInternal, ResidentHeap,
        ResidentMemory, RuntimeInternal, Snapshot, SpecialFile, SpecialSpace, SPPInternal, SystemInternal, Utilities =

BEGIN

TestPilotKernel;

DebuggerMicrocode;

DriverControl;

UserTerminalDriver;

Model31Driver;

RS232CIO;

END.


LOG

Time: August 31, 1978  7:11 PM  By: Horsley    Action: Created file
Time: September 3, 1978  9:07 AM    By: Lauer      Action: Added SpecialSpace to EXPORTS
Time: September 3, 1978  6:16 PM    By: Lauer      Action: Added Log to EXPORTS
Time: September 15, 1978  11:33 AM By: Horsley    Action: Added Communication EXPORTS
Time: September 28, 1978  10:28 AM By: Horsley    Action: Added RS232C EXPORTS
Time: September 28, 1978  6:21 PM   By: McJones   Action: Added SpecialFile to EXPORTS
Time: October 18, 1978  11:50 AM    By: Dalal Action: Added BufferDefs to EXPORTS
Time: February 27, 1979  2:48 PM    By: Horsley    Action: Added Pilot Kernel
Time: April 9, 1979  1:44 PM        By: Gobbel     Action: Added UserTerminalDriver, deleted DisplayPrivate
Time: April 9, 1979  7:01 PM        By: Gobbel     Action: Added DeviceControl
Time: April 17, 1979  1:41 PM       By: Gobbel     Action: DeviceControl renamed DriverControl
Time: April 30, 1979  9:49 PM       By: Lauer      Action: Added Zone to EXPORTS
Time: June 4, 1979  4:18 PM         By: Lauer      Action: Added ZoneExtension (temporary until this can be consolidated with Zone)
        and ResidentHeap to EXPORTS
Time: June 7, 1979  8:46 AM         By: Schwartz   Action: RS232CConfig renamed RS232CIO; added Dialup and RS232CManager to
        EXPORTS
Time: June 15, 1979  5:28 PM        By: Gobbel     Action: Added BitBlt to EXPORTS
Time: June 25, 1979  1:03 PM        By: McJones    Action: Added AltoDisk to EXPORTS
Time: July 9, 1979  2:16 PM By: Gobbel     Action: Added ByteBlt to EXPORTS
Time: July 18, 1979  12:48 PM       By: McJones    Action: Moved DebuggerMicrocode here from MesaRuntime
Time: August 1, 1979  4:06 PM       By: Gobbel     Action: Removed ZoneExtension from EXPORTS
Time: August 9, 1979  10:24 AM By: Dalal Action: Added the Communication stuff
Time: August 15, 1979  4:05 PM By: McJones  Action: Added DiskChannel,KernelFile,PilotSwitches,Snapshot to EXPORTS
```

```
DIRECTORY
  Environment: FROM "Environment" USING [PageNumber],
  File: FROM "File" USING [ID];

BootChannel: DEFINITIONS =

BEGIN

Create: PROCEDURE [POINTER TO Location] RETURNS [Handle];
  -- Create boot channel from specification of device and device address

Location: TYPE = MACHINE DEPENDENT RECORD [
  device: UNSPECIFIED, -- Device.Number, e.g. controller, drive
  vp: SELECT OVERLAID * FROM
    alto => [serial: LONG CARDINAL, daPage1: CARDINAL],
    disk => [id: DiskFileID],
    any => [a, b, c, d, e, f: UNSPECIFIED],
    ENDCASE];

DiskFileID: TYPE = MACHINE DEPENDENT RECORD [
  fID: File.ID,
  da: DiskAddress]; -- first page

DiskAddress: TYPE = MACHINE DEPENDENT RECORD [UNSPECIFIED, UNSPECIFIED]; -- real disk address

Handle: TYPE = POINTER TO Rep;

Rep: TYPE = MACHINE DEPENDENT RECORD [
  Open: PROCEDURE [Handle, Direction] RETURNS [dControllerState: LONG DESCRIPTOR FOR ARRAY OF WORD],
    -- Set up specified channel for transfers in indicated direction,
    --   and return descriptor for state block of controller used for this channel (allows careful save/restore)
    --   (when germ resides in first 64K, controllerState need not be LONG)
  TransferPage: PROCEDURE [Handle, Environment.PageNumber],
    -- Initiate transfer of next page, in direction established by Open
  Wait: PROCEDURE [Handle]
    -- Wait for any pending i/o (including necessary retries) to complete
  ];

Direction: TYPE = {read, write, checkLabels};

END.
```

LOG

Time: December 18, 1978 9:34 AM By: McJones Action: Created file
Time: July 30, 1979 4:32 PM By: McJones Action: Added substructure and disk variant to Location, checkLabels to Direction

```
DIRECTORY
    Environment: FROM "Environment" USING [PageNumber],
    OISProcessorFace: FROM "OISProcessorFace" USING [DeviceHandle, DeviceType, nullDeviceHandle];

DiskChannel: DEFINITIONS =
BEGIN

-- Drives
Drive: TYPE = OISProcessorFace.DeviceHandle;
nullDrive: Drive = OISProcessorFace.nullDeviceHandle;
GetNextDrive: PROCEDURE [prev: Drive] RETURNS [Drive];
GetDriveAttributes: PROCEDURE [drive: Drive] RETURNS [deviceType: OISProcessorFace.DeviceType, nPages:
        DiskPageCount];
GetDriveTag: PROCEDURE [drive: Drive] RETURNS [tag: CARDINAL];
SetDriveTag: PROCEDURE [drive: Drive, tag: CARDINAL];
DiskPageCount: TYPE = LONG CARDINAL;
DiskPageNumber: TYPE = LONG CARDINAL;

-- Channels
Handle: TYPE = PRIVATE RECORD [UNSPECIFIED];
nullHandle: Handle = [0];
Create: PROCEDURE [drive: Drive, completion: CompletionHandle] RETURNS [Handle];
Delete: PROCEDURE [channel: Handle];
GetAttributes: PROCEDURE [channel: Handle] RETURNS [drive: Drive];

-- Completion objects
CompletionHandle: TYPE = PRIVATE RECORD [LONG UNSPECIFIED];
CreateCompletionObject: PROCEDURE RETURNS [CompletionHandle];

-- Suspend, Abort, and Restart
Suspend: PROCEDURE [channel: Handle];
Idle: PROCEDURE [channel: Handle];
Restart: PROCEDURE [channel: Handle];

-- Data Transfer
IORequestHandle: TYPE = LONG POINTER TO IORequest;
IORequest: TYPE = RECORD [ -- Provided by client, lifetime = duration of I/O operation

    -- fields defining the IO Request
    channel: Handle,
    diskPage: DiskPageNumber,
    memoryPage: Environment.PageNumber,
    count: DiskPageCount,
    direction: Direction,
    label: PLabel,
    verifyLabel: [0..labelLength], -- number of words of label to check; 0 => read label

    -- returned status of IO Request
    status: CompletionStatus,

    -- private for use by the DiskChannel implementation
    next: IORequestHandle, -- this pointer is smashed by the driver during an IO transaction

    -- private for use by the drivers
    retryCount: PRIVATE INTEGER,
    drive: PRIVATE DriveNumber,
    address: PRIVATE Address
    ];

Address: TYPE = PRIVATE MACHINE DEPENDENT RECORD [
    track: Track,
    head: Head,
    sector: Sector];
Track: TYPE = CARDINAL;
Head: TYPE = [0..256);
Sector: TYPE = [0..256);
DriveNumber: TYPE = CARDINAL;
PLabel: TYPE = LONG POINTER TO Label;
Label: TYPE = MACHINE DEPENDENT RECORD [
    contents: ARRAY [0..labelLength) OF UNSPECIFIED]; -- labels should be quadword aligned
labelLength: CARDINAL = 8;
```

or write (as determined by 'direction')

```
Direction: TYPE = {put, get};
CompletionStatus: TYPE = {goodCompletion, noSuchPage, labelDoesNotMatch, seekFailed, checkError, checksumError,
        hardwareError, notReady};

-- Data Transfer Operations
InitiateIO: PROCEDURE [req: IORequestHandle];
WaitAny: PROCEDURE [completion: CompletionHandle] RETURNS [IORequestHandle];

-- Auxilliary Operations
GetPageAddress: PROCEDURE [channel: Handle, page: DiskPageNumber] RETURNS [Address];
GetPageNumber: PROCEDURE [drive: Drive, page: Address] RETURNS [DiskPageNumber];

END.
```

LOG

Time: January 8, 1979 7:17 PM By: TH Action: Created file to replace RigidDisk
Time: July 23, 1979 3:06 PM By: Gobbel Action: Replaced "reservedForDriver" field of IORequest by "address"
Time: August 2, 1979 2:39 PM By: Redell Action: Simplified interface. Eliminated DriveHandle/DriveID distinction, reformatted Addresses, added GetPageAddress, removed all ERRORs, etc. etc. Prepared for run-of-pages by adding count field to IORequest.

```
DIRECTORY
    DiskChannel: FROM "DiskChannel",
    OISProcessorFace: FROM "OISProcessorFace";

DiskChannelBackend: DEFINITIONS =
BEGIN

-- Drives
DriveHandle: TYPE = LONG POINTER TO DriveObject;
DriveObject: TYPE = RECORD[
    drive: DiskChannel.Drive,
    deviceType: OISProcessorFace.DeviceType,
    nPages: DiskChannel.DiskPageCount,
    tag: CARDINAL,
    requestIO: RequestIOProc, -- Must get all three of these stupid procedure descriptors out of here
    getPageAddress: GetPAProc,
    getPageNumber: GetPNProc,
    next: PRIVATE DriveHandle]; -- for the exclusive use of the DiskChannel implementation
RequestIOProc: TYPE = PROCEDURE[req: DiskChannel.IORequestHandle];
GetPAProc: TYPE = PROCEDURE [page: DiskChannel.DiskPageNumber] RETURNS [DiskChannel.Address];
GetPNProc: TYPE = PROCEDURE [page: DiskChannel.Address] RETURNS [DiskChannel.DiskPageNumber];

RegisterDrive: PROCEDURE [drive: DriveHandle]; --called by driver once for each drive

NotifyIOComplete: PROCEDURE [req: DiskChannel.IORequestHandle]; --called by driver to indicate completion of IO

GetDrive: PROCEDURE [channel: DiskChannel.Handle] RETURNS[DriveHandle]; -- should be an inline

-- Errors
END.
```

LOG

Time: January 15, 1979 2:51 PM By: Horsley Action: Created file
Time: August 15, 1979 5:12 PM By: Redell Action: changed DriveID to Drive
Time: August 16, 1979 10:57 PM By: Redell Action: Added tag field to DriveObject

```
DIRECTORY
    DriverPrograms: FROM "DriverPrograms",
    Model31Programs: FROM "Model31Programs" USING [FindModel31s],
    UserTerminalPrograms: FROM "UserTerminalPrograms" USING [UserTerminalImpl, BitBltImpl];

DriverControl: PROGRAM
    IMPORTS Model31Programs, UserTerminalPrograms
    EXPORTS DriverPrograms =
BEGIN

StartIODrivers: PUBLIC PROCEDURE =
BEGIN
    START UserTerminalPrograms.UserTerminalImpl;
    START UserTerminalPrograms.BitBltImpl;
END;

StartStorageDrivers: PROCEDURE =
BEGIN
    START Model31Programs.FindModel31s;
END;


-- Mainline code --

StartStorageDrivers[];

END.
```

LOG
Time: April 9, 1979 5:54 PM By: Gobbel Action: Created file
Time: April 17, 1979 1:55 PM By: Gobbel Action: Changed name to DriverControl
Time: June 15, 1979 1:52 PM By: Gobbel Action: Added BitBlt, changed UTMisc to UserTerminalImpl
Time: July 3, 1979 3:19 PM By: Gobbel Action: Deleted BitBltPrograms

```
DriverPrograms: DEFINITIONS =
BEGIN

DriverControl: PROGRAM;

StartIODrivers: PROCEDURE;

END.
```

LOG
Time: April 9, 1979 5:54 PM By: Gobbel Action: Created file
Time: April 17, 1979 1:58 PM By: Gobbel Action: Changed name to DriverPrograms

```
DIRECTORY
    ProcessInternal: FROM "ProcessInternal";

IOCS: DEFINITIONS =
BEGIN

    -- IOCB definitions and allocation
IOCBSize: CARDINAL = 16; -- Each page consists of sixteen 16-word IOCBs

IOCB: TYPE = PRIVATE RECORD [
    next: IOCBPtr,
    unused: ARRAY [2..IOCBSize) OF UNSPECIFIED];
IOCBPtr: TYPE = LONG POINTER TO IOCB;

GrabIOCB: PROCEDURE RETURNS [p: IOCBPtr];
ReleaseIOCB: PROCEDURE [p: IOCBPtr];

    -- Condition variables for Naked Notifies
CVPriority: TYPE = ProcessInternal.NakedNotifyPriority;  --i.e., {low, high}
InitializeNakedConditionVariable: PROCEDURE [cv: LONG POINTER TO CONDITION, priority: CVPriority] RETURNS [mask:
        WORD];  --same as ProcessInternal.SetNakedNotifyPointer[AllocateNakedNotifyLevel[priority]]

    -- Device roll call
BackPlaneSlot: TYPE = [0..maxBackPlaneSlots);
maxBackPlaneSlots: CARDINAL = 24;
GetDeviceType: PROCEDURE[slot: BackPlaneSlot] RETURNS [controllerID: WORD, pCSB: LONG POINTER];
END.
```

LOG
Time: June 26, 1978 2:11 PM By: Jarvis Action: Created file
Time: August 29, 1978 2:42 PM By: Lauer Action: Added CVPriority, InitializeNakedCV, Explode, MachineType, GetMachineType
Time: January 17, 1979 12:57 PM By: Horsley Action: Deleted GetMachineType, Explode, Shave, FindTrash, FindZeros, Added GetDevice

```
Model31Programs: DEFINITIONS =
BEGIN

FindModel31s: PROGRAM;

END.
```

LOG
Time: April 9, 1979 5:54 PM By: Gobbel Action: Created file

```
DIRECTORY
    Mopcodes: FROM "Mopcodes";

ProcessInternal: DEFINITIONS =
BEGIN

    -- Naked Notify facilities
SetNakedNotifyPointer: PROCEDURE [level: NakedNotifyLevel, cv: LONG POINTER TO CONDITION];
AllocateNakedNotifyLevel: PROCEDURE [priority: NakedNotifyPriority] RETURNS [level: NakedNotifyLevel, mask: WORD]; --
        Allocates the next available NakedNotify pointer: if priority = high, lowest numbered available level is chosen; if priority = low, highest
        numbered available level is chosen. (These NakedNotify levels correspond to Alto Interrupt levels.)
NakedNotifyLevel: TYPE = [0..15];
NakedNotifyPriority: TYPE = {low, high};

    -- Enabling and disabling interrupts
DisableInterrupts: PROCEDURE = MACHINE CODE BEGIN Mopcodes.zIWDC END;
EnableInterrupts: PROCEDURE = MACHINE CODE BEGIN Mopcodes.zDWDC END;

END.
```

LOG

Time: June 7, 1978 3:50 PM By: Lauer Action: Created file
Time: June 22, 1978 9:07 AM By: yourName Action: Cleaned up fault queue stuff, naked notify stuff.
Time: May 16, 1979 12:24 PM By: Sandman Action: Removed fault queue stuff.

```
DIRECTORY
    Process: FROM "Process";

ProcessPriorities: DEFINITIONS =
BEGIN

diskInterruptPriority: Process.Priority = 5;
END.
```

LOG

Time: February 19, 1979 5:41 PM By: Horsley Action: Created file
Time: time By: name Action: action

```
DIRECTORY
    Environment: FROM "Environment" USING [PageCount],
    File: FROM "File" USING [Capability];

Snapshot: DEFINITIONS =

BEGIN

-- The files passed to OutLoad and InLoad must be bootable (see SpecialFile.MakeBootable)

OutLoad: PROCEDURE [file: File.Capability] RETURNS [inLoaded: BOOLEAN];
    -- Save machine state on file, then return FALSE

InLoad: PROCEDURE [pMicrocode, pGerm: LONG POINTER, countGerm: Environment.PageCount, file: File.Capability];
    -- Load new microcode and germ (if pMicrocode~ = NIL and pGerm~ = NIL, respectively), and restore machine state from file,
            causing OutLoad to return TRUE
    -- Note: if pMicrocode~NIL, it must point to resident memory (e.g. use SpecialSpace.MakeResident)

END.
```

LOG

Time: August 9, 1979 8:51 AM By: McJones Action: Created file

```
DIRECTORY
    File: FROM "File" USING [Capability, PageCount, PageNumber];

SpecialFile: DEFINITIONS =

BEGIN

MakeBootable: PROCEDURE [file: File.Capability, firstPage: File.PageNumber, count: File.PageCount, lastLink: Link]
    RETURNS [link: Link];
    -- Install forward chain through page labels of file, which must be of type tBootFile.  Use specified link in label of last page
    chained (firstPage + count-1), and return link value corresponding to first page chained. Note that this may make normal Pilot
    access to the file considerably less efficient.

Link: TYPE = RECORD [LONG UNSPECIFIED];

MakeUnBootable: PROCEDURE [file: File.Capability];
    -- Remove boots chains; this will eliminate any inefficiency caused by their presence.

InvalidParameters: ERROR;

SetDebuggerFiles: PROCEDURE [debugger, debuggee: File.Capability];
    -- Store pointers (i.e. Links) to the specified files into root of the system volume. Note that the files must be entirely contained
    within (the first physical volume of) the system volume.

END.
```

LOG

Time: July 31, 1979 4:46 PM *By: Redell* *Action: Created file*

```
DIRECTORY
    Space: FROM "Space" USING [Handle, PageCount, PageOffset];

SpecialSpace: DEFINITIONS =

BEGIN

    -- Create space within 64K boundaries (for code)
CreateForCode: PROCEDURE [size: Space.PageCount, parent: Space.Handle, base: Space.PageOffset]
        RETURNS [newSpace: Space.Handle];

    -- Make spaces resident or swappable
MakeResident, MakeSwappable: PROCEDURE [space: Space.Handle];
MakeCodeResident, MakeCodeSwappable: PROCEDURE [frame: PROGRAM];

    -- Determine real memory size
realMemorySize: READONLY Space.PageCount;

    -- Moving the contents of pages by swinging VM map pointers
--(Operation to be defined and implemented in Pilot 3.0)

END.
```

LOG

Time: August 28, 1978 10:07 AM By: Lauer Action: Created file
Time: July 10, 1979 12:06 PM By: Knutsen Action: Make compatible with Teak interfaces
Time: August 9, 1979 6:41 PM By: McJones Action: Added realMemorySize

```
DIRECTORY
    DiskChannel: FROM "DiskChannel" USING [DiskPageCount, DriveHandle],
    LogicalVolume: FROM "LogicalVolume" USING [Type],
    PhysicalVolume: FROM "PhysicalVolume" USING [Descriptor, ID],
    Volume: FROM "Volume" USING [ID, PageCount];

SpecialVolume: DEFINITIONS =
BEGIN

PhysicalVolumeUnknown: ERROR;
InsufficientSpace: ERROR [freePages: DiskChannel.DiskPageCount];
LogicalVolumeUnknown: ERROR ;

GetPhysicalVolumeAttributes: PROCEDURE [PhysicalVolume.ID] RETURNS [PhysicalVolume.Descriptor];
    -- Can ERROR with PhysicalVolumeUnknown
CreatePhysicalVolume: PROCEDURE [drive: DiskChannel.DriveHandle] RETURNS [PhysicalVolume.ID];
    -- Can ERROR with DiskChannel.mumble
CreateLogicalVolume: PROCEDURE [pVid: PhysicalVolume.ID, size: Volume.PageCount, name: STRING, type: LogicalVolume.Type]
    RETURNS [Volume.ID];
    -- Can ERROR with PhysicalVolumeUnknown, InsufficientSpace
ExtendLogicalVolume: PROCEDURE [lVid: Volume.ID, pVid: PhysicalVolume.ID, increment: Volume.PageCount];
    -- Can ERROR with PhysicalVolumeUnknown, LogicalVolumeUnknown, InsufficientSpace
ReCreateLogicalVolume: PROCEDURE [currentID: Volume.ID, name: STRING, type: LogicalVolume.Type] RETURNS [newID:
    Volume.ID];
    -- Can ERROR with LogicalVolumeUnknown
END.
```

LOG

Time: August 3, 1979 1:35 PM By: Forrest Action: Created file
Time: timeStamp By: yourName Action: shortDescription

```
DIRECTORY
    Environment: FROM "Environment" USING [Long, PageNumber];

Utilities: DEFINITIONS =

BEGIN

LongMove: PROCEDURE [pSource: LONG POINTER, size: CARDINAL, pSink: LONG POINTER];
    -- Move the contents of the size words starting at pSource to the size words starting at pSink
    -- Unlike InlineDefs.LongCOPY, overlapping blocks do not cause replication of source words.
    -- Ideally this too would be implemented as a MACHINE CODE

LongPointerFromPage: PROCEDURE [page: Environment.PageNumber] RETURNS [LONG POINTER];

PageFromLongPointer: PROCEDURE [lp: LONG POINTER] RETURNS [page: Environment.PageNumber];

ShortCARDINAL: PROCEDURE [cc: LONG UNSPECIFIED] RETURNS [CARDINAL] =
    -- Assume cc IN CARDINAL
    INLINE BEGIN RETURN[LOOPHOLE[cc, Environment.Long].lowbits] END;
END.
```

LOG

Time: July 28, 1978 7:44 PM By: Horsley Action: Created file
Time: July 29, 1978 2:02 PM By: Horsley Action: Added GetMachineType and routines from defunct Miscellaneous
Time: August 29, 1978 2:35 PM By: Lauer Action: Added PageFromLongPointer
Time: August 30, 1978 4:41 PM By: Horsley Action: Named argument in PageFromLongPointer
Time: September 22, 1978 2:09 PM By: McJones Action: Made LongCopy into machine code, changed ShortCARDINAL to avoid direct machine op dependency
Time: March 1, 1979 4:31 PM By: McJones Action: Deleted CompareAsLongCardinals, LongCopy

```
-- PilotKernel.config (last edited by: McJones on: August 20, 1979  5:04 PM)
PACK -- resident code: each module corresponds to a PACK in a smaller configuration
    PilotControl, -- Control
    FilerControl, -- Storage
    BootSwapCross, -- MesaRuntime
    ResidentHeapImpl, -- Misc
    IOCSImpl; -- IO

PilotKernel: CONFIGURATION
    IMPORTS DebuggerSwap, DriverPrograms, PilotClient
    EXPORTS
        -- public -- ByteBlt, File, Forgot, Log, NetworkStream, Process, Runtime, Socket, Space, Stream, System,
        Volume, Zone,
        -- for Drivers Only -- DiskChannelBackend, IOCS, PilotSwitches, ProcessInternal, ResidentHeap, RuntimeInternal,
        SpecialSpace, Utilities,
        -- Communication private -- OISCPDefs, StatsDefs, StatsPrivateDefs,
        -- for PUP package only -- BufferDefs, ByteBltDefs, CommUtilDefs, DriverDefs,
        PupDefs =

BEGIN

Control;

Storage;

MesaRuntime;

Misc;

Communication;

IO;

END.
```

LOG

```
Time: February 1979    By: Horsley    Action: Created file
Time: March 19, 1979  3:44 PM    By: Horsley    Action: VFS renamed Storage
Time: April 9, 1979  4:22 PM    By: Gobbel    Action: Deleted DisplayPrivate
Time: April 30, 1979  9:49 PM    By: Lauer    Action: Added Zone to EXPORTS
Time: June 4, 1979  4:16 PM    By: Lauer    Action: Added ZoneExtension (temporary) and ResidentHeap to EXPORTS
Time: July 9, 1979  2:14 PM By: Lauer    Action: Added ByteBlt to EXPORTS
Time: July 31, 1979  3:13 PM    By: Forrest    Action: Deleted ZoneExtension from EXPORTS
Time: August 6, 1979  9:37 AM    By: McJones  Action: Added DebuggerSwap to IMPORTS
Time: August 9, 1979  4:13 PM    By: Dalal Action: Added Communication stuff
Time: August 15, 1979  4:14 PM  By: McJones  Action: Added PilotSwitches to EXPORTS
```

```
-- TestPilotKernel.config (last edited by: McJones on: August 20, 1979  5:03 PM)
PACK -- resident code; each module corresponds to a PACK in a smaller configuration
    PilotControl, -- Control
    FilerControl, -- Storage
    BootSwapCross, -- MesaRuntime
    ResidentHeapImpl, -- Misc
    IOCSImpl; -- IO

TestPilotKernel: CONFIGURATION
    IMPORTS DebuggerSwap, DriverPrograms, PilotClient
    EXPORTS
        -- public -- ByteBlt, File, Forgot, Log, NetworkStream, Process, Runtime, Socket, Space, Stream, System,
        Volume, Zone,
        -- for Drivers Only -- DiskChannelBackend, IOCS, PilotSwitches, ProcessInternal, ResidentHeap, RuntimeInternal,
        SpecialSpace, Utilities,
        -- Communication private -- OISCPDefs, StatsDefs, StatsPrivateDefs,
        -- for PUP package only -- BufferDefs, ByteBltDefs, CommUtilDefs, DriverDefs,
        PupDefs,
        -- for insiders only -- AltoDisk, DiskChannel, KernelFile, ResidentMemory, Snapshot, SpecialFile, SPPInternal,
        SystemInternal =

BEGIN

Control;

Storage;

MesaRuntime;

Misc;

Communication;

IO;

END.


LOG

Time: February 1979    By: Horsley    Action: Created File
Time: March 19, 1979  3:45 PM By: Horsley    Action: Changed VFS to Storage
Time: April 9, 1979  4:23 PM By: Gobbel  Action: Deleted DisplayPrivate
Time: April 30, 1979  9:49 PM    By: Lauer    Action: Added Zone to EXPORTS
Time: June 4, 1979  4:17 PM    By: Lauer    Action: Added ZoneExtension (temporary) and ResidentHeap to EXPORTS
Time: June 25, 1979  1:07 PM    By: McJones  Action: Added AltoDisk to EXPORTS
Time: July 9, 1979  2:15 PM By: Gobbel  Action: Added ByteBlt to EXPORTS
Time: July 31, 1979  3.13 PM    By: Forrest  Action: Deleted ZoneExtension from EXPORTS
Time: August 6, 1979  9:37 AM  By: McJones  Action: Added DebuggerSwap to IMPORTS
Time: August 9, 1979  10:21 AM  By: Dalal Action: Added real Communication and stuff
Time: August 15, 1979  4:02 PM  By: McJones  Action: Added KernelFile,PilotSwitches,Snapshot to EXPORTS
```

```
DIRECTORY
  AltoDefs: FROM "AltoDefs",
  AltoFileDefs: FROM "AltoFileDefs";

AltoDisk: DEFINITIONS =

BEGIN OPEN AltoFileDefs, AltoDefs;

  -- standard disk
  nDisks: CARDINAL = 1;
  nHeads: CARDINAL = 2;
  nTracks: CARDINAL = 203;
  nSectors: CARDINAL = 12;

  -- physical disk address
  DA: PRIVATE TYPE = MACHINE DEPENDENT RECORD [
    sector: [0..17B],
    track: [0..777B],
    head, disk: [0..1],
    restore: [0..1]];

  -- DAs with special meaning
  InvalidDA: DA = DA[17B,777B,1,1,1];

  -- disk header
  DH: TYPE = MACHINE DEPENDENT RECORD [
    packID: CARDINAL,
    diskAddress: DA];

  -- file identifier
  FID: TYPE = MACHINE DEPENDENT RECORD [
    version: CARDINAL,
    serial: SN];

  -- disk label
  DL: TYPE = MACHINE DEPENDENT RECORD [
    next, prev: DA,
    blank: UNSPECIFIED,
    bytes: CARDINAL,
    page: CARDINAL,
    fileID: FID];

  -- disk final status
  DFS: PRIVATE TYPE = {
    CommandComplete, HardwareError,
    CheckError, IllegalSector};

  -- disk status word
  DS: PRIVATE TYPE = MACHINE DEPENDENT RECORD [
    sector: [0..17B],
    done: [0..17B],
    seekFailed: [0..1],
    seekInProgress: [0..1],
    notReady: [0..1],
    dataLate: [0..1],
    noTransfer: [0..1],
    checksumError: [0..1],
    finalStatus: DFS];

  -- useful status configurations
  DSfree: CARDINAL = 1;  DSfake: CARDINAL = 3;  DSdone: CARDINAL = 17B;
  DSmaskStatus: DS = DS[0,DSdone,1,0,1,1,0,1,LAST[DFS]];
  DSgoodStatus: DS = DS[0,DSdone,0,0,0,0,0,0,CommandComplete];
  DSfakeStatus: DS = DS[0,DSfake,0,0,0,0,0,0,CommandComplete];
  DSfreeStatus: DS = DS[0,DSfree,0,0,0,0,0,0,CommandComplete];

  -- disk subcommands
  DSC: PRIVATE TYPE = {DiskRead, DiskCheck, DiskWrite};

  -- hardware disk command
  DC: PRIVATE TYPE = MACHINE DEPENDENT RECORD [
```

```
    seal: BYTE,
    header, label, data: DSC,
    seek, exchange: [0..1]];

  CBptr: TYPE = POINTER TO CB;

-- Disk Command block
  --  NOTE: this format must be used with CB.DC.seal = 111B, since the high-bytes
  --    of the LONG POINTERS to the label and data are inserted before the header
CB: TYPE = PRIVATE MACHINE DEPENDENT RECORD [
    nextCB: POINTER TO CB,
    status: DS,
    command: DC,
    headerAddress: PUBLIC POINTER TO DH,
    labelAddressLow: WORD,
    dataAddressLow: WORD,
    normalWakeups: WORD,
    errorWakeups: WORD,
    labelAddressHigh: BYTE, dataAddressHigh: BYTE,
    header: PUBLIC DH,

    label: PUBLIC DL, -- remainder of CB record used only by software
    page: PUBLIC CARDINAL,
    zone: PUBLIC POINTER TO CBZ];

  nCB: CARDINAL = 3;  -- minimun for full disk speed
  lCBZ: CARDINAL = SIZE[CBZ] + nCB*(SIZE[CB] + SIZE[CBptr]);

  -- Note: if there are n CBs, there are n+1 entries in the
  -- cbQueue (an extra one contains a NIL to mark the end).
  -- The extra one is represented by queueVec: ARRAY [0..1)
  -- and thus is included in SIZE[CBZ].

  CBZptr: TYPE = POINTER TO CBZ;

  CBZ: TYPE = PRIVATE MACHINE DEPENDENT RECORD [
    checkError: PUBLIC BOOLEAN,
    errorCount: PUBLIC [0..77777B],
    info: PUBLIC POINTER,
    cleanup: PUBLIC PROCEDURE[CBptr],
    errorDA: PUBLIC vDA,
    currentPage: PUBLIC CARDINAL,
    currentBytes: PUBLIC CARDINAL,
    normalWakeups: WORD,
    errorWakeups: WORD,
    cbQueue: DESCRIPTOR FOR ARRAY OF CBptr,
    qHead, qTail: CARDINAL,
    queueVec: ARRAY [0..1) OF CBptr];
    -- the queue vector starts at queueVec.
    -- after the queue vector there follows
    -- ARRAY OF CB, the CBs for the zone.


  -- Procedures in DiskIO

  RealDA: PROCEDURE [v:vDA] RETURNS [DA];
  VirtualDA: PROCEDURE [da:DA] RETURNS [vDA];

  SetDisk: PROCEDURE [POINTER TO DISK];
  GetDisk: PROCEDURE RETURNS [POINTER TO DISK];
  ResetDisk: PROCEDURE RETURNS [POINTER TO DISK];

  ResetWaitCell: PROCEDURE;
  SetWaitCell: PROCEDURE [POINTER TO WORD] RETURNS [POINTER TO WORD];

  DDC: TYPE = RECORD [
    cb: CBptr,
    ca: LONG POINTER,
    da: vDA,
    page: PageNumber,
    fp: POINTER TO FP,
    restore: BOOLEAN,
```

```
    action: vDC];

  DoDiskCommand: PROCEDURE [arg:POINTER TO DDC];

  RetryCount: CARDINAL = 8;
  RetryableDiskError: SIGNAL [cb:CBptr];
  UnrecoverableDiskError: SIGNAL [cb:CBptr];

  CBinit: TYPE = {clear,dontClear};

  InitializeCBstorage: PROCEDURE [
    zone:CBZptr, nCBs:CARDINAL, page:PageNumber, init:CBinit];

  GetCB: PROCEDURE [zone:CBZptr, init:CBinit] RETURNS [cb:CBptr];

  CleanupCBqueue: PROCEDURE [zone:CBZptr];

  DiskCheckError: SIGNAL [page:PageNumber];

  DiskRequestOption: TYPE = {swap, update, extend};

  DiskRequest: TYPE = RECORD [
    ca: LONG POINTER,
    da: POINTER TO vDA,
    firstPage: PageNumber,
    lastPage: PageNumber,
    fp: POINTER TO FP,
    fixedCA: BOOLEAN,
    action, lastAction: vDC,
    signalCheckError: BOOLEAN,
    option: SELECT OVERLAID DiskRequestOption FROM
      swap => [desc: POINTER TO DiskPageDesc],
      update => [cleanup: PROCEDURE[CBptr]],
      extend => [lastBytes: CARDINAL],
      ENDCASE];

  DiskPageDesc: TYPE = RECORD [
    prev, this, next: vDA,
    page: PageNumber,
    bytes: CARDINAL];

  SwapPages: PROCEDURE [arg:POINTER TO swap DiskRequest]
    RETURNS [page:PageNumber, byte:CARDINAL];

  END.
```

LOG

Time: September 14, 1978 6:23 PM By: Redell Action: Created File from Alto/Mesa DiskDefs converted to LONG POINTER disk controller.
Time: September 19, 1978 6:16 PM By: Redell Action: Moved high-bytes of label/data pointers to before header to avoid clobbering them when using tricky Alto file label-chaining.
Time: August 1, 1979 10:28 AM By: Knutsen Action: Gave module a Pilot-standard header paragraph.

```
--NOTE: The format of File Page Labels as defined here CAN NEVER BE CHANGED
--       without invalidating all outstanding Pilot volumes.

DIRECTORY
    Environment: FROM "Environment" USING [PageOffset],
    File: FROM "File" USING [ID, nullCapability, PageNumber, Type],
    Inline: FROM "Inline" USING [BITXOR, LongNumber],
    System: FROM "System"; -- fields of System.UniversalID/nullID used indirectly via File

FilePageLabel: DEFINITIONS IMPORTS Inline SHARES File, System =

BEGIN

Label: TYPE = MACHINE DEPENDENT RECORD[
    fileID: File.ID,                                    -- set in label of every page
    filePageLo: CARDINAL, filePageHi: [0..256),         -- 24 bit Page Number, set in label of every page
    immutable: BOOLEAN,                                 -- valid only in label of page 0
    temporary: BOOLEAN,                                 -- valid only in label of page 0
    zeroSize: BOOLEAN,                                  -- valid only in label of page 0
    bootFile: BOOLEAN,                                  -- valid in label of every page
    pad1: [0..16),                                      -- always zero
    var: SELECT OVERLAID * FROM
        normal => [type: File.Type, pad2: UNSPECIFIED], -- valid in label of every page of a non-boot file
        bootFile => [bootChainLink: LONG CARDINAL],     -- valid in label of every page of a boot file
        ENDCASE];

nullLabel: FilePageLabel.Label = [
    fileID: nullID,
    filePageLo: 0,
    filePageHi: 0,
    immutable: FALSE,
    temporary: FALSE,
    zeroSize: FALSE,
    bootFile: FALSE,
    pad1: 0,
    var: normal[[0], 0]];

nullID: File.ID = File.nullCapability.fID;

LN: PRIVATE PROCEDURE [lowbits, highbits: CARDINAL] RETURNS [LONG CARDINAL] =
  MACHINE CODE BEGIN END; -- side-step compiler bug

GetFilePage: PROCEDURE [label: LONG POINTER TO Label] RETURNS [File.PageNumber] = INLINE
  BEGIN
  RETURN[LN[lowbits: label.filePageLo, highbits: label.filePageHi]]
  END;

SetFilePage: PROCEDURE [label: LONG POINTER TO Label, fpn: File.PageNumber] = INLINE
  BEGIN
  label.filePageLo ← LOOPHOLE[fpn, Inline.LongNumber].lowbits;
  label.filePageHi ← LOOPHOLE[fpn, Inline.LongNumber].highbits
  END;

GetType: PROCEDURE [label: LONG POINTER TO Label] RETURNS [File.Type] = INLINE
  BEGIN RETURN [IF label.bootFile THEN tBootFile ELSE label.type] END;

SetType: PROCEDURE [label: LONG POINTER TO Label, type: File.Type] = INLINE
  BEGIN IF NOT (label.bootFile ← type = tBootFile) THEN label.type ← type END;

tBootFile: PRIVATE File.Type = [16]; -- Must agree with FileTypes

LabelCheckSum: PROCEDURE [label: LONG POINTER TO Label, offset: Environment.PageOffset]
        RETURNS [checkSum: CARDINAL] = INLINE
  BEGIN
  filePage: LONG CARDINAL = LN[lowbits: label.filePageLo, highbits: label.filePageHi];
  adjFilePage: Inline.LongNumber = LOOPHOLE[filePage-offset];
  RETURN [Inline.BITXOR[label.fileID.a, Inline.BITXOR[label.fileID.b, Inline.BITXOR[label.fileID.c, Inline.BITXOR[label.fileID.d,
    Inline.BITXOR[adjFilePage.lowbits, adjFilePage.highbits]]]]]]
  END;

END.
```

LOG

Time: July 21, 1979 4:32 PM By: Redell Action: Created file

```
DIRECTORY
    File: FROM "File" USING [Capability, PageNumber, Type],
    SpecialFile: FROM "SpecialFile" USING [Link],
    VMMapLog: FROM "VMMapLog" USING [Entry],
    Volume: FROM "Volume" USING [ID];

KernelFile: DEFINITIONS =

BEGIN

DeleteTemps: PROCEDURE [volume: Volume.ID];
    -- Delete all temporary files on given volume

GetBootLocation: PROCEDURE [file: File.Capability] RETURNS [device: DeviceHandle, link: SpecialFile.Link];
    -- Return device handle and disk address of page 0 of given boot file
DeviceHandle: TYPE = RECORD [UNSPECIFIED]; -- LOOPHOLEd to OISProcessorFace.DeviceHandle

GetFilePoint: PROCEDURE [pEntry: LONG POINTER TO VMMapLog.Entry, pFile: POINTER TO File.Capability, filePage:
      File.PageNumber];

    -- Set pEntry↑ to description of run of backing-store pages beginning with given filePage; page field of returned Entry is
         garbage

GetNextFile: PROCEDURE [volume: Volume.ID, file: File.Capability] RETURNS [nextFile: File.Capability];
    -- Enumerate files on given volume (started with and ends with File.nullCapability)

GetRootFile: PROCEDURE [type: File.Type, volume: Volume.ID] RETURNS [file: File.Capability];
    -- Get file with maxPermissions of given (Root)type from volume root page

OpenVolume: PROCEDURE[drive: CARDINAL, initFlag: BOOLEAN, rebuildFlag: BOOLEAN];
    -- Open (or init or rebuild) volume (should this work by volumeID too?)

Pin: PROCEDURE [file: File.Capability];
    -- Pin file descriptor and page groups of this file into file cache

END.
```

LOG

Time: August 28, 1978 10:05 PM By: Purcell Action: Created file as SpecialFile
Time: September 28, 1978 10:05 PM By: Purcell Action: Added Pin
Time: October 19, 1978 10:49 AM By: Purcell Action: CR 20.103: remove FileType dependancies (removed PutRootFile)
Time: August 16, 1979 8:38 PM By: Redell Action: Changed name to Kernel File; changed GetFilePoint

```
DIRECTORY
    Environment: FROM "Environment";

ResidentMemory: DEFINITIONS =
BEGIN

    -- Allocation of resident memory for I/O and miscellaneous use
Allocate: PROCEDURE [where: Location, pages: CARDINAL] RETURNS [LONG POINTER];
Location: TYPE = {first64K, hyperspace};

    -- Allocation of resident memory within MDS
AllocateMDSPages: PROCEDURE [pages: CARDINAL] RETURNS [POINTER];
FreeMDSPages: PROCEDURE [base: POINTER, pages: CARDINAL];
END.
```

LOG

Time: June 9, 1978 11:04 AM By: Lauer Action: Created file
Time: June 22, 1978 11:38 AM By: Lauer Action: Separated allocation from MDS and elsewhere; added FreeMDSPages; moved Pin and Unpin to module "Special"

```
DIRECTORY
    BootChannel: FROM "BootChannel" USING [DiskFileID, Location],
    Environment: FROM "Environment" USING [PageCount, PageNumber],
    Space: FROM "Space" USING [Handle, WindowOrigin];

StoragePrograms: DEFINITIONS =

BEGIN

-- Programs to be STARTed by PilotControl

FilerControl: PROGRAM;

SwapperControl: PROGRAM [pageBuffer: Environment.PageNumber, countBuffer: Environment.PageCount,
        pMapLogDesc: LONG POINTER];

FileImpl: PROGRAM [bootFile: POINTER TO disk BootChannel.Location, pDebuggerIDs: POINTER TO BootFileIDs]
    RETURNS [debugClass: DebugClass, debuggerDevice: UNSPECIFIED];
    -- If bootFile.device~ = nullDevice then set system volume to logical volume containing bootFile.id.da and return debugClass of
            that volume. If debugClass = debuggerDebugger or our debugger is not installed then return
            debuggerDevice = nullDevice, else return device and boot file id's from our debugger's system volume.

BootFileIDs: TYPE = MACHINE DEPENDENT RECORD [microcode, germ, pilot, debugger, debuggee: BootChannel.DiskFileID];

DebugClass: TYPE = {normal, debugger, debuggerDebugger};

nullDevice: UNSPECIFIED = -1; -- should be equated to something in ProcessorFace

VMMControl: PROGRAM [countVM: Environment.PageCount, pMapLogDesc: LONG POINTER];

-- Initialization of virtual memory database

DescribeSpace: PROCEDURE [options: SpaceOptions, page: Environment.PageNumber, count: Environment.PageCount, window:
        Space.WindowOrigin];
    -- Add to VM database (space and region caches)

SpaceOptions: TYPE = RECORD [
    initiallyResident, special, pinRegionD, error, simpleSpace, emptyInterval, createRegion, pinned, mapped, createSpace,
            subspace, pinSpaceD, mStoreDeallocate: BOOLEAN];

--    in      special pinRegionD error   simpl   empty   region  pin     map     space   subsp   pinSD   mstore

createSimpleSpace: SpaceOptions =
    [       FALSE,  FALSE,      TRUE,   FALSE,  TRUE,   FALSE,  TRUE,   FALSE,  FALSE,  TRUE,   FALSE,  TRUE,       FALSE];

noOp: SpaceOptions =
    [       FALSE,  FALSE,      FALSE,  FALSE,  FALSE,  FALSE,  FALSE,  FALSE,  FALSE,  FALSE,  FALSE,  FALSE,      FALSE];

error: SpaceOptions =
    [       FALSE,  FALSE,      FALSE,  TRUE,   FALSE,  FALSE,  FALSE,  FALSE,  FALSE,  FALSE,  FALSE,  FALSE,      FALSE];

free: SpaceOptions =
    [FALSE, FALSE,  FALSE,      FALSE,  FALSE,  FALSE,  FALSE,  FALSE,  FALSE,  FALSE,  FALSE,  FALSE,  TRUE];

empty: SpaceOptions =
    [       FALSE,  FALSE,      FALSE,  FALSE,  FALSE,  TRUE,   FALSE,  FALSE,  FALSE,  FALSE,  FALSE,  FALSE,      FALSE];

HandleFromPage: PROCEDURE [page: Environment.PageNumber] RETURNS [Space.Handle];
    -- Return handle corresponding to space or parent (not subSpace) starting at given page

SuperFromPage: PROCEDURE [page: Environment.PageNumber] RETURNS [Space.Handle, Environment.PageNumber];
    -- Return handle and first page number of superSpace of space or parent (not subSpace) starting at given page

-- Other

ReplacementProcess: PROCEDURE [threshold: Environment.PageCount];
    -- FORK (and Detach) after all DescribeSpace's with option.mStoreDeallocate = TRUE

END.
```

LOG

Time: July 14, 1978 10:01 AM By: McJones Action: Created file
Time: July 17, 1978 5:30 PM By: McJones Action: Added countReal to SwapperControl, VMMControl

Time: August 10, 1978 5:13 PM	By: Purcell	Action: Added DesribeSpace
Time: August 16, 1978 8:01 PM	By: Purcell	Action: Added pMapLogDesc
Time: August 29, 1978 11:33 AM	By: McJones	Action: Added ReplacementProcess; added pMapLogDesc to SwapperControl
Time: September 29, 1978 5:23 PM	By: McJones	CR20.45: Added UnmapDataSpaces
Time: July 31, 1979 11:16 AM	By: McJones	Action: Added boot file id's to FileImpl
Time: August 23, 1979 9:12 AM	By: Knutsen	Action: Improved readability of SpaceOptions table.

```
DIRECTORY
    Environment: FROM "Environment" USING [maxCARDINAL];

SystemInternal: DEFINITIONS =

BEGIN

    -- Processor identifiifiers
ProcessorID: TYPE = PRIVATE MACHINE DEPENDENT RECORD [
    element: ElementType,
    series: CARDINAL];      -- you should not need access fields of ProcessorID unless you are the Ethernet driver!
    -- Note: The bits are stored in a ProcessorID LOW ORDER BITS FIRST, just as if it were a Mesa LONG CARDINAL

SeriesType: TYPE = {altoFP, ethernet, ois};
ElementType: TYPE = RECORD [SELECT COMPUTED SeriesType FROM
    ethernet => [net: [0..256), host: [0..256)], -- series = ethernetSeries
    ois => [CARDINAL], -- series IN OISProcessorSeries
    ENDCASE];
OISProcessorSeries: TYPE = [0..LAST[CARDINAL]-1);
        -- All OIS processors are assigned ProcessorID's by the factory in a series defined by this subrange
ethernetSeries: CARDINAL = LAST[CARDINAL]-1;
        -- This series of ProcessorID's is reserved for identifiying system elements (Alto's, D0's, Dorado's, etc.) which are attached to the Ethernet
        and for which an Ethernet address must be derivable.
altoFPSeries: CARDINAL = LAST[CARDINAL];
        -- NO ProcessorID's of this series will be issued; this series is only for recognizing UniversalID's which contain Alto FP's (see below)
nullProcessorID: ProcessorID = [element: [ois[0]], series: 0];  --this code must be kept consistent with the communications
        interfaces!
broadcastProcessorID: ProcessorID =  [element: [ois[Environment.maxCARDINAL]], series: Environment.maxCARDINAL]; -- a
        special ProcessorID used for broadcasting messages.  This code must be kept consistent with the communications interfaces!

GetProcessorID: PROCEDURE RETURNS [p: ProcessorID];
        -- Returns the ProcessorID by which this system element is known to software, Xerox Wire, Ethernet,  etc.

    -- Alto file pointers (FP's)
AltoFP: TYPE = RECORD [ --this type is to allow Pilot to access Alto files from the Alto disk
    serial: LONG INTEGER,
    leaderDA: CARDINAL];

    -- Universal identifiers
UniversalID: TYPE = PRIVATE RECORD [
    series: CARDINAL,
    extension: SELECT COMPUTED SeriesType FROM
        altoFP => [fp: AltoFP], -- UniversalID.series = altoFPSeries
        ethernet, ois => [a, b, c: UNSPECIFIED], -- (UniversalID.series IN OISProcessorSeries) OR (UniversalID.series = ethernetSeries)
        ENDCASE]; -- you should not need to access the fields of a UniversalID unless you are trying to recover an AltoFP from it.
nullID: ois UniversalID = [series: 0, extension: ois[a: 0, b: 0, c: 0]];

    -- Signal to be raised if an unimplemented feature is encountered
Unimplemented: SIGNAL;

END.
```

LOG

Time: April 12, 1978 11:56 AM By: HCL Action: Created file
Time: May 4, 1978 3:19 PM By: Lauer Action: Fixed for Mesa 4.0; established this file as the "official" version; added NetworkAddresses
Time: May 5, 1978 11:12 AM By: Lauer Action: Added nullID
Time: June 21, 1978 10:03 AM By: Lauer Action: Corrected bugs in OISProcessorSeries, UniversalID; changed TimerHandle per request from Horsley, added nullNetworkAddress and nullProcessorID
Time: August 11, 1978 10:22 AM By: Lauer Action: Added broadcastProcessorID; moved TimerHandle to System
Time: March 9, 1979 1:45 PM By: McJones Action: Moved NetworkAddress to System
Time: July 19, 1979 10:32 AM By: Knutsen Action: broadcastHostNumber, nullHostNumber *removed from* System, *put into* SystemInternal.
Time: July 19, 1979 10:32 AM By: Knutsen Action: broadcastHostNumber, nullHostNumber *removed from* System, *put into* SystemInternal.
Time: August 10, 1979 5:07 PM By: Forrest Action: Started migration of Forgot.unimplemented to SystemINternal.unimplemented.

```
-- Storage.config (last edited by: McJones on: August 20, 1979  4:38 PM)
PACK -- resident code; each module corresponds to a PACK in a smaller configuration
    FilerControl, -- Filer
    SwapperControl; -- Swapper

Storage: CONFIGURATION
    IMPORTS DiskChannel, Forgot, PilotSwitches, Process, ResidentMemory, Runtime, RuntimeInternal, System,
    SystemInternal, Utilities
    EXPORTS AltoDisk, File, KernelFile, Space, SpecialFile, SpecialSpace, StoragePrograms, Volume =

BEGIN

Filer;

FileMgr;

Swapper;

VMMgr;

END.
```

LOG

Time: July 21, 1978 1:26 PM By: Horsley Action: Created file
Time: August 1, 1978 1:55 PM By: McJones Action: Added RuntimeInternals, Utilities to IMPORTs
Time: August 18, 1978 4:33 PM By: McJones Action: Dropped SimpleSpace from EXPORTs
Time: September 3, 1978 8:09 AM By: Lauer Action: Added SpecialSpace to EXPORTs
Time: September 28, 1978 6:38 PM By: McJones Action: Added SpecialFile to EXPORTs
Time: March 23, 1979 3:43 PM By: McJones Action: Added DiskChannel, ResidentMemory to IMPORTs
Time: April 14, 1979 2:16 PM By: McJones Action: Added AltoDisk to EXPORTs (temporary, for MesaRuntime)
Time: July 19, 1979 9:28 AM By: McJones Action: Added Forgot, Runtime to IMPORTs
Time: August 15, 1979 4:29 PM By: McJones Action: Added KernelFile to EXPORTs; VMMode => PilotSwitches

```
DIRECTORY
    CachedSpace: FROM "CachedSpace" USING [Desc, Level],
    VM: FROM "VM" USING [Interval, PageNumber];

CachedRegion: DEFINITIONS =

BEGIN

Desc: TYPE = RECORD [
    interval: VM.Interval,
    level: CachedSpace.Level,                   -- maximum level of containing space
    dTemperature: DTemperature,                 -- cache replacement algorithm state (private to impl.)
    dPinned: BOOLEAN,                           -- descriptor ineligible for replacement?
    dDirty: BOOLEAN,                            -- descriptor modified since cached?
    state: State,
    levelMapped: CachedSpace.Level,             -- level of mapping space (assuming state IN Mapped)
    beingRemapped: BOOLEAN];                    -- (assuming state IN Mapped)

DTemperature: TYPE = [0..4);                    -- should have 2^n values

State: TYPE = {
    missing,
    checkedOut,
    unmapped,
    outDead,
    outAlive,
    inSwappableColdest, inSwappable2, inSwappable3, inSwappableWarmest,
    inPinned};                                  -- (only if space or ancestor is pinned)

Mapped: TYPE = State[outDead..inPinned];

Swappable: TYPE = Mapped[outDead..inSwappableWarmest];

Out: TYPE = Mapped[outDead..outAlive];

In: TYPE = Mapped[inSwappableColdest..inPinned];

InSwappable: TYPE = In[inSwappableColdest..inSwappableWarmest];

Operation: TYPE = RECORD [
    ifMissing: ReportSkip,
    ifCheckedOut: ReturnWait,
    afterForking: ReturnWait,
    vp: SELECT action: * FROM
        activate => [why: ActivatePagefault],
        age => NULL,
        clean => [andWriteProtect: BOOLEAN],
        deactivate => NULL,
        flush => NULL,
        get => [andResetDDirty: BOOLEAN, pDescResult: POINTER TO Desc],
        kill => NULL,
        map => [level: CachedSpace.Level],
        pin => NULL,
        remapA => [firstClean: BOOLEAN],
        remapB => [from: POINTER TO CachedSpace.Desc],
        unmap => NULL,
        unpin => NULL,
        unused15, unused16 => [u: UNSPECIFIED],
        ENDCASE];

ActivatePagefault: TYPE = {activate, pagefault};

ReportSkip: TYPE = {report, skip};

ReturnWait: TYPE = {return, wait};
```

```
-- Standard region operations
-- name:        mumble Operation    = [ifMissing,  ifCheckedOut,  afterForking,  action[other parameters]]
activate:       activate Operation  = [report,     return,        return,         activate[why: activate]];
pagefault:      activate Operation  = [report,     return,        return,         activate[why: pagefault]];
age:            age Operation       = [skip,       return,        return,         age[]];
deactivate:     deactivate Operation                = [skip,       return,         return,        deactivate[]];
flush:          flush Operation     = [skip,       wait,          wait,           flush[]];
forceOut:       clean Operation     = [skip,       wait,          wait,           clean[andWriteProtect: FALSE]];
kill:           kill Operation      = [report,     wait,          return,         kill[]];
pin:            pin Operation       = [report,     wait,          wait,           pin[]];
unmap:          unmap Operation     = [report,     wait,          wait,           unmap[]];
unpin:          unpin Operation     = [report,     wait,          return,         unpin[]]; -- actually ifMissing = >error
writeProtect:                       clean Operation                = [skip,        wait,          wait,          clean[andWriteProtect:
    TRUE]];

Outcome: TYPE = RECORD [SELECT kind: * FROM
    ok => NULL,
    notePinned => [levelMax: CachedSpace.Level], -- only from unpin
    regionDMissing => NULL,
    regionDDirty => NULL,
    spaceDMissing => [level: CachedSpace.Level],
    error => [state: CachedRegion.State],
    ENDCASE];

Apply: PROCEDURE [pageMember: VM.PageNumber, operation: Operation]
        RETURNS [outcome: Outcome, pageNext: VM.PageNumber];
    -- Sets pageNext to desc.interval.page + desc.interval.count if there exists a cached desc with pageMember in desc.interval;
            else sets pageNext to (smallest) desc.interval.page if there exists a cached desc with
            desc.interval.page>pageMember; else sets pageNext to pageTop

pageTop: VM.PageNumber = LAST[VM.PageNumber];

Insert: PROCEDURE [desc: Desc] RETURNS [descVictim: Desc];
    -- If descVictim.dDirty, then descVictim must be written through to higher level database
    -- desc.dPinned and desc.dDirty are significant; desc.dTemperature is ignored

END.
```

Notes

The level and levelMapped fields can be more compactly encoded by observing that if it is defined, levelMapped is IN (0..level]; we can define the following:

LM: TYPE = {l0, l1m1, l2m1, l2m2, l3m1, l3m2, l3m3, . . .};

So 4 levels requires only 3 bits; the 4 bits currently used for level and levelMapped would allow 6 levels.

To complete the scheme, we define:

LevelFromLM: ARRAY LM OF Level = [0, 1, 2, 2, 3, 3, 3, . . .]; and
LevelMappedFromLM: ARRAY LM OF Level = [dontcare, 1, 1, 2, 1, 2, 3, . . .];

LOG

Time: March 1978 By: McJones Action: Created file
Time: June 7, 1978 12:22 PM By: McJones Action: DTemperature had closed upper bound
Time: June 20, 1978 1:16 PM By: McJones Action: Added get, getResetDDirty operations
Time: June 23, 1978 4:34 PM By: McJones Action: Dropped CheckOut/CheckIn
Time: August 1, 1978 4:00 PM By: McJones Action: Removed pageNext from Outcome; dropped probe, added unusedXX
Time: August 4, 1978 1:32 PM By: McJones Action: Added beingRemapped; remap => remapA + remapB
Time: September 10, 1978 1:20 PM By: McJones Action: Added Swappable, notePinned
Time: March 7, 1979 1:24 PM By: McJones CR20.48: Added andWriteProtect to clean Operation

```
DIRECTORY
    Space: FROM "Space" USING [WindowOrigin],
    VM: FROM "VM" USING [Interval, PageCount, PageNumber];

CachedSpace: DEFINITIONS =

BEGIN

Level: TYPE = [0..4);                                   -- see CachedRegion.Desc for packing considerations

Handle: TYPE = RECORD [level: Level, page: VM.PageNumber];   -- the implemenation of a Space.Handle

handleVM: Handle = [level: 0, page: 0];                 -- the root of the space tree
handleNull: Handle = [level: 0, page: LAST[VM.PageNumber]];   -- not a space

State: TYPE = {missing, unmapped, mapped, beingRemapped};

DataOrFile: TYPE = {data, file};

Desc: TYPE = RECORD [
    interval: VM.Interval,
    level: Level,
    dPinned: BOOLEAN,                                   -- descriptor ineligible for replacement?
    dDirty: BOOLEAN,                                    -- descriptor modified since cached?
    pinned: BOOLEAN,                                    -- TRUE only if space or ancestor is mapped
    state: State,
    writeProtected: BOOLEAN,                            -- window.file is immutable or doesn't have write permission
    dataOrFile: DataOrFile,
    pageRover: VM.PageNumber,                           -- for Space.Create with default base
    vp: SELECT OVERLAID * FROM
        short => NULL,
        long => [
            window: WindowOrigin,
            countMapped: VM.PageCount],
        ENDCASE];

PDesc: TYPE = LONG POINTER TO Desc;

WindowOrigin: TYPE = Space.WindowOrigin;

-- Interpretation of handles by these operations is not "strict": page may be any page of space

Insert: PROCEDURE [pDescVictim: PDesc, pDesc: PDesc];
    -- Upon return, if pDescVictim.dDirty, then pDescVictim must be written through to higher level database
    -- pDesc.dPinned and pDesc.dDirty are significant

Delete: PROCEDURE [space: Handle];

Get: PROCEDURE [pDescResult: PDesc, space: Handle];

Update: PROCEDURE [pDesc: PDesc] RETURNS [found: BOOLEAN];
    -- pDesc.dPinned is significant; cached desc.dDirty is set as side-effect

END.
```

Notes

It is necessary to avoid page faults within the monitor implementing this interface. In a system with a resident and a nonresident frame heap, it is important to prevent long parameter or result records from being allocated in the nonresident frame heap. When the stack is increased to 16 words, all the above parameter/result records would fit on the stack even with the by-reference changed to by-value. In the mean time, it is up to the caller to provide references to resident storage.

LOG

Time: March 1978 By: McJones Action: Created file
Time: August 1, 1978 4:21 PM By: McJones Action: Removed dataOrFile from state
Time: August 11, 1978 11:48 AM By: McJones Action: Added beingRemapped to State
Time: September 10, 1978 1:23 PM By: McJones Action: Added pinned field to Desc
Time: February 1, 1979 1:59 PM By: McJones CR20.169: Added pageRover (and variants to Desc)
Time: March 5, 1979 2:25 PM By: McJones Action: Added Handle, handleVM

```
DIRECTORY
    File: FROM "File",
    FileInternal: FROM "FileInternal";

FileCache: DEFINITIONS =

BEGIN

GetFilePtrs: PROCEDURE [count: CARDINAL, fileID: File.ID]
  RETURNS [success: BOOLEAN, fD: FileInternal.FilePtr];

ReturnFilePtrs: PROCEDURE [count: CARDINAL, fD: FileInternal.FilePtr];

SetFile: PROCEDURE [fd: FileInternal.Descriptor, pinned: BOOLEAN];

FlushFile: PROCEDURE [fileID: File.ID];

GetPageGroup: PROCEDURE [fileID: File.ID, filePage: File.PageNumber]
  RETURNS [success: BOOLEAN, pg: FileInternal.PageGroup];

SetPageGroup: PROCEDURE [fileID: File.ID, group: FileInternal.PageGroup, pinned: BOOLEAN];

END.
```

LOG

Time: April 15, 1978 3:46 PM By: Redell Action: Created file

```
DIRECTORY
File: FROM "File" USING [delete, grow, ID, PageCount, PageNumber, Permissions, read, shrink, Type, write],
Volume: FROM "Volume" USING [ID],
VolumeInternal: FROM "VolumeInternal" USING [Location, PageNumber];

FileInternal: DEFINITIONS =

BEGIN

maxPermissions: File.Permissions = File.read + File.write + File.grow + File.shrink + File.delete;

AllocationStatus: TYPE = {free, busy};

FilePtr: TYPE = POINTER TO Descriptor;

Descriptor: TYPE = RECORD[
    fileID: File.ID,
    volumeID: Volume.ID,
    body: SELECT location: VolumeInternal.Location FROM
        remote => NULL,
        local => [
            immutable: BOOLEAN,
            temporary: BOOLEAN,
            size: File.PageCount,
            type: File.Type],
        ENDCASE];

PageGroup: TYPE = RECORD[
    filePage: File.PageNumber,
    volumePage: VolumeInternal.PageNumber,
    nextFilePage: File.PageNumber];

Operation: TYPE = {read, write, readLabel, writeLabel, verifyLabel, readLabelAndData, writeLabelsAndData};

END.
```

LOG

Time: May 13, 1978 2:49 PM By: Redell Action: Created file
Time: June 23, 1978 3:18 PM By: Redell Action: Broke off volume-related items into VolumeInternal
Time: March 7, 1979 4:20 PM By: Redell Action: Moved definition of filer operations in from old FilePageTransferInternal.
Time: July 21, 1979 4:15 PM By: Redell Action: Moved definition of Label out to FilePageLabel.

```
-- FilePageTransfer.mesa (last edited by: Redell on: March 7, 1979  4:11 PM) --

DIRECTORY
    Environment: FROM "Environment",
    File: FROM "File",
    FileInternal: FROM "FileInternal";

FilePageTransfer: DEFINITIONS =

BEGIN

Request: TYPE = RECORD[
    file: File.Capability,
    filePage: File.PageNumber,
    memoryPage: Environment.PageNumber,
    count: Environment.PageCount,
    opSpecific: SELECT operation: Operation FROM
        read => [priorityPage: File.PageNumber],
        write => NULL,
        ENDCASE];

Operation: TYPE = FileInternal.Operation[read..write];

Initiate: PROCEDURE [req: Request];

Wait: PROCEDURE RETURNS [Environment.PageNumber];

END.
```

LOG

Time: March 10, 1978 3:09 PM By: Redell Action: Created file

```
DIRECTORY
    FilePageTransfer: FROM "FilePageTransfer";

FilerException: DEFINITIONS =
BEGIN
Report: PROCEDURE [FilePageTransfer.Request];
Await: PROCEDURE RETURNS [FilePageTransfer.Request];
END.
```

LOG

Time: February 27, 1979 9:43 PM By: Redell Action: Created file from old CacheMiss.mesa
Time: timeStamp By: yourName Action: shortDescription

```
-- LabelTransfer.mesa (last edited by: Redell on: August 19, 1979  2:12 PM)

DIRECTORY
    DiskChannel: FROM "DiskChannel" USING [Address, Drive],
    Environment: FROM "Environment" USING [PageNumber],
    File: FROM "File" USING [PageNumber],
    FileInternal: FROM "FileInternal" USING [Descriptor, Operation, PageGroup],
    FilePageLabel: FROM "FilePageLabel" USING [Label],
    VolumeInternal: FROM "VolumeInternal" USING [PageNumber];

LabelTransfer: DEFINITIONS =

BEGIN

Operation: TYPE = FileInternal.Operation[readLabel..writeLabelsAndData];

ReadLabel: PROCEDURE [
  file: FileInternal.Descriptor,
  filePage: File.PageNumber,
  volumePage: VolumeInternal.PageNumber] RETURNS [FilePageLabel.Label];

ReadRootLabel: PROCEDURE [
  drive: DiskChannel.Drive,
  rootPage: VolumeInternal.PageNumber] RETURNS [label: FilePageLabel.Label, labelValid: BOOLEAN];

WriteLabels: PROCEDURE [
  file: FileInternal.Descriptor,
  pageGroup: FileInternal.PageGroup];

VerifyLabels: PROCEDURE [
  file: FileInternal.Descriptor,
  pageGroup: FileInternal.PageGroup] RETURNS [labelsValid: BOOLEAN];

ReadLabelAndData: PROCEDURE [
  file: FileInternal.Descriptor,
  filePage: File.PageNumber,
  volumePage: VolumeInternal.PageNumber,
  memoryPage: Environment.PageNumber] RETURNS [FilePageLabel.Label];

WriteLabelsAndData: PROCEDURE [
  file: FileInternal.Descriptor,
  pageGroup: FileInternal.PageGroup,
  memoryPage: Environment.PageNumber,
  type: LabelType ← normal,
  lastLink: DiskChannel.Address ← NULL];

LabelType: TYPE = {normal, chained};

END.
```

LOG

Time: June 15, 1978 11:03 PM By: Redell Action: Created file
Time: March 13, 1979 7:28 PM By: Redell Action: Added ReadRootLabel operation
Time: August 1, 1979 12:13 PM By: Redell Action: Updated to use FilePageLabel
Time: August 13, 1979 3:18 PM By: Redell Action: Changed WriteLabelAndData to do runs of pages with optional chaining; don't loophole drive

```
DIRECTORY
    Space: FROM "Space" USING [Handle, PageCount, PageNumber, WindowOrigin];

SimpleSpace: DEFINITIONS =

BEGIN

-- Change FileMgr (and ?) to use types from Space and drop these definitions:
Handle: TYPE = Space.Handle; -- note intended callers are not in a position to import Space
WindowOrigin: TYPE = Space.WindowOrigin;

-- Operations usable only during Pilot initialization

Location: TYPE = {first64K, mds, hyperspace};

Create: PROCEDURE [count: Space.PageCount, location: Location] RETURNS [handle: Space.Handle];

DisableInitialization: PROCEDURE;
    -- Executed after the last simple space has been created but before the contents of the Swapper space and region caches are
        reflected up into the VMMgr database

-- Operations usable at all times

Page: PROCEDURE [handle: Space.Handle] RETURNS [Space.PageNumber]; -- could be inline ...
    -- Returns starting page number of argument simple space

defaultCount: Space.PageCount = LAST[Space.PageCount];
Map: PROCEDURE [handle: Space.Handle, window: Space.WindowOrigin, andPin: BOOLEAN,
        countMapped: Space.PageCount←defaultCount];
    -- Analogous to Space.Map, except: window.file must exist, must not be immutable (unless window.file lacks write permission),
        and must not end before countMapped, which if defaulted means the size of the simple space
    -- If window = defaultWindow, then we allocate real memory and pin it (no backing file); in this case, andPin and
        countMapped are ignored.

Unmap: PROCEDURE [Space.Handle];
    -- Analogous to Space.Unmap

ForceOut: PROCEDURE [Space.Handle];
    -- Analogous to Space.ForceOut

END.
```

LOG

Time: June 19, 1978 5:50 PM By: McJones Action: Created file
Time: August 7, 1978 11:44 PM By: Purcell Action: Action: Added DisableCreate, parameters type to Create and andPin to Map
Time: August 10, 1978 3:56 PM By: Purcell Action: Moved generalized create to VFSPrograms.DescribeSpace
Time: August 29, 1978 1:17 PM By: McJones Action: Added ForceOut
Time: March 5, 1979 9:48 AM By: McJones Action: Changed definition of handle
Time: August 15, 1979 9:17 AM By: McJones Action: Added countMapped to Map
Time: August 21, 1979 10:18 AM By: Knutsen Action: Added comment to Map (not recompiled)

```
DIRECTORY
    DiskChannel: FROM "DiskChannel" USING [Address, CompletionHandle, Handle],
    Environment: FROM "Environment" USING [PageNumber],
    File: FROM "File" USING [PageNumber],
    FileInternal: FROM "FileInternal" USING [FilePtr, Operation],
    Volume: FROM "Volume" USING [ID],
    PhysicalVolume: FROM "PhysicalVolume" USING [PageNumber, SubVolumeDesc],
    VolumeInternal: FROM "VolumeInternal" USING [PageNumber];

SubVolume: DEFINITIONS =

BEGIN

PageNumber: TYPE = LONG CARDINAL;
PageCount: TYPE = LONG CARDINAL;

Handle: TYPE = LONG POINTER TO Descriptor;

Descriptor: TYPE = RECORD[
    lvID: Volume.ID,
    lvPage: VolumeInternal.PageNumber,
    pvPage: PhysicalVolume.PageNumber,
    nPages: PageCount,
    channel: DiskChannel.Handle];

Find: PROCEDURE [vID: Volume.ID, page: VolumeInternal.PageNumber] RETURNS [success: BOOLEAN, subVolume: Handle];

GetPageAddress: PROCEDURE [vID: Volume.ID, page: VolumeInternal.PageNumber] RETURNS [channel: DiskChannel.Handle,
  address: DiskChannel.Address];

OnLine: PROCEDURE [subVolume: PhysicalVolume.SubVolumeDesc, channel: DiskChannel.Handle];

OffLine: PROCEDURE [vID: Volume.ID];

StartIO: PROCEDURE [op: FileInternal.Operation, subVolume: Handle, subVolumePage: PageNumber, memPage:
  Environment.PageNumber, filePtr: FileInternal.FilePtr, filePage: File.PageNumber, chained: BOOLEAN ← FALSE, link:
  DiskChannel.Address ← NULL];

completion: READONLY DiskChannel.CompletionHandle;

END.
```

LOG

Time: March 1, 1979 10:50 AM By: Redell Action: Created file from old Dlsk.mesa, VolumeCache.mesa, and VolumeInternal.mesa.

Time: March 19, 1979 9:22 AM By: Redell Action: Split Descriptor into onLine and offLine variants.

Time: July 24, 1979 3:26 PM By: Forrest Action: Make subvolume record machine dependent. Add Subvolume Field

Time: July 27, 1979 3:37 PM By: Forrest Action: change subvolume count field to Logical Volume Size Field

Time: August 13, 1979 4:22 PM By: Redell Action: Added PageAddress stuff for boot chains. Removed useless LOOPHOLEs to DiskChannel types.

```
DIRECTORY
    CachedRegion: FROM "CachedRegion" USING [Operation, Outcome],
    VM: FROM "VM" USING [PageNumber];

SwapperException: DEFINITIONS =

BEGIN

Await: PROCEDURE RETURNS [page: VM.PageNumber, operation: CachedRegion.Operation, outcome: CachedRegion.Outcome];

Report: PROCEDURE [page: VM.PageNumber, operation: CachedRegion.Operation, outcome: CachedRegion.Outcome];

END.
```

LOG

Time: March 1978 By: McJones Action: Created file
Time: July 31, 1978 11:03 AM By: McJones Action: Action, Outcome => PageNumber, Operation, Outcome

```
DIRECTORY
    Environment: FROM "Environment" USING [PageCount, PageNumber, PageOffset];

VM: DEFINITIONS =

BEGIN

PageCount: TYPE = Environment.PageCount;

PageNumber: TYPE = Environment.PageNumber;

PageOffset: TYPE = Environment.PageOffset;

Interval: TYPE = RECORD [page: Environment.PageNumber, count: Environment.PageCount];

END.
```

LOG

Time: June 24, 1978 2:09 PM By: McJones Action: Created file

```
DIRECTORY
    System: FROM "System",
    SystemInternal: FROM "SystemInternal",
    Volume: FROM "Volume";

VolumeInternal: DEFINITIONS
    SHARES SystemInternal =
BEGIN

PageNumber: TYPE = LONG CARDINAL;

Descriptor: TYPE = RECORD[
    volumeID: Volume.ID,
    open: BOOLEAN];

Location: TYPE = {local, remote};
DriveHandle: TYPE = RECORD [LONG UNSPECIFIED]; -- LOOPHOLEd to a DiskChannel.DriveHandle.

altoVolume: Volume.ID = [LOOPHOLE[SystemInternal.UniversalID[series: SystemInternal.altoFPSeries, extension: altoFP[fp: [0,
  0]]], System.UniversalID]];

END.
```

LOG

Time: June 23, 1978 2:49 PM By: Redell Action: Created file
Time: August 4, 1978 2:49 PM By: Redell Action: Defined DiskHandle to eliminate compilation dependency on RigidDisk.
Time: August 29, 1978 2:43 PM By: Purcell Action: Loopholed altoVolume from SystemInternal.UniversalID
Time: August 30, 1978 2:39 PM By: Lauer Action: Fixed syntactically incorrect LOOPHOLE of yesterday
Time: March 7, 1979 2:43 PM By: Redell Action: Made PageNumber a LONG CARDINAL, as allowed by Mesa 5.0. Removed physical disk info to SubVolume interface.

```
-- VMMgr.config (last edited by: McJones on: August 20, 1979  6:00 PM)

PACK
    HierarchyImpl,
    MapLogImpl,
    ProjectionImpl,
    SpaceImpl,
    STLeafImpl,
    STreeImpl,
    VMMControl;

VMMgr: CONFIGURATION
    IMPORTS CachedRegion, CachedSpace, File, Forgot, KernelFile, PilotSwitches, Process, Runtime, RuntimeInternal,
            SimpleSpace, SwapperException, Utilities, Volume
    EXPORTS Space, SpecialSpace, StoragePrograms
    CONTROL VMMControl =

BEGIN

-- Put the two instances of STreeImpl in separate configurations to get around debugger Set Octal Context bug

HConfig: CONFIGURATION IMPORTS CachedSpace, STLeaf, Utilities EXPORTS Hierarchy, STree, VMMPrograms =
    BEGIN

    HierarchyImpl[CachedSpace, STree];

    STree ← STreeImpl[];

    END;

PConfig: CONFIGURATION IMPORTS CachedRegion, STLeaf, Utilities EXPORTS Projection, STree, VMMPrograms =
    BEGIN

    ProjectionImpl[CachedRegion, STree];

    STree ← STreeImpl[];

    END;

[Hierarchy, STreeHier: STree, VMMPrograms] ← HConfig[];

[Projection, STreeProj: STree, VMMPrograms] ← PConfig[];

MapLogImpl;

SpaceImpl;

STLeafImpl;

VMMControl[CachedRegion, CachedSpace, Hierarchy, MapLog, Projection, SimpleSpace, STreeHier, STreeProj,
        VMMPrograms, Volume];

END.
```

LOG

Time: June 20, 1978 4:38 PM By: McJones Action: Created file
Time: June 22, 1978 3:44 PM By: McJones Action: ProjectionImpl imported STreeHier
Time: June 26, 1978 10:30 AM By: McJones Action: Added CachedRegion, Projection to VMMControl imports
Time: June 30, 1978 9:14 AM By: McJones Action: Added EXPORT of VMMPrograms
Time: July 10, 1978 1:05 PM By: McJones Action: Added Hierarchy, Volume to VMMControl imports
Time: August 1, 1978 11:01 AM By: McJones Action: Added MapLogImpl, IMPORT of Process
Time: August 3, 1978 3:26 PM By: McJones Action: Added code packing
Time: August 9, 1978 1:25 AM By: Purcell Action: Added SimpleSpace and CachedSpace to VMMControl imports
Time: August 29, 1978 9:16 PM By: McJones Action: Added IMPORT of VMMode
Time: September 6, 1978 10:19 AM By: McJones Action: Added IMPORT of RuntimeInternal
Time: September 15, 1978 5:57 PM By: McJones Action: Added MapLog to VMMControl imports
Time: October 19, 1978 4:02 PM By: McJones Action: Added HConfig and PConfig
Time: August 20, 1979 6:00 PM By: McJones Action: VMMode => PilotSwitches, SpecialFile => KernelFile

```
DIRECTORY
    CachedSpace: FROM "CachedSpace" USING [Desc, Handle, State],
    VM: FROM "VM" USING [Interval, PageCount];

Hierarchy: DEFINITIONS =

BEGIN

Delete: PROCEDURE [handle: CachedSpace.Handle];

FindFirstWithin: PROCEDURE [handle: CachedSpace.Handle, count: VM.PageCount] RETURNS [CachedSpace.Handle];
    -- Return first "n'th cousin" starting within count pages of beginning of space

GetDescriptor: PROCEDURE [pDescResult: POINTER TO CachedSpace.Desc, handle: CachedSpace.Handle]
        RETURNS [validHandle: BOOLEAN];
    -- validHandle iff state~ = missing AND interval.page = space.page

GetInterval: PROCEDURE [handle: CachedSpace.Handle] RETURNS [validHandle: BOOLEAN, interval: VM.Interval];
    -- validHandle iff state~ = missing AND interval.page = space.page

GetState: PROCEDURE [handle: CachedSpace.Handle] RETURNS [CachedSpace.State];
    -- Returns missing if ~GetInterval[space].validHandle

Insert: PROCEDURE [handle: CachedSpace.Handle, count: VM.PageCount];
    -- Sets pinned←FALSE, state←unmapped

Touch: PROCEDURE [handle: CachedSpace.Handle];
    -- May raise NotFound

Update: PROCEDURE [pDesc: POINTER TO CachedSpace.Desc];
    -- Updates cached copy (and sets dDirty←TRUE) if one exists; else updates in stree)

NotFound: ERROR;

END.
```

Notes

Replace GetDescriptor, GetInterval, GetState by GetShort (3 words) and GetLong (11 words)

LOG

Time: April 25, 1978 5:20 PM By: McJones Action: Created file
Time: September 10, 1978 1:31 PM By: McJones Action: Merged ValidHandle with GetXXX, etc.
Time: February 20, 1979 10:48 AM By: McJones CR20.177: Update must preserve dPinned

```
DIRECTORY
    CachedSpace: FROM "CachedSpace" USING [Delete, Desc, Get, Handle, handleNull, Insert, PDesc, State, Update],
    Hierarchy: FROM "Hierarchy",
    STree: FROM "STree" USING [Delete, Desc, Get, Insert, Key, Update],
    VM: FROM "VM" USING [Interval, PageCount],
    VMMPrograms: FROM "VMMPrograms";

HierarchyImpl: PROGRAM
    IMPORTS CachedSpace, STree
    EXPORTS Hierarchy, VMMPrograms =

BEGIN OPEN Hierarchy;

pDescBuff: CachedSpace.PDesc -- must point to pinned storage because of CachedSpace calls
--residentFrames-- = @desc;

desc: CachedSpace.Desc;

LoadDesc: PROCEDURE [handle: CachedSpace.Handle] =
    -- Load matching space descriptor into global variable pDescBuff↑
    BEGIN
    descVictim: CachedSpace.Desc;
    -- Try cache first
    CachedSpace.Get[pDescBuff, handle];
    IF pDescBuff.state~ = missing THEN
        RETURN;
    -- Try full hierarchy second
    [] ← STree.Get[@LOOPHOLE[pDescBuff↑, STree.Desc], [hierarchy[handle]]];
    IF pDescBuff.state = missing THEN
        RETURN;
    -- Cache this (clean) descriptor, possibly displacing a dirty descriptor from the cache
    pDescBuff.dPinned ← FALSE; pDescBuff.dDirty ← FALSE;
    CachedSpace.Insert[@descVictim, pDescBuff];
    IF descVictim.dDirty THEN
        STree.Update[@LOOPHOLE[descVictim, STree.Desc]]
    END;

Delete: PUBLIC PROCEDURE [handle: CachedSpace.Handle] =
    BEGIN
    CachedSpace.Delete[handle];
    STree.Delete[[hierarchy[handle]]]
    END;

FindFirstWithin: PUBLIC PROCEDURE [handle: CachedSpace.Handle, count: VM.PageCount] RETURNS [CachedSpace.Handle] =
    -- Bypasses cache
    BEGIN
    desc: STree.Desc;
    keyNext: STree.Key = STree.Get[@desc, [hierarchy[handle]]];
    IF desc.descH.state~ = missing THEN
        RETURN[[handle.level, desc.descH.interval.page]];
    IF keyNext.handleH.level = handle.level AND keyNext.handleH.page<handle.page + count THEN
        RETURN[keyNext.handleH];
    RETURN[CachedSpace.handleNull]
    END;

GetDescriptor: PUBLIC PROCEDURE [pDescResult: POINTER TO CachedSpace.Desc, handle: CachedSpace.Handle]
        RETURNS [validHandle: BOOLEAN] =
    BEGIN
    LoadDesc[handle];
    pDescResult↑ ← pDescBuff↑;
    RETURN[pDescBuff.state~ = missing AND pDescBuff.interval.page = handle.page]
    END;

GetInterval: PUBLIC PROCEDURE [handle: CachedSpace.Handle] RETURNS [validHandle: BOOLEAN, interval: VM.Interval] =
    BEGIN
    LoadDesc[handle];
    RETURN[pDescBuff.state~ = missing AND pDescBuff.interval.page = handle.page, pDescBuff.interval]
    END;

GetState: PUBLIC PROCEDURE [handle: CachedSpace.Handle] RETURNS [CachedSpace.State] =
    BEGIN
    LoadDesc[handle];
    RETURN[IF pDescBuff.interval.page~ = handle.page THEN missing ELSE pDescBuff.state]
    END;
```

```
Insert: PUBLIC PROCEDURE [handle: CachedSpace.Handle, count: VM.PageCount] =
    BEGIN
    desc: STree.Desc ← [hierarchy[CachedSpace.Desc[
        interval: [handle.page, count],
        level: handle.level,
        dPinned: ,
        dDirty: ,
        pinned: FALSE,
        state: unmapped,
        writeProtected: ,
        dataOrFile: ,
        pageRover: handle.page,
        vp: short[]]]];
    STree.Insert[@desc];
    END;

Touch: PUBLIC PROCEDURE [handle: CachedSpace.Handle] =
    BEGIN
    LoadDesc[handle];
    IF pDescBuff.state = missing THEN
        ERROR NotFound
    END;

Update: PUBLIC PROCEDURE [pDesc: POINTER TO CachedSpace.Desc] =
    BEGIN
    pDescBuff↑ ← pDesc↑;
    IF ~CachedSpace.Update[pDescBuff].found THEN
        STree.Update[@LOOPHOLE[pDescBuff↑, STree.Desc]]
    END;

NotFound: PUBLIC ERROR = CODE;

END.
```

LOG

Time: May 23, 1978 9:45 AM By: McJones Action: Created file
Time: June 27, 1978 6:35 PM By: McJones Action: Didn't set dPinned to FALSE before calling CachedSpace.Insert, .Update
Time: August 29, 1978 6:04 PM By: McJones Action: Clear dDirty before calling CachedSpace.Insert
Time: September 3, 1978 1:25 PM By: McJones Action: FindFirstWithin ignored keyNext.levelH
Time: September 10, 1978 1:38 PM By: McJones Action: Added pinned to constructor in Insert, etc.
Time: February 1, 1979 2:00 PM By: McJones CR20.169: Added pageRover to CachedSpace.Desc
Time: February 20, 1979 10:51 AM By: McJones CR20.177: Changed Update to preserve dPinned (!)

```
DIRECTORY
    CachedSpace: FROM "CachedSpace" USING [Desc],
    VM: FROM "VM" USING [Interval];

MapLog: DEFINITIONS =

BEGIN

WriteLog: PROCEDURE [interval: VM.Interval, pSpaceD: POINTER TO CachedSpace.Desc];
    -- Write one or more log entries; pDesc = NIL => interval is now unmapped

END.
```

LOG

Time: July 31, 1978 9:36 AM By: McJones Action: Created file
Time: July 31, 1979 1:10 PM By: McJones Action: Changed pWindow to pDesc

```
DIRECTORY
    -- CachedRegion: FROM "CachedRegion" USING [Apply, Insert, Operation],
    CachedSpace: FROM "CachedSpace" USING [Desc],
    Environment: FROM "Environment" USING [wordsPerPage],
    File: FROM "File" USING [Capability, Create, GetSize, MakePermanent, Unknown],
    FileTypes: FROM "FileTypes" USING [tMapLog],
    Inline: FROM "Inline" USING [LowHalf],
    KernelFile: FROM "KernelFile" USING [GetFilePoint, GetRootFile],
    MapLog: FROM "MapLog",
    PilotSwitches: FROM "PilotSwitches" USING [switches--.M--],
    RuntimeInternal: FROM "RuntimeInternal" USING [CleanMapLog],
    SimpleSpace: FROM "SimpleSpace" USING [Create, Handle, Map, Page],
    Utilities: FROM "Utilities" USING [LongPointerFromPage],
    VM: FROM "VM" USING [Interval, PageCount, PageNumber--, PageOffset--],
    VMMapLog: FROM "VMMapLog" USING [Descriptor, Entry, EntryBasePointer, EntryPointer],
    VMMPrograms: FROM "VMMPrograms",
    Volume: FROM "Volume" USING [ID, SystemID];

MapLogImpl: PROGRAM [pMapLogDesc: LONG POINTER]
    IMPORTS --CachedRegion,-- File, Inline, KernelFile, PilotSwitches, RuntimeInternal, SimpleSpace, Utilities, Volume
    EXPORTS MapLog, VMMPrograms =

BEGIN OPEN VMMapLog;

pageLog: VM.PageNumber;
countLog: VM.PageCount ← 20;

bLog: VMMapLog.EntryBasePointer;

mapLogging: BOOLEAN = PilotSwitches.switches.M = up;

WriteLog: PUBLIC PROCEDURE [interval: VM.Interval, pSpaceD: POINTER TO CachedSpace.Desc] =
    -- Write one or more log entries; pSpaceD = NIL => interval is now unmapped
    BEGIN
    IF mapLogging THEN
        BEGIN OPEN LOOPHOLE[pMapLogDesc, LONG POINTER TO Descriptor];
        writerNext: EntryPointer;
        pEntry: LONG POINTER TO Entry;
        count: VM.PageCount;
        fileOffset: LONG INTEGER ← 0;
        -- offsetWriter: VM.PageNumber ← LOOPHOLE[writer, CARDINAL]/Environment.wordsPerPage;
        -- offsetWriterNext, offsetWriterNextNext: VM.PageNumber;
        WHILE interval.count~ = 0 DO
            IF (writerNext ← writer + SIZE[Entry]) = limit THEN
                writerNext ← FIRST[EntryPointer];
            WHILE writerNext = reader DO RuntimeInternal.CleanMapLog[] ENDLOOP;
            pEntry ← @bLog[writer];
            count ← MIN[interval.count, 4096];
            IF pSpaceD = NIL THEN
                pEntry↑ ← [page: interval.page, count: count, writeProtected: , fill: , filePoint: nil[]]
            ELSE
                BEGIN
                KernelFile.GetFilePoint[pEntry, @pSpaceD.window.file, pSpaceD.window.base + fileOffset];
                pEntry.page ← interval.page;
                count ← pEntry.count ← MIN[pEntry.count, count];
                pEntry.writeProtected ← pSpaceD.writeProtected;
                fileOffset ← fileOffset + count
                END;
            writer ← writerNext;
            interval ← [interval.page + count, interval.count-count];
            -- If a log page boundary was crossed, adjust buffers
            --IF (offsetWriterNext ← LOOPHOLE[writerNext, CARDINAL]/Environment.wordsPerPage)~ = offsetWriter THEN
                --BEGIN
                --IF (offsetWriterNextNext ← offsetWriterNext + 1) = countLog THEN
                    --offsetWriterNextNext ← 0;
                --AdjustUniform[pageFlush: pageLog + offsetWriter, pageInsert: pageLog + offsetWriterNextNext];
                --IF CachedRegion.Apply[pageLog + offsetWriter, flushUnconditional].outcome~ = [ok[]] THEN ERROR;
                --IF CachedRegion.Insert[[
                        --interval: [pageLog + offsetWriterNextNext, 1],
                        --level: 1, ++ or does uniform swap unit have 1 higher level?
                        --dTemperature: , ++ don't care
                        --dPinned: TRUE,
                        --dDirty: FALSE,
                        --uniform: TRUE,
```

```
                        --state: outAlive, + + or outDead if page is not "between" reader and writer
                        --levelMapped: 1,
                        --beingRemapped: FALSE]].dDirty THEN ERROR;
                --offsetWriter ← offsetWriterNext
                --END
            ENDLOOP
        END
    END;

--flushUnconditional: CachedRegion.Operation =
        --[ifMissing: report, ifCheckedOut: wait, afterForking: wait, vp: flush[unconditional: TRUE]];

Initialize: PROCEDURE =
    -- The following could be generalized and used by STLeafImpl
    BEGIN OPEN LOOPHOLE[pMapLogDesc, LONG POINTER TO Descriptor];
    systemVolume: Volume.ID = Volume.SystemID[];
    file: File.Capability ← KernelFile.GetRootFile[FileTypes.tMapLog, systemVolume];
    handle: SimpleSpace.Handle;
    --offset: VM.PageOffset;
    BEGIN
    countLog ← Inline.LowHalf[File.GetSize[file ! File.Unknown => GO TO Create]];
    EXITS Create =>
        BEGIN
        file ← File.Create[volume: systemVolume, initialSize: countLog, type: FileTypes.tMapLog];
        File.MakePermanent[file]
        END
    END;
    IF countLog ~IN [2..4096] THEN ERROR;
    handle ← SimpleSpace.Create[count: countLog, location: hyperspace];
    SimpleSpace.Map[handle: handle, window: [file, 0], andPin: FALSE];
    pageLog ← SimpleSpace.Page[handle];
    --IF CachedRegion.Apply[pageLog, flushUnconditional].outcome~ = [ok[]] THEN ERROR;
    --FOR offset IN [0..2) DO
        --IF CachedRegion.Insert[[
                --interval: [pageLog + offset, 1],
                --level: 1, + + or does uniform swap unit have 1 higher level?
                --dTemperature: , + + don't care
                --dPinned: TRUE,
                --dDirty: FALSE,
                --uniform: TRUE,
                --state: outAlive, + + or outDead if page is not "between" reader and writer
                --levelMapped: 1,
                --beingRemapped: FALSE]].dDirty THEN ERROR
        --ENDLOOP;
    KernelFile.GetFilePoint[@self, @file, 0];
    IF self.count<countLog THEN ERROR; -- log file not contiguous
    self.page ← pageLog;
    self.count ← countLog;
    writer ← reader ← FIRST[EntryPointer];
    limit ← LOOPHOLE[countLog*Environment.wordsPerPage/SIZE[Entry]*SIZE[Entry], EntryPointer];
    bLog ← LOOPHOLE[Utilities.LongPointerFromPage[self.page]]
    END;

-- Initialization
IF mapLogging THEN Initialize[]

END.
```

LOG

Time: August 1, 1978 10:11 AM By: McJones Action: Created file
Time: August 7, 1978 4:51 PM By: McJones Action: pDesc.self.page wasn't initialized
Time: August 8, 1978 9:10 AM By: McJones Action: WriteLog didn't set entry page field in case of non-nil pWindow
Time: August 8, 1978 3:25 PM By: McJones Action: limit initialization didn't convert pages to words
Time: August 29, 1978 4:44 PM By: McJones Action: Added VMMode
Time: September 5, 1978 6:48 PM By: McJones Action: Replaced signal with CleanMapLog[], GetFilePoint moved to SpecialFile
Time: September 15, 1978 4:47 PM By: McJones Action: Getting ready for "uniform" swap unit management
Time: September 29, 1978 11:05 AM By: McJones CR20.42: Replaced PutRootFile with MakePermanent
Time: July 31, 1979 1:12 PM By: McJones Action: Prepared to add writeProtected to map log entry
Time: August 16, 1979 8:56 PM By: McJones Action: Added writeProtected to map log entry; SpecialFile => KernelFile; VMMode

=> PilotSwitches

```
DIRECTORY
    CachedRegion: FROM "CachedRegion" USING [Desc],
    VM: FROM "VM" USING [PageNumber];

Projection: DEFINITIONS =

BEGIN

ForceOut: PROCEDURE [pageMember: VM.PageNumber];

Get: PROCEDURE [pageMember: VM.PageNumber] RETURNS [CachedRegion.Desc];

Merge: PROCEDURE [pageNext: VM.PageNumber];

Split: PROCEDURE [pageNext: VM.PageNumber];

Touch: PROCEDURE [pageMember: VM.PageNumber];

TranslateLevel: PROCEDURE [pageMember: VM.PageNumber, delta: INTEGER];

END.
```

LOG

Time: April 1978 *By: PMcJ* *Action: Created file*
Time: June 24, 1978 5:26 PM *By: PMcJ* *Action: PageNumber moved to VM*

```
DIRECTORY
    CachedRegion: FROM "CachedRegion" USING [Apply, Desc, Insert, State, Outcome],
    CachedSpace: FROM "CachedSpace" USING [Level],
    Projection: FROM "Projection",
    STree: FROM "STree" USING [Delete, Desc, Get, Insert, Key, PDesc, Update],
    VM: FROM "VM" USING [PageNumber, PageCount],
    VMMPrograms: FROM "VMMPrograms";

ProjectionImpl: PROGRAM [countVM: VM.PageCount]
    IMPORTS CachedRegion, STree
    EXPORTS Projection, VMMPrograms =

BEGIN

ForceOut: PUBLIC PROCEDURE [pageMember: VM.PageNumber] =
    -- If descriptor for projection region containing pageMember is cached and dirty, clean it
    BEGIN
    desc: CachedRegion.Desc;
    outcome: CachedRegion.Outcome = CachedRegion.Apply[pageMember,
            [ifMissing: report, ifCheckedOut: wait, afterForking: , vp: get[andResetDDirty: TRUE, pDescResult:
            @desc]]].outcome;
    WITH outcome SELECT FROM
        ok =>
            IF desc.dDirty THEN
                BEGIN OPEN desc;
                projDesc: projection STree.Desc ← [projection[
                    interval.page,
                    level,
                    IF state ~IN CachedRegion.State[unmapped..outAlive] THEN outAlive ELSE state,
                    levelMapped,
                    beingRemapped]];
                STree.Update[LOOPHOLE[LONG[@projDesc], STree.PDesc]]
                END;
        regionDMissing =>
            NULL;
        ENDCASE =>
            ERROR
    END;

Get: PUBLIC PROCEDURE [pageMember: VM.PageNumber] RETURNS [desc: CachedRegion.Desc] =
    -- Return descriptor for projection region containing pageMember
    BEGIN
    projDesc: projection STree.Desc;
    keyNext: STree.Key;
    descVictim: CachedRegion.Desc;
    outcome: CachedRegion.Outcome = CachedRegion.Apply[pageMember,
            [ifMissing: report, ifCheckedOut: wait, afterForking: , vp: get[andResetDDirty: FALSE, pDescResult:
            @desc]]].outcome;
    WITH outcome SELECT FROM
        ok =>
            -- If descriptor is in cache, just return it
            NULL;
        regionDMissing =>
            -- Otherwise, fetch descriptor from hierarchy and cache it
            BEGIN
            keyNext ← STree.Get[LOOPHOLE[LONG[@projDesc], STree.PDesc], [projection[pageMember]]];
            BEGIN OPEN projDesc;
            desc ← [
                interval: [page, keyNext.pageP-page],
                level: level,
                dPinned: FALSE,
                dTemperature: ,
                dDirty: FALSE,
                state: state,
                levelMapped: levelMapped,
                beingRemapped: beingRemapped]
            END;
            descVictim ← CachedRegion.Insert[desc];
            IF descVictim.dDirty THEN
                BEGIN OPEN descVictim;
                projDesc ← [projection[interval.page, level, state, levelMapped, beingRemapped]];
                STree.Update[LOOPHOLE[LONG[@projDesc], STree.PDesc]]
                END
```

```
            END;
        ENDCASE =>
            ERROR
    END;

Merge: PUBLIC PROCEDURE [pageNext: VM.PageNumber] =
    -- Removes the projection region boundary at pageNext
    -- NOTE: modifies projection file only, does not update region cache
    BEGIN
    projDesc: projection STree.Desc;
--assert--IF pageNext = 0 THEN ERROR;
    -- Unconditionally "resurrect" a dead region: conservative approach to avoid propagating deadness
    [] ← STree.Get[LOOPHOLE[LONG[@projDesc], STree.PDesc], [projection[pageNext-1]]];
    IF projDesc.state = outDead THEN
        BEGIN
        projDesc.state ← outAlive;
        STree.Update[LOOPHOLE[LONG[@projDesc], STree.PDesc]]
        END;
    -- Do the merge (depends on the fact that the size of a projection STree.Desc is implicit)
    STree.Delete[[projection[pageNext]]];
    END;

Split: PUBLIC PROCEDURE [pageNext: VM.PageNumber] =
    -- If there is not already a projection region boundary at pageNext, make one
    -- NOTE: modifies projection file only, does not update region cache
    BEGIN
    projDesc: projection STree.Desc;
    IF pageNext~ = countVM THEN
        BEGIN
        -- Depends on the fact that the size of a projection STree.Desc is implicit
        [] ← STree.Get[LOOPHOLE[LONG[@projDesc], STree.PDesc], [projection[pageNext]]];
        IF projDesc.page~ = pageNext THEN
            BEGIN
            projDesc.page ← pageNext;
            STree.Insert[LOOPHOLE[LONG[@projDesc], STree.PDesc]]
            END
        END
    END;

Touch: PUBLIC PROCEDURE [pageMember: VM.PageNumber] =
    -- Ensure descriptor for projection region containing pageMember is cached
    BEGIN
    [] ← Get[pageMember]
    END;

TranslateLevel: PUBLIC PROCEDURE [pageMember: VM.PageNumber, delta: INTEGER] =
    -- Add (signed) delta to level field of descriptor for region containing pageMember
    -- NOTE: modifies projection file only, does not update region cache
    BEGIN
    projDesc: projection STree.Desc;
    [] ← STree.Get[LOOPHOLE[LONG[@projDesc], STree.PDesc], [projection[pageMember]]];
--assert--IF projDesc.level + delta ~IN CachedSpace.Level THEN ERROR;
    projDesc.level ← projDesc.level + delta;
    STree.Update[LOOPHOLE[LONG[@projDesc], STree.PDesc]]
    END;

END.
```

LOG

Time: May 24, 1978 10:13 AM By: McJones Action: Created file
Time: June 23, 1978 10:57 AM By: McJones Action: Split didn't check for pageNext = countVM
Time: August 2, 1978 9:27 AM By: McJones Action: Updated to new CachedRegion interface
Time: August 4, 1978 2:19 PM By: McJones Action: Added beingRemapped
Time: August 29, 1978 7:32 PM By: McJones Action: Cleared dDirty before calling CachedRegion.Insert

```
DIRECTORY
    CachedRegion: FROM "CachedRegion" USING [activate, Apply, deactivate, Desc, flush, forceOut, kill, Mapped,
        Operation, Outcome, pin, Swappable, unmap, unpin, writeProtect],
    CachedSpace: FROM "CachedSpace" USING [Desc, Handle, handleNull, handleVM, Level],
    ControlDefs: FROM "ControlDefs" USING [Frame, GlobalFrameHandle, StateVector, TrapLink],
    Environment: FROM "Environment" USING [Long],
    File: FROM "File" USING [Create, Delete, Error, firstPageNumber, GetAttributes, GetSize, lastPageNumber, PageCount,
        read, Unknown, write],
    FileTypes: FROM "FileTypes" USING [tAnonymousFile],
    Forgot: FROM "Forgot" USING [Unimplemented],
    FrameOps: FROM "FrameOps" USING [GetReturnLink, MyLocalFrame],
    Hierarchy: FROM "Hierarchy" USING [Delete, FindFirstWithin, GetDescriptor, GetInterval, GetState, Insert, Touch,
        Update],
    Inline: FROM "Inline" USING [BITAND, LowHalf],
    MapLog: FROM "MapLog" USING [WriteLog],
    Process: FROM "Process" USING [Detach],
    ProcessInternal: FROM "ProcessInternal" USING [DisableInterrupts, EnableInterrupts], -- until PrincOp trap parameters
    Projection: FROM "Projection" USING [ForceOut, Get, Merge, Split, Touch, TranslateLevel],
    Runtime: FROM "Runtime" USING [CallDebugger],
    SDDefs: FROM "SDDefs" USING [SD, sWriteProtect],
    Space: FROM "Space",
    SpecialSpace: FROM "SpecialSpace",
    SwapperException: FROM "SwapperException" USING [Await],
    TrapOps: FROM "TrapOps" USING [ReadOTP],
    VM: FROM "VM" USING [Interval, PageCount],
    VMMPrograms: FROM "VMMPrograms",
    Volume: FROM "Volume" USING [ID, Unknown];

SpaceImpl: MONITOR [countVM: VM.PageCount, volumeData: Volume.ID, handleMDS: CachedSpace.Handle]
    IMPORTS CachedRegion, File, Forgot, FrameOps, Hierarchy, Inline, MapLog, Process, ProcessInternal, Projection,
        Runtime, SwapperException, TrapOps, Volume
    EXPORTS SpecialSpace, Space, VMMPrograms
    SHARES File -- to extract permissions -- =

BEGIN OPEN Space;

Interval: TYPE = VM.Interval;

Level: TYPE = CachedSpace.Level;

SpaceD: TYPE = CachedSpace.Desc;

RegionD: TYPE = CachedRegion.Desc;


-- Space implementation:

-- Public Constants

Error: PUBLIC ERROR [type: ErrorType] = CODE;
InsufficientSpace: PUBLIC ERROR [available: PageCount] = CODE;
mds: PUBLIC Handle ← LOOPHOLE[handleMDS];
nullHandle: PUBLIC Handle ← LOOPHOLE[CachedSpace.handleNull];
virtualMemory: PUBLIC Handle ← LOOPHOLE[CachedSpace.handleVM];

-- Private Constants

AddressFault: ERROR [page: PageNumber] = CODE; -- not to be caught by client
InsufficientPermissions: ERROR [page: PageNumber] = CODE; -- not to be caught by client
intervalMDS: Interval = Hierarchy.GetInterval[handleMDS].interval;

maxPagesInCodeBlock: PageCount = 256;
codeBdyMask: WORD = LOOPHOLE[-(maxPagesInCodeBlock-1)-1, WORD]; -- = BITNOT[maxPagesInCodeBlock-1]
```

```
-- Monitor externals:

Create: PUBLIC PROCEDURE [size: PageCount, parent: Handle, base: PageOffset] RETURNS [Handle] =
    BEGIN RETURN[CreateInternal[size: size, parent: parent, base: base, forCode: FALSE]] END;

CreateForCode: PUBLIC PROCEDURE [size: PageCount, parent: Handle, base: PageOffset] RETURNS [Handle] =
    -- (The microcode requires that code blocks not cross 64kw boundaries.)
    BEGIN RETURN[CreateInternal[size: size, parent: parent, base: base, forCode: TRUE]] END;

LongPointerFromPage: PUBLIC PROCEDURE [page: PageNumber] RETURNS [LONG POINTER] =
    BEGIN
    RETURN[LOOPHOLE[Environment.Long[any[highbits: page/256, lowbits: (page MOD 256)*256]]]]
    END;

PageFromLongPointer: PUBLIC PROCEDURE [lp: LONG POINTER] RETURNS [page: PageNumber] =
    BEGIN
    RETURN[LOOPHOLE[lp, Environment.Long].highbits*256 + LOOPHOLE[lp, Environment.Long].lowbits/256]
    END;


-- Monitor entries:

Activate: PUBLIC ENTRY PROCEDURE [space: Handle] =
    BEGIN OPEN handle: LOOPHOLE[space, CachedSpace.Handle];
    IF ~ApplyToSpace[handle, CachedRegion.activate].validHandle THEN
        RETURN WITH ERROR Error[invalidHandle]
    END;

CreateInternal: ENTRY PROCEDURE [size: PageCount, parent: Handle, base: PageOffset, forCode: BOOLEAN] RETURNS
        [Handle] =
    BEGIN OPEN handleParent: LOOPHOLE[parent, CachedSpace.Handle];
    page: PageNumber;
    handleNew: CachedSpace.Handle;
    spaceDParent: SpaceD;
    IF ~Hierarchy.GetDescriptor[@spaceDParent, handleParent].validHandle THEN
        RETURN WITH ERROR Error[invalidHandle];
    IF size=0 OR (base~=defaultBase AND base>=spaceDParent.interval.count) THEN
        RETURN WITH ERROR Error[invalidParameters];
    IF handleParent.level=LAST[Level] THEN
        RETURN WITH ERROR Error[spaceTreeTooDeep];
    IF base~=defaultBase THEN -- explicit base given --
        BEGIN
        regionD: RegionD = Projection.Get[page ← handleParent.page+base]; -- so page IN regionD.interval
        IF size>(regionD.interval.page+regionD.interval.count)-page OR regionD.level~=handleParent.level
          OR (forCode AND page/maxPagesInCodeBlock # (page+size-1)/maxPagesInCodeBlock) THEN
            RETURN WITH ERROR Error[invalidParameters];
        END
    ELSE -- create anywhere --
        BEGIN OPEN spaceDParent; -- USING [interval, pageRover]
        countMax: PageCount ← 0;
        HoleNotFound: INTERNAL PROCEDURE [regionD: RegionD] RETURNS [BOOLEAN] =
            BEGIN codeInterval: VM.Interval;
            IF regionD.level=handleParent.level THEN
                BEGIN
                IF forCode THEN
                    BEGIN
                    [codeInterval] ← MaxIntervalForCode[regionD];
                    IF codeInterval.count>countMax THEN BEGIN page ← codeInterval.page; countMax ← codeInterval.count
                        END
                    END
                ELSE --for data-- IF regionD.interval.count>countMax THEN
                    BEGIN page ← regionD.interval.page; countMax ← regionD.interval.count END;
                IF countMax>=size THEN RETURN[FALSE]; --the current region is big enough--
                END;
            RETURN[TRUE]
            END;
        IF ForAllRegions[[pageRover, interval.count-(pageRover-interval.page)], HoleNotFound] THEN IF
                ForAllRegions[[interval.page, pageRover-interval.page], HoleNotFound] THEN
            RETURN WITH ERROR InsufficientSpace[available: countMax];
        pageRover ← page+size; -- pageRover = interval.page + interval.count is ok
        Hierarchy.Update[@spaceDParent]
        END;
    handleNew ← CachedSpace.Handle[1+handleParent.level, page];
    Hierarchy.Insert[handleNew, size];
```

```
    ApplyToInterval[Interval[page, 1], CachedRegion.flush];
    Projection.Split[page];
    Projection.Split[page + size];
    Projection.TranslateLevel[pageMember: page, delta: 1];
    RETURN[LOOPHOLE[handleNew]]
    END;

CreateUniformSwapUnits: PUBLIC --ENTRY-- PROCEDURE [size: PageCount, parent: Handle] =
    BEGIN
    SIGNAL Forgot.Unimplemented
    END;

Deactivate: PUBLIC ENTRY PROCEDURE [space: Handle] =
    BEGIN OPEN handle: LOOPHOLE[space, CachedSpace.Handle];
    IF ~ApplyToSpace[handle, CachedRegion.deactivate].validHandle THEN
        RETURN WITH ERROR Error[invalidHandle]
    END;

Delete: PUBLIC ENTRY PROCEDURE [space: Handle] =
    BEGIN OPEN handle: LOOPHOLE[space, CachedSpace.Handle];
    level: Level;
    DeleteIfLeaf: INTERNAL PROCEDURE [regionD: RegionD] RETURNS [BOOLEAN] =
        -- Assume level>0 (i.e. space~ = virtualMemory)
        BEGIN
        IF regionD.level = level THEN
            -- Region described by regionD corresponds to a leaf space
            BEGIN
            intervalParent: Interval = Hierarchy.GetInterval[CachedSpace.Handle[level-1, regionD.interval.page]].interval;
            pageNext: PageNumber = regionD.interval.page + regionD.interval.count;
            mergeLeft: BOOLEAN = regionD.interval.page~ = intervalParent.page
                AND Projection.Get[regionD.interval.page-1].level<regionD.level;
            mergeRight: BOOLEAN = pageNext~ = intervalParent.page + intervalParent.count
                AND regionD.level>Projection.Get[pageNext].level;
            deltaPageFlush: CARDINAL = IF mergeLeft THEN 1 ELSE 0;
            deltaCountFlush: CARDINAL = deltaPageFlush + (IF mergeRight THEN 1 ELSE 0);
            IF regionD.state IN CachedRegion.Mapped AND regionD.levelMapped = regionD.level THEN
                BEGIN
                spaceD: SpaceD;
                [] ← Hierarchy.GetDescriptor[@spaceD, CachedSpace.Handle[regionD.level, regionD.interval.page]];
                UnmapInternal[@spaceD]
                END;
            Hierarchy.Delete[CachedSpace.Handle[regionD.level, regionD.interval.page]];
            ApplyToInterval[Interval[regionD.interval.page-deltaPageFlush, regionD.interval.count + deltaCountFlush],
                CachedRegion.flush];
            Projection.TranslateLevel[pageMember: regionD.interval.page, delta: -1];
            IF mergeLeft THEN
                Projection.Merge[regionD.interval.page];
            IF mergeRight THEN
                Projection.Merge[pageNext];
            END;
        RETURN[TRUE]
        END;
    levelLeaves: Level ← handle.level;
    MaxLevel: PROCEDURE [regionD: RegionD] RETURNS [BOOLEAN] =
        BEGIN
        levelLeaves ← MAX[levelLeaves, regionD.level];
        RETURN[TRUE]
        END;
    valid: BOOLEAN; interval: Interval; [valid, interval] ← Hierarchy.GetInterval[handle];
    IF ~valid THEN
        RETURN WITH ERROR Error[invalidHandle];
    IF handle.level< = handleMDS.level AND handleMDS.page IN [interval.page..interval.page + interval.count) THEN
        RETURN WITH ERROR Error[invalidParameters];
    [] ← ForAllRegions[interval, MaxLevel]; -- levelLeaves ← max region level, region in interval
    FOR level DECREASING IN [handle.level..levelLeaves] DO
        [] ← ForAllRegions[interval, DeleteIfLeaf]
        ENDLOOP
    END;
```

```
DeleteSwapUnits: PUBLIC --ENTRY-- PROCEDURE [space: Handle] =
    BEGIN
    SIGNAL Forgot.Unimplemented
    END;

ForceOut: PUBLIC ENTRY PROCEDURE [space: Handle] =
    BEGIN OPEN handle: LOOPHOLE[space, CachedSpace.Handle];
    IF ~ApplyToSpace[handle, CachedRegion.forceOut].validHandle THEN
        RETURN WITH ERROR Error[invalidHandle]
    END;

GetAttributes: PUBLIC ENTRY PROCEDURE [space: Handle]
        RETURNS [parent, lowestChild, nextSibling: Handle, base: PageOffset, size: PageCount, mapped: BOOLEAN] =
    BEGIN OPEN handle: LOOPHOLE[space, CachedSpace.Handle];
    valid: BOOLEAN; interval: Interval; [valid, interval] ← Hierarchy.GetInterval[handle];
    IF ~valid THEN
        RETURN WITH ERROR Error[invalidHandle];
    IF handle.level = 0 THEN
        BEGIN
        parent ← nextSibling ← nullHandle;
        base ← 0
        END
    ELSE
        BEGIN
        intervalParent: Interval = Hierarchy.GetInterval[CachedSpace.Handle[handle.level-1, interval.page]].interval;
        pageNext: PageNumber = interval.page + interval.count;
        countNext: PageCount = intervalParent.page + intervalParent.count-pageNext;
        parent ← LOOPHOLE[CachedSpace.Handle[handle.level-1, intervalParent.page]];
        nextSibling ← LOOPHOLE[Hierarchy.FindFirstWithin[CachedSpace.Handle[handle.level, pageNext], countNext]];
        base ← interval.page-intervalParent.page
        END;
    lowestChild ← IF handle.level<LAST[Level] THEN
            LOOPHOLE[Hierarchy.FindFirstWithin[CachedSpace.Handle[handle.level + 1, handle.page], interval.count]]
        ELSE
            nullHandle;
    size ← interval.count;
    mapped ← Hierarchy.GetState[handle] = mapped
    END;

GetHandle: PUBLIC ENTRY PROCEDURE [page: PageNumber] RETURNS [Handle] =
    BEGIN
    level: Level;
    IF page>= countVM THEN
        RETURN WITH ERROR Error[invalidParameters];
    level ← Projection.Get[page].level;
    RETURN[LOOPHOLE[CachedSpace.Handle[level, Hierarchy.GetInterval[CachedSpace.Handle[level, page]].interval.page]]]
    END;

GetWindow: PUBLIC ENTRY PROCEDURE [space: Handle] RETURNS [WindowOrigin] =
    BEGIN OPEN handle: LOOPHOLE[space, CachedSpace.Handle];
    spaceD: SpaceD;
    IF ~Hierarchy.GetDescriptor[@spaceD, handle].validHandle THEN
        RETURN WITH ERROR Error[invalidHandle];
    IF spaceD.state~ = mapped THEN
        RETURN WITH ERROR Error[noWindow];
    IF spaceD.state~ = mapped THEN ERROR;
    RETURN[IF spaceD.dataOrFile = file THEN spaceD.window ELSE defaultWindow]
    END;

Kill: PUBLIC ENTRY PROCEDURE [space: Handle] =
    BEGIN OPEN handle: LOOPHOLE[space, CachedSpace.Handle];
    IF ~ApplyToSpace[handle, CachedRegion.kill].validHandle THEN
        RETURN WITH ERROR Error[invalidHandle]
    END;
```

```
LongPointer: PUBLIC ENTRY PROCEDURE [space: Handle] RETURNS [LONG POINTER] =
    BEGIN OPEN handle: LOOPHOLE[space, CachedSpace.Handle];
    IF Hierarchy.GetState[handle] = missing THEN
        RETURN WITH ERROR Error[invalidHandle];
    RETURN[LongPointerFromPage[handle.page]]
    END;

MakeReadOnly: PUBLIC ENTRY PROCEDURE [space: Handle] =
    BEGIN OPEN handle: LOOPHOLE[space, CachedSpace.Handle];
    spaceD: SpaceD;
    IF ~Hierarchy.GetDescriptor[@spaceD, handle].validHandle THEN
        RETURN WITH ERROR Error[invalidHandle];
    IF spaceD.state~ = mapped THEN
        RETURN WITH ERROR Error[noWindow];
    IF ~ForAllRegions[spaceD.interval, NotPinned] THEN
        RETURN WITH ERROR Error[invalidMappingOperation];
    spaceD.writeProtected ← TRUE;
    Hierarchy.Update[@spaceD];
    ApplyToInterval[spaceD.interval, CachedRegion.writeProtect]
    END;

Map: PUBLIC ENTRY PROCEDURE [space: Handle, window: WindowOrigin] =
    BEGIN OPEN handle: LOOPHOLE[space, CachedSpace.Handle];
    Unmapped: PROCEDURE [regionD: RegionD] RETURNS [BOOLEAN] =
        BEGIN
        RETURN[regionD.state = unmapped]
        END;
    volumeUnknown: Volume.ID;
    spaceD: SpaceD;
    IF ~Hierarchy.GetDescriptor[@spaceD, handle].validHandle THEN
        RETURN WITH ERROR Error[invalidHandle];
    IF ~ForAllRegions[spaceD.interval, Unmapped] THEN
        RETURN WITH ERROR Error[invalidMappingOperation];
    BEGIN ENABLE
        BEGIN
        File.Unknown => GO TO UnknownFile;
        Volume.Unknown --[volume]-- => BEGIN volumeUnknown ← volume; GO TO UnknownVolume END
        END;
    spaceD.dataOrFile ← data;
    IF window.file~ = defaultWindow.file THEN
        BEGIN
        IF window.base ~IN [File.firstPageNumber..File.lastPageNumber] THEN -- LONG subrange types not yet implemented
            RETURN WITH ERROR Error[invalidWindow];
        IF Inline.BITAND[window.file.permissions, File.read] = 0 THEN
            RETURN WITH ERROR File.Error[insufficientPermissions];
        spaceD.dataOrFile ← file;
        spaceD.window ← window
        END;
    -- Allocate backing file if data, and update space descriptor
    spaceD.state ← mapped;
    MapInternal[@spaceD]
    EXITS
        UnknownFile => RETURN WITH ERROR File.Unknown[window.file];
        UnknownVolume => RETURN WITH ERROR Volume.Unknown[volumeUnknown];
    END;
    ApplyToInterval[spaceD.interval, [ifMissing: report, ifCheckedOut: wait, afterForking: , vp: map[handle.level]]];
    IF spaceD.dataOrFile = data THEN
        ApplyToInterval[spaceD interval, CachedRegion.kill];
    MapLog.WriteLog[Interval[spaceD.interval.page, spaceD.countMapped], @spaceD]
    END;

Pointer: PUBLIC ENTRY PROCEDURE [space: Handle] RETURNS [POINTER] =
    BEGIN OPEN handle: LOOPHOLE[space, CachedSpace.Handle];
    IF Hierarchy.GetState[handle] = missing THEN
        RETURN WITH ERROR Error[invalidHandle];
    IF handle.levelK = handleMDS.level OR handle.page ~IN [intervalMDS.page..intervalMDS.page + intervalMDS.count) THEN
        RETURN WITH ERROR Error[invalidParameters];
    RETURN[LOOPHOLE[(handle.page-intervalMDS.page)*wordsPerPage]]
    END;
```

```
Remap: PUBLIC ENTRY PROCEDURE [space: Handle, window: WindowOrigin] =
    BEGIN OPEN handle: LOOPHOLE[space, CachedSpace.Handle];
    volumeUnknown: Volume.ID;
    spaceDOld: SpaceD;
    spaceD: SpaceD;
    IF ~Hierarchy.GetDescriptor[@spaceD, handle].validHandle THEN
        RETURN WITH ERROR Error[invalidHandle];
    IF spaceD.state~ = mapped THEN
        RETURN WITH ERROR Error[noWindow];
    IF spaceD.state~ = mapped THEN ERROR;
    spaceDOld ← spaceD;
    BEGIN ENABLE
        BEGIN
        File.Unknown => GO TO UnknownFile;
        Volume.Unknown --[volume]-- => BEGIN volumeUnknown ← volume; GO TO UnknownVolume END
        END;
    spaceD.dataOrFile ← data;
    IF window.file~ = defaultWindow.file THEN
        BEGIN
        IF window.base ~IN [File.firstPageNumber..File.lastPageNumber] THEN -- LONG subrange types not yet implemented
            RETURN WITH ERROR Error[invalidWindow];
        IF File.GetAttributes[window.file].immutable THEN
            RETURN WITH ERROR File.Error[immutable];
        IF Inline.BITAND[File.read, window.file.permissions] = 0 OR Inline.BITAND[File.write, window.file.permissions] = 0 THEN
            RETURN WITH ERROR File.Error[insufficientPermissions];
        spaceD.dataOrFile ← file;
        spaceD.window ← window
        END;
    IF ~ForAllRegions[spaceD.interval, NotPinned] THEN
        RETURN WITH ERROR Error[invalidMappingOperation];
    -- Mark all regions as "beingRemapped", and force out dirty ones unless space was "data"
    -- Assume writeProtected implies ~dirty , i.e. no magic stores by debugger, etc.
    ApplyToInterval[spaceD.interval, [ifMissing: report, ifCheckedOut: wait, afterForking: return,
            vp: remapA[firstClean: spaceDOld.dataOrFile = file AND ~spaceDOld.writeProtected]]];
    -- Allocate backing file if data, and update space descriptor
    spaceD.state ← beingRemapped;
    MapInternal[@spaceD]
    EXITS
        UnknownFile => RETURN WITH ERROR File.Unknown[window.file];
        UnknownVolume => RETURN WITH ERROR Volume.Unknown[volumeUnknown];
    END;
    -- Ensure each region has been dirtied since it and space became "beingRemapped"
    ApplyToInterval[spaceD.interval, [ifMissing: report, ifCheckedOut: wait, afterForking: return, vp:
            remapB[@spaceDOld]]];
    -- Mark space as no longer "beingRemapped"
    spaceD.state ← mapped;
    Hierarchy.Update[@spaceD];
    -- (For remote swapping, must unpin old file (and containing volume) from FilePageTransferrer cache)
    IF spaceDOld.dataOrFile = data THEN
        File.Delete[spaceDOld.window.file];
    MapLog.WriteLog[Interval[spaceD.interval.page, spaceD.countMapped], @spaceD];
    IF spaceD.countMapped<spaceD.interval.count THEN
        MapLog.WriteLog[Interval[spaceD.countMapped, spaceD.interval.count-spaceD.countMapped], NIL]
    END;

Unmap: PUBLIC ENTRY PROCEDURE [space: Handle] =
    BEGIN OPEN handle: LOOPHOLE[space, CachedSpace.Handle];
    spaceD: SpaceD;
    IF ~Hierarchy.GetDescriptor[@spaceD, handle].validHandle THEN
        RETURN WITH ERROR Error[invalidHandle];
    IF spaceD.state~ = mapped THEN
        RETURN WITH ERROR Error[noWindow];
    UnmapInternal[@spaceD]
    END;
```

```
VMPageNumber: PUBLIC ENTRY PROCEDURE [space: Handle] RETURNS [PageNumber] =
    BEGIN OPEN handle: LOOPHOLE[space, CachedSpace.Handle];
    IF Hierarchy.GetState[handle] = missing THEN
        RETURN WITH ERROR Error[invalidHandle];
    RETURN[handle.page]
    END;

-- Monitor internals

NotePinned: SIGNAL [levelMax: Level, page: PageNumber] = CODE;

ApplyToInterval: INTERNAL PROCEDURE [interval: Interval, operation: CachedRegion.Operation] =
    -- May raise NotePinned (only if operation.action = unmap)
    BEGIN
    page, pageNext: PageNumber;
    outcome: CachedRegion.Outcome;
    page ← interval.page;
    DO
        BEGIN
        [outcome, pageNext] ← CachedRegion.Apply[page, operation];
        WITH outcome SELECT FROM
            ok => GO TO NextRegion;
            notePinned -- [levelMax] -- => BEGIN SIGNAL NotePinned[levelMax, page]; GO TO NextRegion END;
            regionDMissing => Projection.Touch[page];
            regionDDirty => Projection.ForceOut[page];
            spaceDMissing -- [level] -- => Hierarchy.Touch[CachedSpace.Handle[level, page]];
            -- error -- ENDCASE => ERROR
        EXITS NextRegion => IF interval.page + interval.count <= pageNext THEN EXIT ELSE page ← pageNext
        END
        ENDLOOP
    END;

ApplyToSpace: INTERNAL PROCEDURE [handle: CachedSpace.Handle, operation: CachedRegion.Operation]
        RETURNS [validHandle: BOOLEAN] =
    BEGIN
    interval: Interval;
    [validHandle, interval] ← Hierarchy.GetInterval[handle];
    IF validHandle THEN ApplyToInterval[interval, operation]
    END;

ForAllRegions: INTERNAL PROCEDURE [interval: Interval,
        Predicate: PROCEDURE [RegionD] RETURNS [true: BOOLEAN]] RETURNS [BOOLEAN] =
    BEGIN
    page: PageNumber;
    regionD: RegionD;
    FOR page ← interval.page, regionD.interval.page + regionD.interval.count WHILE page<interval.page + interval.count DO
        regionD ← Projection.Get[page];
        IF ~Predicate[regionD] THEN RETURN[FALSE]
        ENDLOOP;
    RETURN[TRUE]
    END;

MapInternal: INTERNAL PROCEDURE [pSpaceD: POINTER TO SpaceD] =
    BEGIN OPEN pSpaceD;
    IF dataOrFile = data THEN
        BEGIN
        window ← [File.Create[volume: volumeData, initialSize: interval.count, type: FileTypes.tAnonymousFile], 0];
        writeProtected ← FALSE;
        countMapped ← interval.count
        END
    ELSE
        BEGIN
        countFile: File.PageCount = MAX[File.GetSize[window.file], window.base]-window.base;
        writeProtected ← File.GetAttributes[window.file].immutable OR Inline.BITAND[window.file.permissions, File.write] = 0;
        countMapped ← IF interval.count <= countFile THEN interval.count ELSE Inline.LowHalf[countFile]
        END;
    -- (For remote swapping, must pin file (and containing volume) in FilePageTransferrer caches)
    Hierarchy.Update[pSpaceD]
    END;
```

```
MaxIntervalForCode: INTERNAL PROCEDURE [regionD: RegionD] RETURNS [Interval] =
    BEGIN OPEN regionD.interval; -- page, count
    endRegion: PageNumber;
    firstBdy: PageNumber = Inline.BITAND[page + maxPagesInCodeBlock, codeBdyMask]; --overflow is ok, yields 0--
    IF (endRegion ← page + count) <= firstBdy OR firstBdy = 0 THEN RETURN[[page, count]]; -- region ends before first boundary
    IF Inline.BITAND[endRegion, codeBdyMask] # firstBdy THEN RETURN[[firstBdy, maxPagesInCodeBlock]]; -- region contains
            a whole CodeBlock
    IF firstBdy-page >= endRegion-firstBdy THEN RETURN[[page,firstBdy-page]]
        ELSE RETURN[[firstBdy,endRegion-firstBdy]] ; -- region contains exactly one code boundary
    END;

NotPinned: INTERNAL PROCEDURE [regionD: RegionD] RETURNS [BOOLEAN] =
    BEGIN RETURN[regionD.state~ = inPinned] END;

UnmapInternal: INTERNAL PROCEDURE [pSpaceD: POINTER TO SpaceD] =
    BEGIN OPEN pSpaceD;
--assert--IF state~ = mapped THEN ERROR;
    IF dataOrFile = data THEN
        ApplyToInterval[interval, CachedRegion.kill];
    ApplyToInterval[interval, CachedRegion.unmap !
        NotePinned -- [levelMax, page] -- =>
            BEGIN
            levelSub: Level; spaceDSub: SpaceD;
            FOR levelSub IN [level + 1..levelMax] DO
                [] ← Hierarchy.GetDescriptor[@spaceDSub, CachedSpace.Handle[levelSub, page]];
                IF spaceDSub.pinned THEN
                    BEGIN spaceDSub.pinned ← FALSE; Hierarchy.Update[@spaceDSub]; EXIT END
                ENDLOOP;
            RESUME
            END];
    pinned ← FALSE;
    state ← unmapped;
    Hierarchy.Update[pSpaceD];
    -- (For remote swapping, must unpin old file (and containing volume) from FilePageTransferrer cache)
    IF dataOrFile = data THEN
        File.Delete[window.file];
    MapLog.WriteLog[Interval[interval.page, countMapped], NIL]
    END;

-- SpecialSpace implementation

-- Monitor entries

MakeResident: PUBLIC ENTRY PROCEDURE [space: Handle] =
    BEGIN OPEN handle: LOOPHOLE[space, CachedSpace.Handle];
    Pinnable: PROCEDURE [regionD: RegionD] RETURNS [BOOLEAN] =
        BEGIN
        RETURN[regionD.state IN CachedRegion.Swappable AND regionD.levelMapped<= handle.level]
        END;
    spaceD: SpaceD;
    IF ~Hierarchy.GetDescriptor[@spaceD, handle].validHandle THEN
        RETURN WITH ERROR Error[invalidHandle];
    IF ~ForAllRegions[spaceD.interval, Pinnable] THEN
        RETURN WITH ERROR Error[invalidMappingOperation];
    spaceD.pinned ← TRUE;
    Hierarchy.Update[@spaceD];
    ApplyToInterval[spaceD.interval, CachedRegion.pin]
    END;

MakeSwappable: PUBLIC ENTRY PROCEDURE [space: Handle] =
    BEGIN OPEN handle: LOOPHOLE[space, CachedSpace.Handle];
    spaceD: SpaceD;
    IF ~Hierarchy.GetDescriptor[@spaceD, handle].validHandle THEN
        RETURN WITH ERROR Error[invalidHandle];
    IF ~spaceD.pinned THEN
        RETURN WITH ERROR Error[invalidMappingOperation];
    spaceD.pinned ← FALSE;
    Hierarchy.Update[@spaceD];
    ApplyToInterval[spaceD.interval, CachedRegion.unpin]
    END;
```

```
-- Monitor externals

MakeCodeResident: PUBLIC PROCEDURE [frame: PROGRAM] =
    BEGIN
    MakeResident[SpaceForCode[frame]]
    END;

MakeCodeSwappable: PUBLIC PROCEDURE [frame: PROGRAM] =
    BEGIN
    MakeSwappable[SpaceForCode[frame]]
    END;

SpaceForCode: PROCEDURE [frame: PROGRAM] RETURNS [Handle] =
    BEGIN OPEN f: LOOPHOLE[frame, ControlDefs.GlobalFrameHandle];
    RETURN[GetHandle[PageFromLongPointer[f.code.longbase]]]
    END;

-- Other

VMMHelperProcess: PROCEDURE = -- root of VMM helper process, a monitor external
    BEGIN
    page: PageNumber;
    operation: CachedRegion.Operation;
    outcome: CachedRegion.Outcome;
    HandleException: ENTRY PROCEDURE =
        INLINE BEGIN
        DO
            WITH outcome SELECT FROM
                ok => RETURN;
                regionDMissing => Projection.Touch[page];
                spaceDMissing --[level]-- => Hierarchy.Touch[CachedSpace.Handle[level, page]];
                error --[state]-- =>
                    IF operation.action = activate THEN
                        Runtime.CallDebugger["AddressFault"]
                    ELSE
                        ERROR;
                ENDCASE => ERROR;
            outcome ← CachedRegion.Apply[page, operation].outcome
            ENDLOOP
        END;
    DO
        [page, operation, outcome] ← SwapperException.Await[];
        HandleException[]
        ENDLOOP
    END;

WriteProtectTrap: PROCEDURE = -- handler for processor-generated trap, independent of monitor
    BEGIN
    page: PageNumber;
    state: ControlDefs.StateVector;
    ProcessInternal.DisableInterrupts[]; -- must be first instruction
    state ← STATE; state.dest ← FrameOps.GetReturnLink[]; state.source ← ControlDefs.TrapLink;
    page ← LOOPHOLE[FrameOps.MyLocalFrame[], POINTER TO local ControlDefs.Frame].unused
        ← LOOPHOLE[TrapOps.ReadOTP[], PageNumber]; -- trap parameter
    ProcessInternal.EnableInterrupts[];
    Runtime.CallDebugger["WriteProtectFault"]
    END;

-- Initialization
Process.Detach[FORK VMMHelperProcess[]];
SDDefs.SD[SDDefs.sWriteProtect] ← WriteProtectTrap;

END.
```

Notes

GetAttributes could be augmented by operations for accessing individual space attributes, avoiding unnecessary disk accesses, if performance requirements so dictated.

LOG

Time: April 1978 *By: McJones* *Action: Created file*

Time: June 20, 1978 10:24 AM By: McJones Action: Test of explicit base to Create could overflow
Time: June 23, 1978 5:08 PM By: McJones Action: GetHandle didn't check for page in range
Time: June 26, 1978 2:16 PM By: McJones Action: Create, Delete passed Handle instead of Interval to ApplyRegionOp;
To fix, split ApplyRegionOp into ApplyToInterval, ApplyToSpace
Time: July 10, 1978 12:52 PM By: McJones Action: Added handleMDS module parameter
Time: July 11, 1978 6:16 PM By: McJones Action: Added File., Volume.Unknown logic
Time: July 12, 1978 2:55 PM By: McJones Action: Delete sometimes flushed too much (problem when neighbor pinned)
Time: August 3, 1978 8:58 AM By: McJones Action: Added map logging, helper;
added CallDebugger for address, writeProtect faults
Time: August 29, 1978 9:05 AM By: Redell Action: Installed "graycode" beingRemapped machinery
Time: September 6, 1978 9:15 AM By: McJones Action: Changed Remap to suppress forceOut if writeProtected
Time: September 10, 1978 1:49 PM By: McJones Action: Implemented SpecialSpace operations
Time: September 14, 1978 2:20 PM By: McJones Action: Added LongPointerFromPage, PageFromLongPointer
Time: September 28, 1978 11:57 AM By: McJones CR20.29: Create didn't detect too large base
Time: September 28, 1978 11:57 AM By: McJones CR20.30: HandleException had wrong test for AddressFault
Time: September 29, 1978 3:45 PM By: Redell CR20.32: In-place Remaps are allowed to set write-protect (by passing a read-only capability)
Time: October 16, 1978 9:28 AM By: McJones CR20.51: Fix for 20.29 checked base against count of VM instead of parent
Time: October 16, 1978 9:28 AM By: McJones CR20.61: Added check for interval not pinned to Remap
Time: February 1, 1979 2:07 PM By: McJones CR20.169: Added pageRover to CachedSpace.Desc for Space.Create
Time: February 20, 1979 6:01 PM By: McJones CR20.177: Changed Map, Remap to preserve spaceD.dPinned
Time: March 9, 1979 11:17 AM By: McJones CR20.48,134: Close timing window in Remap[read only]; calls this MakeReadOnly
Time: July 10, 1979 3:43 PM By: Knutsen Action: Make compatible with Teak interfaces.
Time: July 18, 1979 4:48 PM By: McJones Action: Added CreateUniformSwapUnits, DeleteSwapUnits (dummy only)
Time: July 31, 1979 2:08 PM By: McJones Action: Changed MapLog to include writeProtected
Time: August 10, 1979 4:26 PM By: Knutsen Action: Implemented CreateForCode.

```
DIRECTORY
    CachedRegion: FROM "CachedRegion" USING [State],
    CachedSpace: FROM "CachedSpace" USING [Desc, Level],
    Environment: FROM "Environment" USING [Word, wordsPerPage],
    VM: FROM "VM" USING [PageNumber];

STLeaf: DEFINITIONS =

BEGIN

ILeaf: TYPE = RECORD [CARDINAL]; -- Index to Leaf

PLeaf: TYPE = LONG POINTER TO Leaf; -- Pointer to Leaf

sizeLeaf: CARDINAL = Environment.wordsPerPage; -- could be varied for testing

Leaf: TYPE = RECORD [SELECT OVERLAID * FROM
    sTree => [
        cDesc: CDesc, -- descriptors in use
        descFirst: Desc], -- first of array of bound Desc's
    free => [iLeafNext: ILeaf], -- next free page
    fill => [ARRAY [0..sizeLeaf) OF RECORD [Environment.Word]], -- occupies a page
    ENDCASE];

sizeLeafBody: CARDINAL = SIZE[Leaf]-SIZE[CDesc];

CDesc: TYPE = CARDINAL; -- Count of Descriptors

Kind: TYPE = {hierarchy, projection};

Desc: TYPE = RECORD [SELECT OVERLAID Kind FROM
    hierarchy => [
        descH: CachedSpace.Desc],
    projection => [
        page: VM.PageNumber, -- key
        level: CachedSpace.Level,
        state: CachedRegion.State[unmapped..outAlive],
        levelMapped: CachedSpace.Level, -- (assuming state IN Mapped)
        beingRemapped: BOOLEAN], -- (assuming state IN Mapped)
    ENDCASE];

Create: PROCEDURE RETURNS [ILeaf];

Delete: PROCEDURE [ILeaf];

Open: PROCEDURE [ILeaf] RETURNS [PLeaf];

Close: PROCEDURE [ILeaf];

END.
```

LOG

Time: May 16, 1978 11:21 AM By: PMcJ Action: Created file
Time: June 24, 1978 5:38 PM By: PMcJ Action: PageNumber moved to VM
Time: August 4, 1978 2:14 PM By: PMcJ Action: Added beingRemapped to projection Desc

```
DIRECTORY
    File: FROM "File" USING [Capability, Create, MakePermanent, SetSize, Unknown],
    FileTypes: FROM "FileTypes" USING [tSpaceBTree],
    KernelFile: FROM "KernelFile" USING [GetRootFile],
    SimpleSpace: FROM "SimpleSpace" USING [Create, Handle, Page, Map, Unmap],
    STLeaf: FROM "STLeaf",
    Utilities: FROM "Utilities" USING [LongPointerFromPage],
    Volume: FROM "Volume" USING [ID, SystemID],
    VMMPrograms: FROM "VMMPrograms";

STLeafImpl: PROGRAM
    IMPORTS File, KernelFile, SimpleSpace, Utilities, Volume
    EXPORTS STLeaf, VMMPrograms =

BEGIN OPEN STLeaf;

iLeafNull: ILeaf = [LAST[CARDINAL]];

iLeafFree: ILeaf ← iLeafNull;

file: File.Capability;
fileSizeMin: CARDINAL = 10; -- initial size for backing file
filePageNext: CARDINAL ← 0;

Buff: TYPE = RECORD [
    handle: SimpleSpace.Handle,
    open: BOOLEAN,
    iLeafContent: ILeaf];

IBuff: TYPE = [0..1];

rgBuff: ARRAY IBuff OF Buff;
iBuffMru: IBuff ← FIRST[IBuff]; -- for hack replacement algorithm

Close: PUBLIC PROCEDURE [iLeaf: ILeaf] =
    BEGIN
    iBuff: IBuff;
    FOR iBuff IN IBuff DO OPEN rgBuff[iBuff];
        IF iLeafContent = iLeaf THEN GO TO Found
        REPEAT Found =>
            BEGIN
--assert--IF ~open THEN ERROR;
            open ← FALSE;
            RETURN
            END
        ENDLOOP;
--assert--IF ~FALSE THEN ERROR -- not open
    END;

Create: PUBLIC PROCEDURE RETURNS [iLeaf: ILeaf] =
    BEGIN
    IF iLeafFree~ = iLeafNull THEN
        BEGIN
        pLeaf: PLeaf = Open[iLeafFree];
        iLeaf ← iLeafFree;
        iLeafFree ← pLeaf.iLeafNext;
        Close[iLeaf]
        END
    ELSE
        BEGIN
        iLeaf ← [filePageNext];
        filePageNext ← filePageNext + 1;
        IF filePageNext>fileSizeMin THEN
            File.SetSize[file, filePageNext]
        END
    END;

Delete: PUBLIC PROCEDURE [iLeaf: ILeaf] =
    BEGIN
    pLeaf: PLeaf = Open[iLeaf];
    pLeaf.iLeafNext ← iLeafFree;
    iLeafFree ← iLeaf;
    Close[iLeaf]
    END;

Open: PUBLIC PROCEDURE [iLeaf: ILeaf] RETURNS [PLeaf] =
```

```
    BEGIN
    iBuff: IBuff;
    BEGIN
    FOR iBuff IN IBuff DO OPEN rgBuff[iBuff];
        IF iLeafContent = iLeaf THEN
            BEGIN
--assert--IF open THEN ERROR; -- open of already open buffer
            GO TO Found
            END
        ENDLOOP;
    iBuff ← iBuffMru;
    DO OPEN rgBuff[iBuff];
        iBuff ← IF iBuff = LAST[IBuff] THEN FIRST[IBuff] ELSE iBuff + 1;
        IF ~open THEN GO TO Found;
        IF iBuff = iBuffMru THEN EXIT
        ENDLOOP;
    EXITS Found =>
        BEGIN OPEN rgBuff[iBuff];
        pLeaf: PLeaf = LOOPHOLE[Utilities.LongPointerFromPage[SimpleSpace.Page[handle]], PLeaf];
        IF iLeafContent~ = iLeaf THEN
            -- Buffer is not mapped to correct page
            BEGIN
            IF iLeafContent~ = iLeafNull THEN
                SimpleSpace.Unmap[handle]; -- buffer is mapped, but to wrong page; unmap it
            SimpleSpace.Map[handle, [file, iLeaf], FALSE]; -- Map buffer space to [file, iLeaf]
            iLeafContent ← iLeaf
            END;
        open ← TRUE;
        iBuffMru ← iBuff;
        RETURN[pLeaf]
        END
    END;
--assert IF ~FALSE THEN-- RETURN[ERROR] -- not enough buffers
    END;

-- Initialization
BEGIN
systemVolume: Volume.ID = Volume.SystemID[];
iBuff: IBuff;
file ← KernelFile.GetRootFile[FileTypes.tSpaceBTree, systemVolume];
BEGIN
File.SetSize[file, fileSizeMin ! File.Unknown => GO TO Create]
EXITS Create =>
    BEGIN
    file ← File.Create[volume: systemVolume, initialSize: fileSizeMin, type: FileTypes.tSpaceBTree];
    File.MakePermanent[file]
    END
END;
FOR iBuff IN IBuff DO
    rgBuff[iBuff] ← [
        open: FALSE,
        handle: SimpleSpace.Create[1, hyperspace],
        iLeafContent: iLeafNull]
    ENDLOOP
END

END.
```

LOG

Time: May 16, 1978 11:29 AM By: McJones Action: Created file
Time: June 20, 1978 2:26 PM By: McJones Action: Added SimpleSpace calls
Time: June 21, 1978 10:31 AM By: McJones Action: Open didn't set rgBuff.open
Time: June 21, 1978 10:46 AM By: McJones Action: Close didn't stop searching when buffer found
Time: June 26, 1978 5:43 PM By: McJones Action: Added iBuffMru hack
Time: August 2, 1978 10:09 AM By: McJones Action: Enabled File.SetSize calls
Time: August 9, 1978 1:42 AM By: Purcell Action: Use new SimpleSpace
Time: September 6, 1978 9:51 AM By: McJones Action: Use GetRootFile

Time: September 29, 1978 10:52 AM By: McJones CR20.42: Replaced MakePermanent with PutRootFile
Time: August 8, 1979 4:30 PM By: McJones Action: SpecialFile => KernelFile

```
DIRECTORY
    CachedSpace: FROM "CachedSpace" USING [Handle],
    STLeaf: FROM "STLeaf" USING [Desc, Kind],
    VM: FROM "VM" USING [PageCount, PageNumber];

STree: DEFINITIONS =

BEGIN

Key: TYPE = RECORD [SELECT OVERLAID STLeaf.Kind FROM
    hierarchy => [handleH: CachedSpace.Handle],
    projection => [pageP: VM.PageNumber],
    ENDCASE];

Desc: TYPE = STLeaf.Desc;

PDesc: TYPE = LONG POINTER TO Desc;

Get: PROCEDURE [pDescResult: PDesc, key: Key] RETURNS [keyNext: Key];

Insert: PROCEDURE [pDesc: PDesc];

Delete: PROCEDURE [key: Key];

Update: PROCEDURE [pDesc: PDesc];

STreeImpl: PROGRAM [countVM: VM.PageCount, kind: STLeaf.Kind];

END.
```

LOG

Time: May 16, 1978 6:17 PM By: McJones Action: Created file
Time: June 21, 1978 9:36 AM By: McJones Action: Added STreeImpl

```
-- Things to consider:
-- 1) Use LongMove for root insert/delete
-- 2) Collapse merge/balance logic
-- 3) For multi-mds, remove variables from global frame
-- 4) When "too many entries" (root full on split), clean up and signal
-- 5) Move cDesc from leaves to root; improve merge/balance logic

DIRECTORY
    CachedSpace: FROM "CachedSpace" USING [handleVM, Level],
    InlineDefs: FROM "InlineDefs" USING [LowHalf],
    STLeaf: FROM "STLeaf",
    STree: FROM "STree",
    Utilities: FROM "Utilities" USING [LongMove],
    VM: FROM "VM" USING [PageCount];

STreeImpl: PROGRAM [countVM: VM.PageCount, kind: STLeaf.Kind]
    IMPORTS InlineDefs, STLeaf, Utilities
    EXPORTS STree =

BEGIN OPEN STLeaf, STree;

-- Constants (initialized later):

sizeDesc: CARDINAL; -- SIZE[kind Desc]

cDescMax: CDesc; -- maximum number of descriptors per leaf

cDescMin: CDesc; -- threshold for balancing

keyTop: Key; -- upper bound for kind Key

-- The root:

-- If cKey = 0, then mplILeafKey has no entries; rglLeaf[0] specifies the only leaf page.
-- If cKey>0, then the values mplILeafKey[i], for i IN [1..cKey], are ascending:
--     leaf rglLeaf[0] contains descriptors matching keys<mplILeafKey[1];
--     leaf rglLeaf[i] contains descriptors matching keys IN [mplILeafKey[i]..mplILeafKey[i + 1]), i IN [1..cKey);
--     leaf rglLeaf[cKey] contains descriptors matching keys> = mplILeafKey[cKey].

IlLeaf: TYPE = CARDINAL; -- Index of lLeaf entry in root
cKeyMax: CARDINAL = 20;
cKey: CARDINAL;
mplILeafKey: ARRAY IlLeaf(0..cKeyMax] OF Key;
rglLeaf: ARRAY IlLeaf[0..cKeyMax] OF lLeaf;

KeyFromPDesc: PROCEDURE [pDesc: PDesc] RETURNS [Key] =
    BEGIN
    WITH pDesc SELECT kind FROM
        hierarchy => RETURN[[hierarchy[[level: descH.level, page: descH.interval.page]]]];
        ENDCASE -- projection -- => RETURN[[projection[pDesc.page]]]
    END;

SearchRoot: PROCEDURE [key: Key] RETURNS [ilLeaf: IlLeaf] =
    BEGIN
    FOR ilLeaf ← 0, ilLeaf + 1 DO
        IF ilLeaf = cKey OR (WITH mplILeafKey[ilLeaf + 1] SELECT kind FROM
                hierarchy => key.handleH.level<handleH.level OR
                    key.handleH.level = handleH.level AND key.handleH.page<handleH.page,
                ENDCASE -- projection -- => key.pageP<pageP) THEN
            EXIT
        ENDLOOP
    END;

SearchLeaf: PROCEDURE [key: Key, pLeaf: PLeaf] RETURNS [found: BOOLEAN, pDesc: PDesc, last: BOOLEAN] =
    BEGIN
    iDesc: CARDINAL;
```

```
    pDescNext: PDesc;
--assert--IF pLeaf.cDesc = 0 THEN ERROR;
    found ← TRUE; last ← FALSE;
    pDesc ← @pLeaf.descFirst;
    FOR iDesc IN [0..pLeaf.cDesc-(IF kind = projection THEN 1 ELSE 0)) DO
        pDescNext ← pDesc + sizeDesc;
        WITH pDesc SELECT kind FROM
            hierarchy =>
                BEGIN OPEN descH;
                IF key.handleH.level = level AND key.handleH.page<interval.page + interval.count OR key.handleH.level<level
                        THEN
                    BEGIN
                    IF key.handleH.level~ = level OR key.handleH.page<interval.page THEN found ← FALSE;
                    GO TO CheckLast
                    END
                END;
            projection =>
                IF key.pageP<pDescNext.page THEN
                    GO TO CheckLast;
            ENDCASE;
        pDesc ← pDescNext
        REPEAT
            CheckLast => IF iDesc + 1 = pLeaf.cDesc THEN last ← TRUE;
            FINISHED => BEGIN last ← TRUE; IF kind = hierarchy THEN found ← FALSE END
        ENDLOOP
    END;

UpdateRoot: PROCEDURE [ilLeaf: IlLeaf, pLeaf: PLeaf] =
    BEGIN
    IF ilLeaf~ = 0 AND pLeaf.cDesc~ = 0 THEN
        mpIlLeafKey[ilLeaf] ← KeyFromPDesc[@pLeaf.descFirst]
    END;

Delete: PUBLIC PROCEDURE [key: Key] =
    BEGIN
    ilLeaf: IlLeaf = SearchRoot[key];
    pLeaf: PLeaf = STLeaf.Open[rgILeaf[ilLeaf]];
    matching: BOOLEAN;
    pDesc: PDesc;
    [matching, pDesc] ← SearchLeaf[key, pLeaf];
--assert--IF ~matching THEN ERROR;
    -- Move descriptors following pDesc up one place
    pLeaf.cDesc ← pLeaf.cDesc-1;
    Utilities.LongMove[
        pSource: pDesc + sizeDesc,
        size: InlineDefs.LowHalf[@pLeaf.descFirst + sizeDesc*pLeaf.cDesc-pDesc],
        pSink: pDesc];
    -- Update root entry in case deleted descriptor was first on leaf
    UpdateRoot[ilLeaf, pLeaf];
    -- If number of descriptors remaining on leaf is small enough, perform merge or balance
    IF pLeaf.cDesc = 0 AND ilLeaf = cKey THEN
        BEGIN
        -- Delete last leaf
        STLeaf.Close[rgILeaf[ilLeaf]];
        STLeaf.Delete[rgILeaf[ilLeaf]];
        cKey ← cKey-1;
        RETURN
        END;
    IF pLeaf.cDesc<cDescMin AND ilLeaf<cKey THEN
        BEGIN
        ilLeafNext: IlLeaf = ilLeaf + 1;
        pLeafNext: PLeaf = STLeaf.Open[rgILeaf[ilLeafNext]];
        IF pLeaf.cDesc + pLeafNext.cDesc<= cDescMax THEN
            -- Merge higher neighbor into this leaf
            BEGIN
            ilLeafAfter: IlLeaf;
            -- Append contents of merged leaf
            Utilities.LongMove[
                pSource: @pLeafNext.descFirst,
                size: sizeDesc*pLeafNext.cDesc,
                pSink: @pLeaf.descFirst + sizeDesc*pLeaf.cDesc];
            pLeaf.cDesc ← pLeaf.cDesc + pLeafNext.cDesc;
```

```
            -- Delete merged leaf
            STLeaf.Close[rglLeaf[ilLeafNext]];
            STLeaf.Delete[rglLeaf[ilLeafNext]];
            -- Delete its root entry
            cKey ← cKey-1;
            FOR ilLeafAfter IN [ilLeafNext..cKey] DO
                mpIlLeafKey[ilLeafAfter] ← mpIlLeafKey[ilLeafAfter + 1];
                rglLeaf[ilLeafAfter] ← rglLeaf[ilLeafAfter + 1]
                ENDLOOP
            END
        ELSE
            -- Balance with next leaf
            BEGIN
            cDescDelta: CDesc = (pLeafNext.cDesc-pLeaf.cDesc)/2; -- >0, or we would have merged
            -- Append first cDescDelta descriptors from next leaf to this one
            Utilities.LongMove[
                pSource: @pLeafNext.descFirst,
                size: sizeDesc*cDescDelta,
                pSink: @pLeaf.descFirst + sizeDesc*pLeaf.cDesc];
            pLeaf.cDesc ← pLeaf.cDesc + cDescDelta;
            -- Delete first cDescDelta descriptors of next leaf
            pLeafNext.cDesc ← pLeafNext.cDesc-cDescDelta;
            Utilities.LongMove[
                pSource: @pLeafNext.descFirst + sizeDesc*cDescDelta,
                size: sizeDesc*pLeafNext.cDesc,
                pSink: @pLeafNext.descFirst];
            -- Update root entry since identity of first descriptor changed
            UpdateRoot[ilLeafNext, pLeafNext];
            STLeaf.Close[rglLeaf[ilLeafNext]]
            END
        END;
    STLeaf.Close[rglLeaf[ilLeaf]]
    END;

Get: PUBLIC PROCEDURE [pDescResult: PDesc, key: Key] RETURNS [keyNext: Key] =
    BEGIN
    ilLeaf: IlLeaf = SearchRoot[key];
    pLeaf: PLeaf = STLeaf.Open[rglLeaf[ilLeaf]];
    found, last: BOOLEAN;
    pDesc: PDesc;
    [found, pDesc, last] ← SearchLeaf[key, pLeaf];
    IF ~found THEN
        WITH pDescResult SELECT kind FROM
            hierarchy => descH.state ← missing;
            projection => ERROR;
            ENDCASE
    ELSE
        Utilities.LongMove[pSource: pDesc, size: sizeDesc, pSink: pDescResult];
    SELECT TRUE FROM
        ~last => keyNext ← KeyFromPDesc[pDesc + (IF found THEN sizeDesc ELSE 0)];
        -- last AND -- ilLeaf<cKey => keyNext ← mpIlLeafKey[ilLeaf + 1];
        -- last AND ilLeaf = cKey -- ENDCASE => keyNext ← keyTop;
    STLeaf.Close[rglLeaf[ilLeaf]]
    END;

Insert: PUBLIC PROCEDURE [pDesc: PDesc] =
    BEGIN
    key: Key = KeyFromPDesc[pDesc];
    ilLeaf: IlLeaf;
    pLeaf: PLeaf;
    inserted: BOOLEAN;
    DO
        ilLeaf ← SearchRoot[key];
        pLeaf ← STLeaf.Open[rglLeaf[ilLeaf]];
        IF pLeaf.cDesc = cDescMax THEN
            -- Leaf is full; split it and try again
            BEGIN
            ilLeafNew: IlLeaf = ilLeaf + 1;
            ilLeafAfter: IlLeaf;
            -- Create new leaf to follow current one
            iLeafNew: ILeaf = STLeaf.Create[];
            pLeafNew: PLeaf = STLeaf.Open[iLeafNew];
```

```
            cDescNew: CDesc = pLeaf.cDesc/2;
--assert--IF cKey = cKeyMax THEN ERROR;
            -- Move last cDescNew descriptors of this leaf to new one
            pLeaf.cDesc ← pLeaf.cDesc-cDescNew;
            pLeafNew.cDesc ← cDescNew;
            Utilities.LongMove[
                pSource: @pLeaf.descFirst + sizeDesc*pLeaf.cDesc,
                size: sizeDesc*cDescNew,
                pSink: @pLeafNew.descFirst];
            -- Insert new root entry
            FOR ilLeafAfter DECREASING IN [ilLeafNew..cKey] DO
                mpIlLeafKey[ilLeafAfter + 1] ← mpIlLeafKey[ilLeafAfter];
                rgIlLeaf[ilLeafAfter + 1] ← rgIlLeaf[ilLeafAfter]
                ENDLOOP;
            cKey ← cKey + 1;
            rgIlLeaf[ilLeafNew] ← iLeafNew;
            UpdateRoot[ilLeafNew, pLeafNew];
            STLeaf.Close[iLeafNew];
            -- Restart
            inserted ← FALSE
            END
        ELSE
            -- Leaf is not full; perform insert and exit
            BEGIN
            matching: BOOLEAN;
            pDescTarget: PDesc;
            [matching, pDescTarget] ← SearchLeaf[key, pLeaf];
            SELECT kind FROM
                hierarchy => IF matching THEN ERROR;
                projection => pDescTarget ← pDescTarget + sizeDesc; -- insert after current matching entry
                ENDCASE;
            -- Move all descriptors not before pDescTarget down one place
            Utilities.LongMove[
                pSource: pDescTarget,
                size: InlineDefs.LowHalf[@pLeaf.descFirst + sizeDesc*pLeaf.cDesc-pDescTarget],
                pSink: pDescTarget + sizeDesc];
            -- Insert descriptor
            Utilities.LongMove[pSource: pDesc, size: sizeDesc, pSink: pDescTarget];
            pLeaf.cDesc ← pLeaf.cDesc + 1;
            -- Update root entry in case inserted descriptor is first on leaf
            UpdateRoot[ilLeaf, pLeaf];
            -- Indicate finished
            inserted ← TRUE
            END;
        STLeaf.Close[rgIlLeaf[ilLeaf]];
        IF inserted THEN EXIT
        ENDLOOP
    END;

Update: PUBLIC PROCEDURE [pDesc: PDesc] =
    BEGIN
    key: Key = KeyFromPDesc[pDesc];
    ilLeaf: IlLeaf = SearchRoot[key];
    pLeaf: PLeaf = STLeaf.Open[rgIlLeaf[ilLeaf]];
    matching: BOOLEAN;
    pDescTarget: PDesc;
    [matching, pDescTarget] ← SearchLeaf[key, pLeaf];
--assert--IF ~matching THEN ERROR;
    -- Update descriptor
    Utilities.LongMove[pSource: pDesc, size: sizeDesc, pSink: pDescTarget];
    -- Update root entry in case updated descriptor is first on leaf
    UpdateRoot[ilLeaf, pLeaf];
    STLeaf.Close[rgIlLeaf[ilLeaf]];
    END;

-- Initialize
BEGIN
iLeaf: IlLeaf = STLeaf.Create[];
pLeaf: PLeaf = STLeaf.Open[iLeaf];
cKey ← 0;
rgIlLeaf[0] ← iLeaf;
pLeaf.cDesc ← 1;
```

```
SELECT kind FROM
    hierarchy =>
        BEGIN
        sizeDesc ← SIZE[hierarchy Desc];
        keyTop ← [hierarchy[[level: LAST[CachedSpace.Level], page: CachedSpace.handleVM.page + countVM]]];
        pLeaf.descFirst ← [hierarchy[[
            level: CachedSpace.handleVM.level,
            interval: [CachedSpace.handleVM.page, countVM],
            dPinned: , dDirty: , -- don't care
            pinned: FALSE,
            state: unmapped,
            writeProtected: , -- don't care
            dataOrFile: , -- don't care
            pageRover: CachedSpace.handleVM.page,
            vp: short[]]]] -- don't care
        END;
    -- projection -- ENDCASE =>
        BEGIN
        sizeDesc ← SIZE[projection Desc];
        keyTop ← [projection[CachedSpace.handleVM.page + countVM]];
        pLeaf.descFirst ← [projection[
            page: CachedSpace.handleVM.page,
            level: CachedSpace.handleVM.level,
            state: unmapped,
            levelMapped: , -- don't care
            beingRemapped: ]] -- don't care
        END;
STLeaf.Close[iLeaf];

cDescMax ← sizeLeafBody/sizeDesc;

cDescMin ← cDescMax/3; -- must be < cDescMax/2 for merge/balance logic
END

END.
```

LOG

Time: May 5, 1978 2:55 PM By: McJones Action: Created file
Time: June 22, 1978 4:06 PM By: McJones Action: keyTop.page was one too large (both kinds)
Time: June 26, 1978 1:06 PM By: McJones Action: projection Insert put new entry before old matching entry
Time: July 17, 1978 12:06 PM By: McJones Action: Upper bound for split root update loop was) instead of];
Delete allowed last page to become empty;
Delete wouldn't merge second-to-last page;
Upper bound for merge root update loop was) instead of];
Balance updated wrong root entry
Time: August 31, 1978 11:59 AM By: McJones Action: Bugs in Get's keyNext computation
Time: September 9, 1978 5:02 PM By: McJones Action: Added pinned to CachedSpace constructor
Time: September 22, 1978 6:33 PM By: McJones Action: SearchLeaf didn't set last when found = TRUE
Time: February 1, 1979 2:24 PM By: McJones Action: Added pageRover to CachedSpace.Desc

```
DIRECTORY
    CachedRegion: FROM "CachedRegion" USING [Apply, Desc, Outcome, pageTop],
    CachedSpace: FROM "CachedSpace" USING [Desc, Get, Handle, Level],
    Environment: FROM "Environment" USING [maxPagesInMDS, PageCount, PageNumber],
    File: FROM "File" USING [nullID],
    Hierarchy: FROM "Hierarchy" USING [Insert],
    MapLog: FROM "MapLog" USING [WriteLog],
    Projection: FROM "Projection" USING [Split, TranslateLevel],
    SimpleSpace: FROM "SimpleSpace" USING [DisableInitialization],
    StoragePrograms: FROM "StoragePrograms",
    STree: FROM "STree" USING [STreeImpl],
    VM: FROM "VM" USING [Interval],
    VMMPrograms: FROM "VMMPrograms",
    Volume: FROM "Volume" USING [SystemID];

VMMControl: PROGRAM [countVM: Environment.PageCount, pMapLogDesc: LONG POINTER]
    IMPORTS CachedRegion, CachedSpace, Hierarchy, MapLog, Projection, SimpleSpace, STreeHier: STree,
            STreeProj: STree, VMMPrograms, Volume
    EXPORTS StoragePrograms SHARES File =

BEGIN

CreateSpace: PROCEDURE [level: CachedSpace.Level, interval: VM.Interval, pSpaceD: POINTER TO CachedSpace.Desc] =
    -- Creates hierarchy & projection entries for one space; dirty cache entries should already exist
    BEGIN OPEN interval; -- page, count
    Hierarchy.Insert[CachedSpace.Handle[level, page], count];
    Projection.Split[page]; Projection.Split[page + count]; Projection.TranslateLevel[pageMember: page, delta: 1];
    IF pSpaceD~ = NIL AND pSpaceD.window.file.fID~ = File.nullID THEN MapLog.WriteLog[interval, pSpaceD]
    END;

FillHierarchyAndProjection: PROCEDURE =
    BEGIN
    pageRegion, pageSpace: Environment.PageNumber;
    outcome: CachedRegion.Outcome;
    region: CachedRegion.Desc;
    SimpleSpace.DisableInitialization[];
    CreateSpace[first64K.level, VM.Interval[first64K.page, Environment.maxPagesInMDS], NIL]; -- create first64K space
    pageRegion ← 0;
    pageSpace ← CachedRegion.pageTop; -- page not contained in any space
    DO
        [outcome, pageRegion] ← CachedRegion.Apply[pageRegion,
                [ifMissing: report, ifCheckedOut: wait, afterForking: , vp: get[andResetDDirty: FALSE, pDescResult:
                @region]]];
        SELECT outcome.kind FROM
            ok =>
                BEGIN
                subSpace: BOOLEAN = region.level = (IF region.interval.page<256 THEN 3 ELSE 2);
                levelSpace: CachedSpace.Level = region.level-(IF subSpace THEN 1 ELSE 0);
                space: CachedSpace.Desc;
                CachedSpace.Get[@space, [level: levelSpace, page: region.interval.page]];
                IF ~subSpace OR space.interval.page~ = pageSpace -- first subspace seen with this parent -- THEN
                    BEGIN -- put potentially mapped space into VMMgr database
                    CreateSpace[space.level, space.interval, IF space.state = mapped THEN @space ELSE NIL];
                    pageSpace ← space.interval.page
                    END;
                IF subSpace THEN CreateSpace[region.level, region.interval, NIL]
                END;
            regionDMissing => NULL;
            ENDCASE => ERROR;
        IF pageRegion = CachedRegion.pageTop THEN EXIT
        ENDLOOP
    END;

first64K: CachedSpace.Handle = [level: FIRST[CachedSpace.Level] + 1, page: 0];

START VMMPrograms.MapLogImpl[pMapLogDesc: pMapLogDesc];
START VMMPrograms.STLeafImpl[];
START STreeHier.STreeImpl[countVM: countVM, kind: hierarchy];
START STreeProj.STreeImpl[countVM: countVM, kind: projection];
START VMMPrograms.HierarchyImpl;
START VMMPrograms.ProjectionImpl[countVM: countVM];
FillHierarchyAndProjection[];
START VMMPrograms.SpaceImpl[countVM: countVM, volumeData: Volume.SystemID[], handleMDS: first64K];
```

END.

LOG

Time: June 20, 1978 5:58 PM By: McJones Action: Created file
Time: June 22, 1978 6:12 PM By: McJones Action: Added projection(/hierarchy?) init
Time: June 23, 1978 1:23 PM By: McJones Action: Added countVM parameter to ProjectionImpl, SpaceImpl
Time: July 10, 1978 10:18 AM By: McJones Action: Added creation of first64K
Time: July 18, 1978 10:59 AM By: McJones Action: Added countReal, ...
Time: August 2, 1978 10:18 AM By: McJones Action: Added MapLogImpl
Time: August 7, 1978 8:11 PM By: Purcell Action: Call SimpleSpace.DisableInitialization;
new Hierarchy/Projection filling logic
Time: August 29, 1978 4:14 PM By: McJones Action: (De)typed pMapLogDesc
Time: September 6, 1978 10:18 AM By: McJones Action: Fixed test for first subspace of parent
Time: September 15, 1978 5:20 PM By: McJones Action: Added map logging
Time: July 11, 1979 3:28 PM By: McJones Action: Nonparent mapped space handled incorrectly
Time: July 31, 1979 2:03 PM By: McJones Action: Changed MapLog to include writeProtected

```
DIRECTORY
    CachedSpace: FROM "CachedSpace" USING [Handle],
    VM: FROM "VM" USING [PageCount],
    Volume: FROM "Volume" USING [ID];

VMMPrograms: DEFINITIONS =

BEGIN

HierarchyImpl: PROGRAM;

MapLogImpl: PROGRAM [pMapLogDesc: LONG POINTER];

ProjectionImpl: PROGRAM [countVM: VM.PageCount];

SpaceImpl: PROGRAM [countVM: VM.PageCount, volumeData: Volume.ID, handleMDS: CachedSpace.Handle];

STLeafImpl: PROGRAM;

-- STreeImpl is defined in STree (because of multiple instances)

END.
```

LOG

Time: June 20, 1978 4:30 PM By: McJones Action: Created file
Time: June 23, 1978 10:23 AM By: McJones Action: Added countVM parameter to ProjectionImpl, SpaceImpl
Time: July 10, 1978 10:20 AM By: McJones Action: Added mds parameter to SpaceImpl
Time: August 1, 1978 10:07 AM By: McJones Action: Added MapLogImpl
Time: August 29, 1978 4:24 PM By: McJones Action: (De)typed pDesc parameter to MapLogImpl

-- **FileMgr**.config *(last edited by: McJones on: August 20, 1979 9:45 AM)*

```
PACK
    FileImpl,
    VolAllocMapImpl,
    VolFileMapImpl,
    VolumeImpl;

FileMgr: CONFIGURATION
    IMPORTS DiskChannel, FileCache, FilePageTransfer, FilerException, LabelTransfer, PilotSwitches, Process, SimpleSpace,
            Space, SubVolume, System, SystemInternal, Utilities
    EXPORTS File, KernelFile, SpecialFile, StoragePrograms, Volume =

BEGIN

FileImpl;

VolAllocMapImpl;

VolFileMapImpl;

VolumeImpl;

END.
```

LOG

Time: June 5, 1978 10:32 AM By: Purcell Action: Created file
Time: July 21, 1978 5:42 PM By: Horsley Action: Removed FMPrograms from EXPORTS
Time: August 3, 1978 2:03 PM By: McJones Action: Added code packing
Time: August 11, 1978 1:35 AM By: Purcell Action: Added Utilities to IMPORTS
Time: August 29, 1978 2:02 PM By: Purcell Action: LogicalVolume => SpecialFile (in EXPORTS)
Time: September 23, 1978 4:11 PM By: Horsley Action: Removed FilePointImpl.HelperImpl; added Space to IMPORTS
Time: March 21, 1979 9:25 AM By: Horsley Action: Converted to Mesa 5.0
Time: August 20, 1979 9:45 AM By: McJones Action: Added PilotSwitches to IMPORTS, KernelFile to EXPORTS

```
-- FileMgrTest.mesa (last edited by: Purcell on: July 21, 1978 7:13 PM) --

DIRECTORY
    AltoFileDefs: FROM "AltoFileDefs",
    File: FROM "File",
    FileInternal: FROM "FileInternal",
    FilePoint: FROM "FilePoint",
    --LogicalVolume: FROM "LogicalVolume",--      --not repeatable
    PilotClient: FROM "PilotClient",
    System: FROM "System",
    SystemInternal: FROM "SystemInternal",
    Volume: FROM "Volume";

InitializePilotClient: PROGRAM    -- must have this name to be started by pilot
        IMPORTS File, --LogicalVolume,-- Volume
        EXPORTS PilotClient
        SHARES File, System, SystemInternal =
BEGIN

maxFileN: CARDINAL = 64;
table: ARRAY [0..maxFileN) OF Entry;
Entry: TYPE = RECORD[
    file: File.Capability,
    exists: BOOLEAN,
    temporary: BOOLEAN,
    size: File.PageCount];

FileTestAtYourServiceProceedWhenReady: SIGNAL = CODE;
MatchBreak: SIGNAL = CODE;
TestError: SIGNAL = CODE;

breakN: CARDINAL ← maxFileN;  -- for matching fileN
breaker: CARDINAL ← 0;

altoFlag: BOOLEAN ← TRUE;
altoFile: File.Capability ← File.Capability[System.FileID[SystemInternal.UniversalID[SystemInternal.altoFPSeries,
        altoFP[LOOPHOLE[AltoFileDefs.DirFP]]]], FileInternal.maxPermissions];
altoSize: File.PageCount ← 0;
altoVolume: Volume.ID ← Volume.nullID;

ApplyTest: PROCEDURE[proc: PROCEDURE[action: CARDINAL] RETURNS[success: BOOLEAN], weight: DESCRIPTOR FOR ARRAY OF
        CARDINAL] =
    BEGIN
    action, counter, sample, maxSample: CARDINAL ← 0;
    FOR action IN [0..LENGTH[weight]) DO maxSample ← maxSample + weight[action]; ENDLOOP;
    DO
        FOR counter ← (action ← 0), counter + 1 WHILE proc[action] DO    --init on zero
            IF breaker = counter THEN SIGNAL FileTestAtYourServiceProceedWhenReady;
            sample ← Random[maxSample];        --randomSeed may be inited with action zero
            FOR action IN [0..LENGTH[weight]) UNTIL sample<weight[action] DO
                sample ← sample - weight[action];
                ENDLOOP;
            ENDLOOP;
        SIGNAL TestError;
        breaker ← counter-1;
        ENDLOOP;
    END;

Test: PROCEDURE =
    BEGIN
    ApplyTest[TestOne, DESCRIPTOR[weight]];
    END;

weight: ARRAY [0..14] OF CARDINAL ← [0,10,10,10,10,10,10, 1, 0, 10,10,10,5,0, 20];

tries: CARDINAL ← 5;

TestOne: PROCEDURE [action: CARDINAL] RETURNS [success: BOOLEAN] =
    BEGIN
    op: {init, getTemp, getType, getSize, makePermanent, delete, create, close, filePoint, setSize0, setSize1, setSize2,
```

```
            setSize20, setSize100, setSizeDelta} ← LOOPHOLE[action];
    fileN: CARDINAL;
    freePageCount: Volume.PageCount ← Volume.GetAttributes[Volume.nullID].freePageCount;
    success ← TRUE;
    IF action = 0 THEN -- init
        BEGIN
        randomSeed ← 3021B;
        FOR fileN IN [0..maxFileN) DO table[fileN].exists ← FALSE; ENDLOOP;
        --LogicalVolume.Init[LogicalVolume.GetHandle[Volume.nullID].channel];--      --not repeatable
        altoVolume ← File.GetAttributes[altoFile].volume;
        IF altoFlag THEN altoSize ← File.GetSize[altoFile];
        RETURN;
        END;
    THROUGH [0..tries) DO
        fileN ← Random[maxFileN];
        IF table[fileN].exists = (action # 6) THEN EXIT
        REPEAT
            FINISHED => RETURN;
        ENDLOOP;
    BEGIN OPEN table[fileN];
    IF fileN = breakN THEN SIGNAL MatchBreak;
    SELECT action FROM
        1 => success ← (File.GetAttributes[file].temporary = temporary);
        2 => success ← (File.GetAttributes[file].type = fileN + 10);
        3 => success ← (File.GetSize[file] = size);
        4 => BEGIN File.MakePermanent[file]; temporary←FALSE; END;
        5 => BEGIN File.Delete[file]; exists ← FALSE; END;
        6 => IF freePageCount > 5 THEN table[fileN] ← Entry[
                    file: File.Create[Volume.nullID, 0, File.Type[fileN + 10]],
                    exists: TRUE,
                    temporary: TRUE,
                    size: 0];
        7 => Volume.Close[Volume.SystemID[]];      -- hack to exercise various close/open or rebuilds
        --8 => success ← (FilePoint.GetFilePoint[file, 0, 10b].count <= size);--      -- exercise FilePoint, but can't reach through pilot
                interfaces
        ENDCASE =>
            BEGIN
            size ← MIN[size + freePageCount - MIN[freePageCount, LONG[20]], SELECT action FROM
                9 => 0,
                10 => 1,
                11 => 2,
                12 => Random[20],
                13 => Random[100],
                ENDCASE => size - MIN[size, LONG[3]] + Random[7]];
            File.SetSize[file, size];
            END;
    END;    --of OPEN table[fileN]
    END;

randomSeed: CARDINAL ← 3021B;

Random: PROCEDURE [limit: CARDINAL] RETURNS [n: CARDINAL] =
    BEGIN
    RETURN[(randomSeed ← 24205B*randomSeed + 33031B) MOD limit];
    END;

Test[]

END.
```

LOG

Time: June 21, 1978 5:41 PM By: Purcell Action: Created file from fileImpl.mesa
Time: June 30, 1978 2:26 PM By: Purcell Action: renamed from TestFileImpl to FileTestClient, PilotClientImpl
Time: July 17, 1978 2:01 PM By: Purcell Action: renamed from FileTestClient to FileMgrTest and added close

```
DIRECTORY
    AltoFileDefs: FROM "AltoFileDefs" USING [FA, LD],
    BootChannel: FROM "BootChannel" USING [Location],
    DiskChannel: FROM "DiskChannel" USING [Address, Handle, Drive, GetAttributes, GetDriveTag],
    Environment: FROM "Environment" USING [PageCount, PageNumber],
    File: FROM "File" USING [Capability, delete, ErrorType, grow, ID, lastPageNumber, nullCapability, PageCount,
        PageNumber, Permissions, read, shrink, Type, write],
    FileCache: FROM "FileCache" USING [GetFilePtrs, GetPageGroup, SetFile, SetPageGroup, ReturnFilePtrs],
    FileInternal: FROM "FileInternal" USING [Descriptor, FilePtr, maxPermissions, PageGroup],
    FilePageLabel: FROM "FilePageLabel" USING [GetFilePage, GetType, Label, LabelCheckSum, nullID, SetFilePage,
        tBootFile],
    FilePageTransfer: FROM "FilePageTransfer" USING [Initiate, Request],
    FilerException: FROM "FilerException" USING [Await],
    FMPrograms: FROM "FMPrograms" USING [VolAllocMapImpl, VolFileMapImpl, VolumeImpl],
    Inline: FROM "Inline" USING [BITAND, LowHalf],
    KernelFile: FROM "KernelFile" USING [DeviceHandle],
    LabelTransfer: FROM "LabelTransfer" USING [LabelType, ReadLabel, VerifyLabels, WriteLabelsAndData, WriteLabels],
    LogicalVolume: FROM "LogicalVolume" USING [create, FileAccessProc, FileVolumeAccess, Handle, noOp,
        nullVolumePage, Operations, PageNumber, PutRootFile, systemID, VolumeAccess],
    MStore: FROM "MStore" USING [Promise],
    PhysicalVolume: FROM "PhysicalVolume" USING [PageNumber],
    PilotFileTypes: FROM "PilotFileTypes" USING [PilotRootFileType, PilotVFileType, tBeingMoved, tBeingReplicated,
        tBootFile, tFreePage],
    PilotSwitches: FROM "PilotSwitches" USING [switches--A,F--],
    Process: FROM "Process" USING [Detach],
    Runtime: FROM "Runtime" USING [CallDebugger],
    SimpleSpace: FROM "SimpleSpace" USING [Create, Handle, Map, Page, Unmap],
    Space: FROM "Space" USING [Create, defaultBase, Delete, Handle, Map, Remap, Unmap, virtualMemory, WindowOrigin],
    SpecialFile: FROM "SpecialFile" USING [Link],
    StoragePrograms: FROM "StoragePrograms" USING [BootFileIDs, DebugClass],
    SubVolume: FROM "SubVolume" USING [Find, GetPageAddress, Handle],
    System: FROM "System" USING [GetUniversalID, VolumeID],
    SystemInternal: FROM "SystemInternal" USING [altoFPSeries, Unimplemented, UniversalID],
    Utilities: FROM "Utilities" USING [LongPointerFromPage],
    VMMapLog: FROM "VMMapLog" USING [Entry, PilotFID],
    VolAllocMap: FROM "VolAllocMap" USING [AllocPageGroup, FreePageGroup],
    VolFileMap: FROM "VolFileMap" USING [DeletePageGroup, GetNextFileProc, GetPageGroup, InsertPageGroup],
    Volume: FROM "Volume" USING [GetNext, ID, InsufficientSpace, nullID, Unknown],
    VolumeInternal: FROM "VolumeInternal" USING [altoVolume];

FileImpl: MONITOR -- to protect pageBuffer and to serialize changes to files
    [bootFile: POINTER TO disk BootChannel.Location, pDebuggerIDs: POINTER TO StoragePrograms.BootFileIDs] RETURNS
        [debugClass: StoragePrograms.DebugClass, debuggerDevice: UNSPECIFIED--OISProcessorFace.DeviceHandle--]
    IMPORTS DiskChannel, FileCache, FilePageLabel, FilePageTransfer, FilerException, FMPrograms, Inline, LabelTransfer,
        LogicalVolume, MStore, PilotSwitches, Process, Runtime, SimpleSpace, Space--(Space!?)--, SubVolume,
        System, SystemInternal, Utilities, VolAllocMap, VolFileMap, Volume
    EXPORTS File, KernelFile, SpecialFile, StoragePrograms
    SHARES File, FilePageLabel, System, SystemInternal =
BEGIN OPEN File, FileInternal;

FileAccessProc: TYPE = LogicalVolume.FileAccessProc;

-- Set by FileChange; used by ForceOutAttributes
volumePage: LogicalVolume.PageNumber; -- protected by monitor (as is buffer)

-- Size Used for Move & Replicate Imutable
spaceSize: Environment.PageCount = 1; -- should be at least one disk track


-- One-page buffer used by FileChange and GetAltoSize
pageBuffer: SimpleSpace.Handle = SimpleSpace.Create[1, hyperspace];
bufferPointer: LONG POINTER = Utilities.LongPointerFromPage[SimpleSpace.Page[pageBuffer]];

-- Maintained by AltoSize
cacheA, cacheB: RECORD[file: File.ID, size: PageCount] ← [FilePageLabel.nullID, 0];


-- Multi-page buffer used by MakeBootable
longBufferSize: CARDINAL = 20; -- increase batch size to speed up boot chain installation
longBuffer: SimpleSpace.Handle = SimpleSpace.Create[longBufferSize, hyperspace];
longBufferPage: Environment.PageNumber = SimpleSpace.Page[longBuffer];


Error: PUBLIC ERROR [type: ErrorType] = CODE;
```

```
InvalidParameters: PUBLIC ERROR = CODE;
LabelError: SIGNAL = CODE;
Unknown: PUBLIC ERROR [file: Capability] = CODE;

-- MONITOR EXTERNAL PROCEDURES

Create: PUBLIC PROCEDURE [volume: System.VolumeID, initialSize: PageCount, type: Type]
    RETURNS [file: Capability] =
  BEGIN
  file ← Capability[File.ID[System.GetUniversalID[]], FileInternal.maxPermissions];
  CreateWithID[volume, initialSize, type, file];
  RETURN[file];
  END;

-- Create file given the file identifier (Called ONLY from Create???)
CreateWithID: PROCEDURE [volume: System.VolumeID, initialSize: PageCount, type: File.Type, file: Capability] = INLINE
  BEGIN
  fileD: local FileInternal.Descriptor ← [file.fID, volume, local[FALSE, TRUE, 0, type]];
  IF type IN PilotFileTypes.PilotVFileType THEN ERROR Error[reservedType];
  -- Should check for existence of file with this ID
  LogicalVolume.FileVolumeAccess[Grow1, @fileD, LogicalVolume.create, @initialSize]
  END;

Delete: PUBLIC PROCEDURE [file: Capability] =
  BEGIN
  zero: File.PageCount ← 0;
  FileChange[Shrink1, @file, File.delete, @zero];
  END;

DeleteTemps: PUBLIC PROCEDURE [volume: Volume.ID] =
  BEGIN
  file: File.Capability ← File.nullCapability;
  immutable, temporary: BOOLEAN;

  UNTIL (file ← GetNextFile[volume, file]) = File.nullCapability DO
    --The catch phrase in the following statement is a kludge to compensate for the shortcomings of File.Delete and the ine
        scavenger that cause crashes during deletion of a page group to leave an incomplete set of the group's pages on
        disk. It should be removed when the intentions page and the real scavenger are implemented.
    [immutable: immutable, temporary: temporary] ← GetAttributes[file ! Unknown => BEGIN
        Runtime.CallDebugger["Damaged file (lost space). Proceed to ignore."L]; LOOP END];
    IF temporary THEN
      IF immutable THEN DeleteImmutable[file, volume] ELSE Delete[file]
    ENDLOOP
  END;

FileHelperProcess: PROCEDURE =
  BEGIN
  helpCount: CARDINAL ← 0;
  fileD: FileInternal.Descriptor;
  req: FilePageTransfer.Request;
  countInitiated: Environment.PageCount;
  Helper1: FileAccessProc--vol, filePtr, op, newSizeP-- =
    BEGIN
    success: BOOLEAN;
    group: FileInternal.PageGroup;
    IF ~FileCache.GetFilePtrs[0, filePtr.fileID].success THEN RETURN; -- fell out of cache
    [success, group] ← VolFileMap.GetPageGroup[vol, filePtr, req.filePage];
    IF success AND req.filePage<group.nextFilePage THEN
      FileCache.SetPageGroup[req.file.fID, group, FALSE] -- page is within file?
    ELSE
      Runtime.CallDebugger["Mapped off File (Helper)"L]
    END;

  DO
    req ← FilerException.Await[];
    helpCount ← helpCount + 1;
    fileD ← GetFileDescriptor[@req.file]; -- file cache updated as side effect
    WITH fileD SELECT FROM
      remote => NULL;
      local => LogicalVolume.FileVolumeAccess[Helper1, LOOPHOLE[@fileD], LogicalVolume.noOp, NIL];
      ENDCASE;
    countInitiated ← FilePageTransfer.Initiate[req];
    IF req.promise THEN MStore.Promise[countInitiated]
    ENDLOOP
```

```
    END;

GetAttributes: PUBLIC PROCEDURE [file: Capability] RETURNS [type: Type, immutable, temporary: BOOLEAN, volume:
      System.VolumeID] =
    BEGIN
    fileD: FileInternal.Descriptor = GetFileDescriptor[@file];
    WITH fileD SELECT FROM
        local => RETURN[type, immutable, temporary, fileD.volumeID]; -- caveat: on scoping
        ENDCASE => RETURN[File.Type[0], FALSE, FALSE, VolumeInternal.altoVolume]
    END;

GetBootLocation: PUBLIC PROCEDURE [file: File.Capability] RETURNS [device: KernelFile.DeviceHandle, link: SpecialFile.Link] =
    BEGIN    -- What if systemID changes while this runs?
    vID: Volume.ID;
    group: FileInternal.PageGroup;
    channel: DiskChannel.Handle;

    GetVIDAndGroup[@vID, @group, @file, 0]; -- set vID, group
    [channel, LOOPHOLE[link, DiskChannel.Address]] ← SubVolume.GetPageAddress[vID, group.volumePage];
    device ← [DiskChannel.GetAttributes[channel].drive]
    END;


-- Look up file in cache and then in all vfm's, return a descriptor or signal error
GetFileDescriptor: PROCEDURE [file: POINTER TO READONLY File.Capability] RETURNS [fileD: FileInternal.Descriptor] =
    BEGIN
    success: BOOLEAN;
    fD: POINTER TO FileInternal.Descriptor;
    localFileD: local FileInternal.Descriptor ← [file.fID, Volume.nullID, local[,,,]];

    IF file.fID = FilePageLabel.nullID THEN ERROR Unknown[file↑];
    IF LOOPHOLE[file.fID, SystemInternal.UniversalID].series = SystemInternal.altoFPSeries THEN
        IF GetAltoSize[file] = 0 THEN ERROR Unknown[file↑] ELSE RETURN[[file.fID, VolumeInternal.altoVolume, remote[]]];
    [success, fD] ← FileCache.GetFilePtrs[1, file.fID];
    IF success THEN BEGIN fileD ← fD↑; FileCache.ReturnFilePtrs[1, fD]; RETURN END;
    DO
        IF (localFileD.volumeID ← Volume.GetNext[localFileD.volumeID]) = Volume.nullID THEN ERROR Unknown[file↑];
        LogicalVolume.FileVolumeAccess[GetFileDescriptor1, @localFileD, LogicalVolume.noOp, NIL];
        IF localFileD.type # PilotFileTypes.tFreePage THEN RETURN[localFileD]
        ENDLOOP
    END;


-- Look up file in vfm; fill in & cache the descriptor (volumeID←null if not found)
-- Called from GetFileDescriptor and from Delete Imutable
GetFileDescriptor1: FileAccessProc --vol, filePtr, op, newSizeP-- =
    BEGIN
    success: BOOLEAN;
    group: FileInternal.PageGroup;
    label: FilePageLabel.Label;

    [success, group] ← VolFileMap.GetPageGroup[vol, filePtr, 0];
    IF ~success THEN BEGIN filePtr.type ← PilotFileTypes.tFreePage; RETURN END;
    label ← LabelTransfer.ReadLabel[filePtr↑, 0, group.volumePage];
    IF label.fileID~ = filePtr.fileID OR FilePageLabel.GetFilePage[@label]~ = 0 THEN SIGNAL LabelError;
    filePtr.body ← local[immutable: label.immutable, temporary: label.temporary, size: VolFileMap.GetPageGroup[vol, filePtr,
      File.lastPageNumber].group.filePage --success?--, type: FilePageLabel.GetType[@label]];
    FileCache.SetFile[filePtr↑, FALSE];
    END;

GetFilePoint: PUBLIC PROCEDURE [pEntry: LONG POINTER TO VMMapLog.Entry, pFile: POINTER TO File.Capability, filePage:
      File.PageNumber] =
    BEGIN
    countMax: CARDINAL = 4096;
    WITH uID: LOOPHOLE[pFile.fID, SystemInternal.UniversalID] SELECT
            (IF uID.series = SystemInternal.altoFPSeries THEN altoFP ELSE ois) FROM
        altoFP =>
            pEntry↑ ← VMMapLog.Entry[, countMax, , , alto31[LOOPHOLE[uID.fp], Inline.LowHalf[filePage]]];
        ois =>
            BEGIN
            vID: Volume.ID;
            group: FileInternal.PageGroup;
            lvPage: LogicalVolume.PageNumber;
            success: BOOLEAN;
            svH: SubVolume.Handle;
```

```
        pvPage: PhysicalVolume.PageNumber;
        count: Environment.PageCount;
        drive: DiskChannel.Drive;
        label: FilePageLabel.Label;

        GetVIDAndGroup[@vID, @group, pFile, filePage];
        lvPage ← filePage-group.filePage + group.volumePage; -- map file page to logical vol page
        [success, svH] ← SubVolume.Find[vID, lvPage]; -- assumes vol won't disappear !!
        IF ~success THEN ERROR; -- we must have a Pilot 31!
        pvPage ← lvPage-svH.lvPage + svH.pvPage; -- logical vol page to physical vol page
        count ← MIN[countMax, Inline.LowHalf[group.nextFilePage-filePage]];
        drive ← DiskChannel.GetAttributes[svH.channel].drive;
        label.fileID ← pFile.fID; FilePageLabel.SetFilePage[@label, filePage];
        IF PilotSwitches.switches.A = down THEN -- CoPilot 0
            IF drive = DiskChannel.Drive[1] --DP1-- THEN
                pEntry↑ ← [, count, , , pilot31[vda: Inline.LowHalf[pvPage], fid: VMMapLog.PilotFID[uID.a, uID.b], filePage:
                  Inline.LowHalf[filePage]]]
            ELSE
                pEntry↑ ← [, count, , , nil[]]
        ELSE
            pEntry↑ ← [, count, , , disk[driveTag: DiskChannel.GetDriveTag[drive], diskPage: pvPage, labelCheck:
              FilePageLabel.LabelCheckSum[@label, 0]]];
        END;
    ENDCASE;
END;

GetNextFile: PUBLIC PROCEDURE [volume: Volume.ID, file: File.Capability] RETURNS [nextFile: File.Capability] =
    BEGIN
    fileD: local FileInternal.Descriptor ← [file.fID, volume, local[,,,]];

    LogicalVolume.FileVolumeAccess[VolFileMap.GetNextFileProc, @fileD, LogicalVolume.noOp, NIL];
    RETURN [File.Capability[fileD.fileID, IF fileD.fileID = FilePageLabel.nullID THEN 0 ELSE FileInternal.maxPermissions]]
    END;

GetSize: PUBLIC PROCEDURE [file: Capability] RETURNS [size: PageCount] =
    BEGIN
    fileD: FileInternal.Descriptor = GetFileDescriptor[@file];
    RETURN[WITH fileD SELECT FROM
        local => size,
        ENDCASE => GetAltoSize[@file]]
    END;

-- Look up file in cache and then in all vfm's, set vID and page group descriptor or signal error
GetVIDAndGroup: PROCEDURE [pVID: POINTER TO Volume.ID, pGroup: POINTER TO FileInternal.PageGroup, pFile: POINTER TO
      File.Capability, filePage: File.PageNumber] =
    BEGIN
    GetVIDAndGroup1: FileAccessProc --vol, filePtr, op, newSizeP-- =
        BEGIN
        success: BOOLEAN;
        pVID↑ ← filePtr.volumeID;
        [success, pGroup↑] ← FileCache.GetPageGroup[filePtr.fileID, filePage];
        IF ~success THEN [success, pGroup↑] ← VolFileMap.GetPageGroup[vol, filePtr, filePage];
        IF ~(success AND filePage<pGroup.nextFilePage) THEN ERROR;
        END;
    FileChange[GetVIDAndGroup1, pFile, LogicalVolume.noOp, NIL];
    END;

Grow1: FileAccessProc --vol, filePtr, op, newSizeP-- =
    BEGIN OPEN filePtr;
    -- GetPageGroup might watch for unsuccess but doesn't
    group: FileInternal.PageGroup ← IF op = LogicalVolume.create THEN [0,LogicalVolume.nullVolumePage,0] ELSE
            VolFileMap.GetPageGroup[vol, filePtr, size-MIN[size, 1]].group;

    IF newSizeP = NIL THEN ERROR;
    WHILE size<newSizeP↑ OR newSizeP↑ = 0 AND op = LogicalVolume.create DO
        group ← FileInternal.PageGroup[group.nextFilePage, group.volumePage + group.nextFilePage-group.filePage,
                newSizeP↑];
        VolAllocMap.AllocPageGroup[vol, filePtr, @group, op = LogicalVolume.create]; -- results in modified group and size
        VolFileMap.InsertPageGroup[vol, filePtr, @group];
        IF size = newSizeP↑ THEN EXIT
        ENDLOOP
    END;
```

```
IsOnVolume: PUBLIC PROCEDURE [file: Capability, volume: System.VolumeID] =
    BEGIN
    fD: FileInternal.Descriptor = FileInternal.Descriptor[file.fID, volume, remote[]];
    [] ← GetFileDescriptor[@file ! Unknown => BEGIN FileCache.SetFile[fD, FALSE]; CONTINUE; END]
    END;

MakeImmutable: PUBLIC PROCEDURE [file: Capability] =
    BEGIN
    MakeImmutable1: FileAccessProc --vol, filePtr, op, newSizeP-- =
        BEGIN
        IF NOT LabelTransfer.VerifyLabels[filePtr↑, FileInternal.PageGroup[0, volumePage, 1]] THEN SIGNAL LabelError;
        filePtr.immutable ← TRUE;
        ForceOutAttributes[filePtr];
        END;
    FileChange[MakeImmutable1, @file, File.write, NIL]
    END;

MakePermanent: PUBLIC PROCEDURE [file: Capability] =
    -- Notices files of Pilot root file type
    BEGIN
    MakePermanent1: FileAccessProc --vol, filePtr, op, newSizeP-- =
        BEGIN
        IF NOT LabelTransfer.VerifyLabels[filePtr↑, FileInternal.PageGroup[0, volumePage, 1]] THEN SIGNAL LabelError;
        -- setting the root file in the array must be done first to be consistant with the Pilot Philosophy
        IF filePtr.type IN PilotFileTypes.PilotRootFileType THEN LogicalVolume.PutRootFile[vol, filePtr.type, @file];
        filePtr.temporary ← FALSE;
        ForceOutAttributes[filePtr];
        END;
    FileChange[MakePermanent1, @file, File.write, NIL];
    END;

MakeUnBootable: PUBLIC PROCEDURE [file: File.Capability] =
    BEGIN SIGNAL SystemInternal.Unimplemented END;

PermissionsAND: PROCEDURE [greaterPermissions, maxPermissions: Permissions]
        RETURNS [lesserPermissions: Permissions] = LOOPHOLE[Inline.BITAND];

Pin: PUBLIC PROCEDURE [file: File.Capability] =
    BEGIN
    Pin1: FileAccessProc--vol, filePtr, op, newSizeP-- =
        BEGIN
        success: BOOLEAN;
        group: FileInternal.PageGroup ← [0,0,0];
        FileCache.SetFile[filePtr↑, TRUE];
        WHILE group.nextFilePage < filePtr.size DO
            [success, group] ← VolFileMap.GetPageGroup[vol, filePtr, group.nextFilePage];
            IF ~success THEN ERROR;
            FileCache.SetPageGroup[filePtr.fileID, group, TRUE]
            ENDLOOP
        END;
    FileChange[Pin1, @file, LogicalVolume.noOp, NIL]
    END;

SetDebuggerFiles: PUBLIC PROCEDURE [debugger, debuggee: File.Capability] =
    BEGIN -- what if systemID changes while this runs?
    deviceEr, deviceEe: KernelFile.DeviceHandle;
    linkEr, linkEe: SpecialFile.Link;
    SetDebuggerFiles1: PROCEDURE [volume: LogicalVolume.Handle] =
        BEGIN OPEN bootFileIDs: LOOPHOLE[@volume.bootingInfo, LONG POINTER TO StoragePrograms.BootFileIDs];
        bootFileIDs.debugger.fID ← debugger.fID;
        bootFileIDs.debugger.da ← LOOPHOLE[linkEr];
        bootFileIDs.debuggee.fID ← debuggee.fID;
        bootFileIDs.debuggee.da ← LOOPHOLE[linkEe]
        END;
    [deviceEr, linkEr] ← GetBootLocation[debugger]; [deviceEe, linkEe] ← GetBootLocation[debuggee];
    IF deviceEr~ = deviceEe OR GetFileDescriptor[@debugger].fileD.volumeID~ = LogicalVolume.systemID THEN
        ERROR InvalidParameters;
    LogicalVolume.VolumeAccess[pVID: @LogicalVolume.systemID, proc: SetDebuggerFiles1, forceOut: TRUE]
    END;
```

```
SetSize: PUBLIC PROCEDURE [file: Capability, size: PageCount] = --(called by Delete)
    BEGIN
    oldSize: PageCount = GetSize[file];
    IF size < oldSize THEN FileChange[Shrink1, @file, File.shrink + File.write, @size];
    IF size > oldSize THEN FileChange[Grow1, @file, File.grow + File.write, @size]
    END;

-- Nobody calls????
--SetType: PROCEDURE [file: Capability, newType: Type] =
    --BEGIN
    --SetType1: FileAccessProc + + vol, filePtr, op, newSizeP + + =
        --BEGIN filePtr.type ← newType END;
    --FileChange[SetType1, @file, File.read, NIL]
    --END;

Shrink1: FileAccessProc--vol, filePtr, op, newSizeP-- =
    BEGIN OPEN filePtr;
    group: FileInternal.PageGroup;

    IF newSizeP = NIL THEN ERROR;
    WHILE size>newSizeP↑ OR newSizeP↑ = 0 AND op = File.delete DO
        group ← FileInternal.PageGroup[newSizeP↑, LogicalVolume.nullVolumePage, size];
        VolFileMap.DeletePageGroup[vol, filePtr, @group]; -- results in modified group
        IF size = newSizeP↑ THEN EXIT; -- extra call to DeletePageGroup to delete attributes
        VolAllocMap.FreePageGroup[vol, filePtr, @group, op = File.delete] -- filePtr.size updated
        ENDLOOP
    END;

-- ENTRY (or at least Pseudo-entry) PROCEDURES

-- Monitor protects validity of fileD
DeleteImmutable: PUBLIC ENTRY PROCEDURE [file: Capability, volume: System.VolumeID] =
    BEGIN
    fileD: local FileInternal.Descriptor ← [file.fID, volume, local[,,,]];
    success: BOOLEAN;
    fD: FileInternal.FilePtr;
    zero: File.PageCount ← 0;

    IF PermissionsAND[file.permissions, File.delete] = 0 THEN RETURN WITH ERROR Error[insufficientPermissions];
    [success, fD] ← FileCache.GetFilePtrs[1, file.fID];
    IF success THEN
        BEGIN
        IF (success ← fD.volumeID = volume) THEN
            WITH f: fD↑ SELECT FROM
                local => fileD ← f;
                ENDCASE => ERROR;
        FileCache.ReturnFilePtrs[1, fD];
        END;
    IF ~success THEN LogicalVolume.FileVolumeAccess[GetFileDescriptor1, @fileD, LogicalVolume.noOp, NIL];
    IF fileD.type = PilotFileTypes.tFreePage THEN RETURN WITH ERROR Unknown[file];
    IF ~fileD.immutable THEN RETURN WITH ERROR Error[notImmutable];
    LogicalVolume.FileVolumeAccess[Shrink1, @fileD, File.delete, @zero]
    END;

-- Check permissions, call FileVolumeAccess to make a change
-- Prepare for writing label of page zero (Map & Unmap may cause helper to call FileVolumeAccess)
-- Monitor protects pageBuffer, volumePage and serializes file changes
FileChange: ENTRY PROCEDURE [proc: FileAccessProc, fileP: POINTER TO READONLY Capability, op: LogicalVolume.Operations,
      newSizeP: POINTER TO READONLY PageCount] =
    BEGIN
    zeroSize, success: BOOLEAN;
    group: FileInternal.PageGroup;
    GetGroup0: FileAccessProc --vol, filePtr, op, newSizeP-- =
        BEGIN -- gets the zeroth page group if it should get bumped out of cache after Map
        [success, group] ← VolFileMap.GetPageGroup[vol, filePtr, 0]
        END;
    fileD: FileInternal.Descriptor ← GetFileDescriptor[fileP ! Unknown => GOTO unknown];

    WITH fileD SELECT FROM
        local => BEGIN zeroSize ← size = 0; IF immutable THEN RETURN WITH ERROR Error[immutable]; END;
        ENDCASE => RETURN WITH ERROR Error[insufficientPermissions]; -- remote server not implemented
    IF op = File.read THEN ERROR;                                   --update all labels not implemented yet
    IF op = File.write THEN
```

```
        BEGIN
        IF ~zeroSize THEN SimpleSpace.Map[pageBuffer, Space.WindowOrigin[fileP↑, 0], TRUE];
        [success, group] ← FileCache.GetPageGroup[fileP.fID, 0];
        IF ~success THEN LogicalVolume.FileVolumeAccess[GetGroup0, LOOPHOLE[@fileD], LogicalVolume.noOp, NIL];
        volumePage ← group.volumePage
        END
    ELSE IF PermissionsAND[fileP.permissions, op] # op THEN RETURN WITH ERROR Error[insufficientPermissions];
    LogicalVolume.FileVolumeAccess[proc, LOOPHOLE[@fileD], op, newSizeP !
            Volume.InsufficientSpace => GOTO insufficientSpace;
            Volume.Unknown => ERROR --won't happen:--];
    IF op # File.delete THEN FileCache.SetFile[fileD, FALSE]; -- could restore group too
    IF op = File.write AND ~zeroSize THEN BEGIN FileCache.SetPageGroup[fileP.fID, group, FALSE]; SimpleSpace.Unmap[pageBuffer]
            END
    EXITS
        unknown => RETURN WITH ERROR Unknown[fileP↑];
        insufficientSpace => RETURN WITH ERROR Volume.InsufficientSpace
    END;

-- Access file attributes in label given fileID keeping 2 element MRU cache
GetAltoSize: ENTRY PROCEDURE [file: POINTER TO READONLY File.Capability] RETURNS[size: PageCount] =
    BEGIN
    faP: LONG POINTER TO AltoFileDefs.FA = @LOOPHOLE[bufferPointer, LONG POINTER TO AltoFileDefs.LD].eofFA;
    IF file.fID = cacheA.file THEN RETURN[cacheA.size];
    IF file.fID = cacheB.file THEN size ← cacheB.size ELSE
        BEGIN
        SimpleSpace.Map[pageBuffer, Space.WindowOrigin[file↑, 0], FALSE];
        size ← faP.page + MIN[faP.byte, 1];
        SimpleSpace.Unmap[pageBuffer]
        END;
    cacheB ← cacheA;
    cacheA ← [file.fID, size]
    END;

MakeBootable: PUBLIC ENTRY PROCEDURE [file: File.Capability, firstPage: File.PageNumber, count: File.PageCount, lastLink:
      SpecialFile.Link] RETURNS [firstLink: SpecialFile.Link] =
    BEGIN OPEN
            lastLink: LOOPHOLE[lastLink, DiskChannel.Address],
            firstLink: LOOPHOLE[firstLink, DiskChannel.Address];
    fileD: FileInternal.Descriptor ← GetFileDescriptor[@file];
    limitPage: File.PageNumber = firstPage + count;
    group, nextGroup: FileInternal.PageGroup;
    lastGroupLink: DiskChannel.Address;

    InitGroups: PROCEDURE [volume: LogicalVolume.Handle] =
        BEGIN
        group ← VolFileMap.GetPageGroup[volume, @fileD, firstPage].group; -- what if ~success?
        END;

    GetNewGroups: PROCEDURE [volume: LogicalVolume.Handle] =
        BEGIN OPEN group;

        IF filePage <= firstPage THEN -- chain begins in this page group: trim group and set outer result value
            BEGIN
            volumePage ← volumePage + (firstPage-filePage);
            filePage ← firstPage;
            firstLink ← SubVolume.GetPageAddress[fileD.volumeID, volumePage].address
            END;

        IF nextFilePage < limitPage THEN -- chain extends beyond this group
            BEGIN -- chain last page of this group to first page of next group
            nextGroup ← VolFileMap.GetPageGroup[volume, @fileD, nextFilePage].group; -- what if ~success?
            lastGroupLink ← SubVolume.GetPageAddress[fileD.volumeID, nextGroup.volumePage].address
            END
        ELSE -- chain ends in this page group: trim group and use caller-supplied link
            BEGIN
            nextGroup.filePage ← limitPage; -- to stop loop after this iteration
            lastGroupLink ← lastLink;
            nextFilePage ← limitPage
            END;
        END;

    TransferLabels: PROCEDURE = --INLINE, when compiler bug is fixed....
        BEGIN OPEN group;
```

```
        countMapped: Environment.PageCount;
        groupLink: DiskChannel.Address;
        WHILE filePage<nextFilePage DO -- rewrite labels in batches via longBuffer
            countMapped ← Inline.LowHalf[MIN[nextFilePage-filePage, longBufferSize]];
            groupLink ← IF filePage + countMapped = nextFilePage THEN lastGroupLink ELSE
                    SubVolume.GetPageAddress[fileD.volumeID, volumePage + countMapped].address;
            SimpleSpace.Map[longBuffer, Space.WindowOrigin[file, filePage], TRUE, countMapped];
            LabelTransfer.WriteLabelsAndData[fileD, [filePage, volumePage, filePage + countMapped], longBufferPage,
                    chained, groupLink];
            SimpleSpace.Unmap[longBuffer];
            filePage ← filePage + countMapped;
            volumePage ← volumePage + countMapped
            ENDLOOP;
        END;

    IF (WITH fD: fileD SELECT FROM
        local => firstPage + count>fD.size OR fD.type~ = PilotFileTypes.tBootFile,
        ENDCASE => TRUE) THEN ERROR InvalidParameters;
    LogicalVolume.VolumeAccess[@fileD.volumeID, InitGroups, FALSE];
    WHILE group.filePage<limitPage DO
        LogicalVolume.VolumeAccess[@fileD.volumeID, GetNewGroups, FALSE];
        TransferLabels[];
        group ← nextGroup;
        ENDLOOP;
    END;

-- Move with the protection of an intention page (not fully implemented yet)
-- Needs to be protected by a monitor
-- THIS CANNOT WORK!!!!!!!!
Move: PUBLIC PROCEDURE [file: Capability, volume: System.VolumeID] =
    BEGIN
    Move1: FileAccessProc =
        BEGIN
        filePtr↑ ← oldFileD
        END;
    space: Space.Handle = Space.Create[spaceSize, Space.virtualMemory, Space.defaultBase];
    base: File.PageNumber;
    newFile: File.Capability;
    fileD: FileInternal.Descriptor = GetFileDescriptor[@file];
    oldFileD: local FileInternal.Descriptor;

    IF fileD.volumeID = volume THEN RETURN;
    WITH newFileD: fileD SELECT FROM
        local => oldFileD ← newFileD;
        ENDCASE => ERROR Error[insufficientPermissions];
    IF oldFileD.immutable THEN ERROR Error[immutable];
    newFile ← Create[volume, oldFileD.size, PilotFileTypes.tBeingMoved];
    -- Should create an intentions page
    FOR base ← 0, base+spaceSize WHILE base<oldFileD.size DO      -- end condition? overshoot?
        Space.Map[space, Space.WindowOrigin[file, base]];
        Space.Remap[space, Space.WindowOrigin[newFile, base]];
        Space.Unmap[space]
        ENDLOOP;
    Space.Delete[space];
    -- Commit (on intentions page)
    Delete[file];
    -- Note progress (on intentions page)
    oldFileD.volumeID ← volume;
    FileChange[Move1, @newFile, File.read, NIL];          -- change id, type, temp
    -- Delete the intentions page
    END;

-- Similar to Move but leave original intact (could be more efficient)
-- Needs to be protected by a monitor
```

```
-- THIS CANNOT WORK!!!!!!!!
ReplicateImmutable: PUBLIC PROCEDURE [file: Capability, volume: System.VolumeID] =
    BEGIN
    Replicate1: FileAccessProc =
        BEGIN
        filePtr↑ ← oldFileD
        END;
    space: Space.Handle = Space.Create[spaceSize, Space.virtualMemory, Space.defaultBase];
    base: File.PageNumber;
    newFile: File.Capability;
    fileD: FileInternal.Descriptor = GetFileDescriptor[@file];
    oldFileD: local FileInternal.Descriptor;

    IF fileD.volumeID = volume THEN RETURN;
    WITH newFileD: fileD SELECT FROM
        local => oldFileD ← newFileD;
        ENDCASE => ERROR Error[insufficientPermissions];
    IF ~oldFileD.immutable THEN ERROR Error[notImmutable];
    newFile ← Create[volume, oldFileD.size, PilotFileTypes.tBeingReplicated];
    FOR base ← 0, base + spaceSize WHILE base<oldFileD.size DO      -- end condition? overshoot?
        Space.Map[space, Space.WindowOrigin[file, base]];
        Space.Remap[space, Space.WindowOrigin[newFile, base]];
        Space.Unmap[space]
        ENDLOOP;
    Space.Delete[space];
    oldFileD.volumeID ← volume;
    FileChange[Replicate1, @newFile, File.read, NIL] -- change id, type, temp
    END;

-- INTERNAL PROCEDURES

-- Can be called only from within FileChange (esp. by FileAccessProcs)
-- Forces out attributes in file descriptor to label of file page 0
ForceOutAttributes: --INTERNAL-- PROCEDURE [filePtr: POINTER TO local FileInternal.Descriptor] =
    BEGIN
    IF filePtr.size = 0 THEN -- no data or boot-chain in file; simply write label and zero data
        LabelTransfer.WriteLabels[filePtr↑, FileInternal.PageGroup[0, volumePage, 1]]
    ELSE -- write label and data of page zero, taking care to preserve boot-chain link (if any)
        BEGIN
        labelType: LabelTransfer.LabelType ← normal;
        link: LONG CARDINAL ← 0;
        IF filePtr.type = PilotFileTypes.tBootFile THEN
            BEGIN
            labelType ← chained;
            link ← LabelTransfer.ReadLabel[filePtr↑, 0, volumePage].bootChainLink;
            END;
        LabelTransfer.WriteLabelsAndData[filePtr↑, [0, volumePage, 1], SimpleSpace.Page[pageBuffer], labelType,
                LOOPHOLE[link]];
        END;
    END;


-- Initialization

IF PilotSwitches.switches.F = down THEN Runtime.CallDebugger["Key Stop F"L];

IF NOT FilePageLabel.tBootFile = PilotFileTypes.tBootFile THEN ERROR; -- static assertion

Process.Detach[FORK FileHelperProcess[]];

START FMPrograms.VolAllocMapImpl;
START FMPrograms.VolFileMapImpl;
[debugClass, debuggerDevice] ← START FMPrograms.VolumeImpl[bootFile, pDebuggerIDs];
RETURN

END.
```

LOG

Time: May 22, 1978 1:05 PM By: Purcell Action: Created file from file.mesa

Time: June 21, 1978 7:35 PM By: Redell Action: Added START HelperImpl
Time: June 22, 1978 1:49 AM By: Purcell Action: fix to setSize, remove testing
Time: September 4, 1978 10:42 PM By: Purcell Action: reorganize with interlocks
Time: September 23, 1978 4:05 PM By: Horsley Action: remove reference to FilePoint
Time: September 25, 1978 11:50 PM By: Purcell Action: simple space temporarily in mds
Time: September 26, 1978 2:37 PM By: Purcell Action: CR: 20.23 & 20.24; resolve errors with monitors
Time: September 28, 1978 11:20 AM By: Purcell Action: CR: 20.33 & 20.34 & 20.39; more error checks (double immutable, shrink + write) (plus independent temp and immut);
Time: September 28, 1978 2:54 PM By: Purcell Action: CR 20.35 improve serialization
Time: September 28, 1978 9:42 PM By: Purcell Action: CR: 20.35; delete bug fixed
Time: September 29, 1978 2:37 PM By: Purcell Action: CR: 20.44; fix to GetFilePoint
Time: October 3, 1978 4:57 PM By: Purcell CR 20.44: SpecialFile.Pin fix for Remap to PilotVolume
Time: October 19, 1978 10:39 AM By: Purcell Action: CR 20.103: FileTypes directly
Time: March 19, 1979 5:46 PM By: Redell Action: Converted to Mesa 5.0 and new DiskChannel interface; removed FileType constants; general cleanup
Time: May 1, 1979 2:46 PM By: Redell Action: Fixed bug in GetFilePoint: wasn't converting logical volume page numbers to physical volume page numbers
Time: May 1, 1979 4:52 PM By: McJones Action: DeleteTemp coughed on immutable
Time: July 31, 1979 6:19 PM By: Forrest Action: Make export new Storage program and import new FMPrograms
Time: August 10, 1979 3:47 PM By: Redell Action: Converted to FilePageLabel; added MakeBootable and other stuff for bootchains
Time: August 16, 1979 2:42 PM By: Redell Action: Added GetBootLocation, SetDebuggerFiles, new map log variant
*Time: August 31, 1979 6:07 PM By: Forrest Action: Alphabetized according to monitor conventions; replaces passing some objects by pointers to objects; made use new logical volume, new Vol*Map. Added a few comments*
Time: September 4, 1979 4:27 PM By: Forrest Action: Re-organized MakeBootable to avoid deadlock
Time: September 6, 1979 2:49 PM By: McJones AR1593: Helper must Promise countInitiated to Swapper

```
DIRECTORY
    BootChannel: FROM "BootChannel" USING [Location],
    StoragePrograms: FROM "StoragePrograms" USING [BootFileIDs, DebugClass];

FMPrograms: DEFINITIONS =

BEGIN

--FileImpl: PROGRAM; now part of StoragePrograms???

FilePointImpl: PROGRAM;

HelperImpl: PROGRAM;

VolAllocMapImpl: PROGRAM;

VolFileMapImpl: PROGRAM;

VolumeImpl: PROGRAM [bootFile: POINTER TO disk BootChannel.Location, pDebuggerIDs: POINTER TO
        StoragePrograms.BootFileIDs] RETURNS [debugClass: StoragePrograms.DebugClass, debuggerDevice: UNSPECIFIED];
        Basically Copy of StoragePrograms.FileImpl

END.
```

LOG

Time: June 29, 1978 2:34 PM By: Purcell Action: Created file from initializeFM
Time: August 5, 1978 11:17 PM By: Redell Action: Replaced PageBufferImpl with FilePointImpl.
Time: July 31, 1979 5:31 PM By: Forrest Action: Commented out FileImpl; changed VolumeImpl.

```
DIRECTORY
    File: FROM "File" USING [Capability, ID, lastPageNumber, nullCapability, PageCount, PageNumber, Type],
    FileInternal: FROM "FileInternal" USING [Descriptor, PageGroup],
    System: FROM "System" USING [UniversalID],
    Volume: FROM "Volume" USING [ID, nullID, PageCount],
    VolumeInternal: FROM "VolumeInternal" USING [PageNumber];

LogicalVolume: DEFINITIONS
    SHARES File, System =
BEGIN

--Open: PROCEDURE[drive: CARDINAL, initFlag: BOOLEAN, rebuildFlag: BOOLEAN];--    --SpecialFile.VolumeOpen
--Close: PROCEDURE[volume: Volume.ID];--    --Volume.Close

FileAccessProc: TYPE = PROCEDURE [filePtr: POINTER TO local FileInternal.Descriptor, op: Operations, newSize:
        File.PageCount];


FileVolumeAccess: PROCEDURE [proc: FileAccessProc, filePtr: POINTER TO local FileInternal.Descriptor, op: Operations,
        newSize: File.PageCount];
    -- gives proc exclusive access to volume


PutRootFile: PROCEDURE [type: File.Type --LogicalVolume.RootFileType--, file: File.Capability, volume: Volume.ID];
    -- store file name in root page; a side effect of makePermanent

systemID: Volume.ID; -- purposefully not READONLY, so can be passed by reference

VolumeAccess: PROCEDURE [pVID: POINTER TO Volume.ID, proc: PROCEDURE [volume: Handle], forceOut: BOOLEAN];


-- Op: TYPE = MACHINE DEPENDENT RECORD[ + + can't this junk go away??
    -- fill: [0..256),
    -- setFileAttributes,
    -- setVolumeAttributes,
    -- create,
    -- delete,
    -- shrink,
    -- grow,
    -- write,
    -- read: BOOLEAN];

-- NOTE: LogicalVolume.Operations MUST be coordinated with File.Permissions
-- (This should be cleaned up somehow to eliminate parallel definitions...)

Operations: TYPE = [0..256);

noOp: Operations = 0;
-- File.Permissions slip in here
create: Operations = 32;
setVolumeAttributes: Operations = 64;
setFileAttributes: Operations = 128;

PageNumber: TYPE = VolumeInternal.PageNumber;

GroupPtr: TYPE = POINTER TO FileInternal.PageGroup;

Handle: TYPE = LONG POINTER TO Descriptor;

seal, Seal: CARDINAL = 131313B; -- word zero of valid logical volume root page
currentVersion: CARDINAL = 2; -- increment each time Descriptor below is reformatted

Descriptor: TYPE = MACHINE DEPENDENT RECORD[
        --**~~~~~~~~~~~~~~~~~~~~~~~~~ Do not reorder the beggining of this record ~~~~~~~~~~~~~~~~~~~~~~~~~**
        --**--  seal: CARDINAL ← Seal,                    -- absolutly must be 1st field--    --**
        --**--  version: CARDINAL ← currentVersion,       -- must be 2nd field--              --**
        --**--  vID: Volume.ID,                                                               --**
        --**--  open, changing: BOOLEAN,                                                      --**
        --**--  treeLevel: TreeLevel,                                                         --**
        --**--  labelLength: CARDINAL[0..maxLogicalVolumeLabelLength),                        --**
        --**--  type: Type,                                                                   --**
        --**--  pad: [0..7B] ← 0,                                                             --**
        --**--  volumeSize: Volume.PageCount,                                                 --**
        --**--  bootingInfo: ARRAY [0..48) OF WORD,                                           --**
        --**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Reorder from here on only~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ **
        freePageCount: Volume.PageCount,
```

```
        upperBound: Volume.PageCount,
        vamStart: PageNumber,
        vfmStart: PageNumber,
        rootFileID: ARRAY [0..15] OF File.ID,    -- Coordinate Changes with FileTypes.PilotRootFileTYpe
        vam: File.ID,
        vfm: File.ID,
        free: File.ID,
        clientRootFile: File.Capability,
        interval: Intervals,
        label: PACKED ARRAY [0..maxLogicalVolumeLabelLength) OF CHARACTER];

rootPageNumber: PageNumber = 0;

Type: TYPE = {client, debugger, debuggerDebugger, diagnostic};

nullDescriptor: Descriptor = [
        --seal: seal,
        --version: currentVersion,
        vID: Volume.nullID,
        open: TRUE, changing: TRUE,
        treeLevel: LAST[TreeLevel],        -- can be reduced by taking volume size into account
        labelLength: 0,
        --pad: 0,
        type: client,
        volumeSize: 0,
        bootingInfo: nullBoot,
        freePageCount: 0,
        upperBound: 0,
        vamStart: 1,
        vfmStart: 2 -- + (DiskChannel.pagesPerDisk-1)/(Environment.bitsPerWord*Environment.wordsPerPage)--,
        rootFileID: nullRootFileIDs,
        vfm: nullID,
        vam: nullID,
        free: nullID,
        clientRootFile: File.nullCapability,
        interval: nullIntervals,
        label: nullName];  --client root

nullID: File.ID = File.nullCapability.fID;

TreeLevel: TYPE = Level;        -- has two names. TreeLevel and Level
Level: TYPE = [0..5];  -- six levels of worst case branching of 2^4 (= 16) reaches 2^24 pages

Interval: TYPE = RECORD [
    key: Key,
    volumePage: PageNumber,
    nextKey: Key];

Intervals: TYPE = ARRAY Level OF Interval;      --lives in volRootPage

Key: TYPE = RECORD [
    fileID: File.ID,
    filePage: File.PageNumber];


-- constants
maxLogicalVolumeLabelLength: CARDINAL = 40;
maxID: File.ID = LOOPHOLE[System.UniversalID[177777B,177777B,177777B,177777B], File.ID];
maxKey: Key = Key[maxID, File.lastPageNumber];
nullKey: Key = Key[nullID, 0];
nullVolumePage: PageNumber = 0;
nullInterval: Interval = Interval[nullKey, nullVolumePage, nullKey];
nullIntervals: ARRAY Level OF Interval = [nullInterval, nullInterval, nullInterval, nullInterval, nullInterval];
nullName: PACKED ARRAY [0..maxLogicalVolumeLabelLength) OF CHARACTER = [0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C,
        0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C, 0C,
        0C, 0C, 0C];
nullRootFileIDs: ARRAY [0..LENGTH[nullDescriptor.rootFileID]-1] OF File.ID = [nullID, nullID, nullID, nullID, nullID, nullID, nullID,
        nullID, nullID, nullID, nullID, nullID, nullID, nullID, nullID, nullID];

nullBoot: ARRAY [0..48) OF UNSPECIFIED = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];

END.
```

LOG

Time: June 1, 1978 2:18 PM By: Redell Action: Created file
Time: June 5, 1978 4:28 PM By: Purcell Action: added descriptor, and vfm types
Time: August 14, 1978 11:32 AM By: Purcell Action: removed dependency on rigiddisk
Time: August 24, 1978 10:05 PM By: Purcell Action: more opens and closes
Time: October 19, 1978 10:30 AM By: Purcell Action: remove FileType dependency
Time: March 11, 1979 10:57 AM By: Redell Action: Changed bogus out-of-range File.Permissions to LogicalVolume.Operations; removed absolute definitions of assorted file types(!).
Time: March 18, 1979 2:03 PM By: Redell Action: Reformatted descriptor and made machine-dependent.
Time: July 18, 1979 4:07 PM By: Forrest Action: Added label and length to descriptor, eliminate Reference of FileTypes!
Time: July 18, 1979 4:07 PM By: Forrest Action: Add bootingInfo
Time: July 30, 1979 1:59 PM By: Forrest Action: Change Booting info to array [0..48) of UNSPECIFIED
Time: August 1, 1979 1:54 PM By: Redell Action: Remove dependency on SystemInternal; added systemID, VolumeAccess

```
-- Perhaps Template for "Offline SubVolume.Descriptor" should be declared here

DIRECTORY
    SubVolume: FROM "SubVolume" USING [Descriptor],
    System: FROM "System" USING [nullID, VolumeID];

PhysicalVolume: DEFINITIONS =
BEGIN

PageNumber: TYPE = LONG CARDINAL;          -- Physical disk page number

Handle: TYPE = LONG POINTER TO Descriptor;

seal, Seal: CARDINAL = 121212B; -- word zero of a valid physical volume descriptor (root page)
currentVersion: CARDINAL = 2; -- increment each time Descriptor below is reformatted

Descriptor: TYPE = MACHINE DEPENDENT RECORD[
        seal: CARDINAL ← Seal,        -- must be 1st field
        version: CARDINAL ← currentVersion,        -- must be 2nd field
        pvID: ID,
        subVolumeCount: [0..maxSubVols], fill: [0..177B] ← 0,
        labelLength: CARDINAL [0..physicalVolumeLabelLength),
        label: PACKED ARRAY [0..physicalVolumeLabelLength) OF CHARACTER,
        subVolumes: ARRAY [1..maxSubVols] OF offLine SubVolume.Descriptor];
maxSubVols: CARDINAL = 7; -- maximum subvolumes on a physical volume
physicalVolumeLabelLength: CARDINAL = 40;

rootPageNumber: PageNumber = 0;
ID: TYPE = System.VolumeID;
nullID: ID = [System.nullID];

END.
```

LOG

Time: March 14, 1979 2:59 PM By: Redell Action: Created file
Time: March 18, 1979 2:08 PM By: Redell Action: Reformatted descriptor and made machine dependent.
Time: July 24, 1979 3:35 PM By: Forrest Action: add code to support Physical Volume Names. Add stuff for Volume type, swattee, debugger, etc.. Made ID a type
Time: July 27, 1979 3:51 PM By: Forrest Action: undid most of last change (moved to logicalVolume)

```
DIRECTORY
    --File: FROM "File" USING [ID, Type],
    FileInternal: FROM "FileInternal" USING [FilePtr, PageGroup],
    LogicalVolume: FROM "LogicalVolume" USING [Handle, GroupPtr, PageNumber],
    Volume: FROM "Volume" USING [ID];

VolAllocMap: DEFINITIONS =
BEGIN


-- operations on whole volume (map)
Open: PROCEDURE [volume: LogicalVolume.Handle, initFlag, rebuildFlag: BOOLEAN];
Close: PROCEDURE [volume: LogicalVolume.Handle, final: BOOLEAN];


-- page group interfaces writing labels
-- GetPageGroup: PROCEDURE [volume: Volume.ID, hint: FileInternal.PageGroup] RETURNS [FileInternal.PageGroup];
AllocPageGroup: PROCEDURE [filePtr: FileInternal.FilePtr, groupPtr: LogicalVolume.GroupPtr, createFile: BOOLEAN];
    -- allocates a group of pages and writes labels (group is a modifiable hint)
FreePageGroup: PROCEDURE [filePtr: FileInternal.FilePtr, groupPtr: LogicalVolume.GroupPtr, deleteFile: BOOLEAN];
    -- free a page group (deleting file if filePage = 0 and deleteFile = TRUE);


-- single page interfaces writing labels
-- CreateZPage: PROCEDURE [file: File.ID, type: File.Type] RETURNS [page: LogicalVolume.PageNumber];
-- FreeZPage: PROCEDURE [page: LogicalVolume.PageNumber, fileD: FileInternal.Descriptor];
-- CreateVPage: PROCEDURE [file: File.ID]RETURNS [LogicalVolume.PageNumber];
-- FreeVPage: PROCEDURE [file: File.ID, page: LogicalVolume.PageNumber];

AccessVAM: PROCEDURE [volumePtr: LONG POINTER TO Volume.ID, volumePage: LogicalVolume.PageNumber, set: BOOLEAN,
        clear: BOOLEAN] RETURNS [busy: BOOLEAN];
    -- set, clear or read one bit of vam, always returns the (previous) value;


-- low level bit map interface (no label writing)
-- GetBusy: PROCEDURE [volumePage: LogicalVolume.PageNumber] RETURNS [busy: BOOLEAN];
    -- read the allocation state of page; TRUE if busy;
-- GetSetBusy: PROCEDURE [volumePage: LogicalVolume.PageNumber] RETURNS [busy: BOOLEAN];
    -- set a page to busy and return its previous state (semaphore like lock here)
-- SetFree: PROCEDURE [volumePage: LogicalVolume.PageNumber];
    -- free a page
-- GrabFirstFree: PROCEDURE [startPage: LogicalVolume.PageNumber] RETURNS [page: LogicalVolume.PageNumber];
    -- find first free page and set it busy
END.
```

LOG

Time: April 13, 1978 3:32 PM By: SP Action: Created file
Time: August 30, 1978 2:08 PM By: SP Action: use FilePtr's

```
DIRECTORY
    Environment: FROM "Environment" USING [bitsPerWord, wordsPerPage],
    File: FROM "File" USING [Capability, Type],
    FileInternal: FROM "FileInternal" USING [Descriptor, FilePtr, maxPermissions, PageGroup],
    FilePageLabel: FROM "FilePageLabel" USING [GetType, Label],
    FileTypes: FROM "FileTypes" USING [PilotVFileType, tFreePage, tVolumeAllocationMap, tVolumeFileMap],
    FMPrograms: FROM "FMPrograms",
    Inline: FROM "Inline" USING [BITAND, BITSHIFT, BITXOR],
    LabelTransfer: FROM "LabelTransfer" USING [ReadLabel, VerifyLabels, WriteLabels],
    LogicalVolume: FROM "LogicalVolume" USING [GroupPtr, Handle, nullVolumePage, PageNumber],
    SimpleSpace: FROM "SimpleSpace" USING [Create, ForceOut, Map, Page, Unmap],
    Space: FROM "Space" USING [Handle, WindowOrigin],
    Utilities: FROM "Utilities" USING [LongPointerFromPage, ShortCARDINAL],
    VolAllocMap: FROM "VolAllocMap",
    Volume: FROM "Volume" USING [ID];

VolAllocMapImpl: MONITOR
    IMPORTS FilePageLabel, Inline, LabelTransfer, SimpleSpace, Utilities
    EXPORTS FMPrograms, VolAllocMap
    SHARES File =
BEGIN OPEN Inline;

volumeP: POINTER TO LogicalVolume.Handle ← @volume; --name scope hack
volume: LogicalVolume.Handle ← NIL;
freeFileD: FileInternal.Descriptor;  --loaded by open

bitsPerPage: CARDINAL = (Environment.bitsPerWord*Environment.wordsPerPage);

saveBufferPage: LogicalVolume.PageNumber ← LogicalVolume.nullVolumePage;
bufferHandle: Space.Handle = SimpleSpace.Create[1, hyperspace];
bufferPointer: LONG POINTER = Utilities.LongPointerFromPage[SimpleSpace.Page[bufferHandle]];

interlace: CARDINAL ← 2;

quickFlag: BOOLEAN ← FALSE; -- DOESN'T WORK WITH quickFlag = TRUE

Open: PUBLIC PROCEDURE[volume: LogicalVolume.Handle, initFlag, rebuildFlag: BOOLEAN] =
    -- open, init or rebuild disk
    BEGIN OPEN volume;
    lace: CARDINAL;
    volumePage, bound: LogicalVolume.PageNumber;

    -- open
    IF volumeP↑ ~= NIL THEN Close[volumeP↑, TRUE];
    volumeP↑ ← volume;
    freeFileD ← FileInternal.Descriptor[free, vID, local[FALSE, FALSE, volume.volumeSize, FileTypes.tFreePage]];
    IF ~initFlag AND ~rebuildFlag THEN RETURN;    -- end of open

    -- init
    bound ← (IF initFlag THEN vfmStart ELSE volumeSize);
    IF ~initFlag AND quickFlag THEN
        FOR volumePage ← 0, volumePage + 1 WHILE volumePage<volumeSize DO
            IF AccessVAM[@volume.vID, volumePage, FALSE, FALSE] THEN bound ← volumePage + 100; ENDLOOP;
    LabelTransfer.WriteLabels[
        FileInternal.Descriptor[vam, vID, local[FALSE, FALSE, volumeSize, FileTypes.tVolumeAllocationMap]],
        FileInternal.PageGroup[vamStart, vamStart, vfmStart]];    --allocate the vam.
    bound ← MIN[bound, volumeSize];
    freePageCount ← volumeSize; -- isn't this too big?
    upperBound ← vfmStart;

    -- clear disk
    IF initFlag THEN LabelTransfer.WriteLabels[freeFileD, [vfmStart, vfmStart, volumeSize]]; --free
    FOR lace IN [0..interlace) DO -- hack to run disk as fast as possible
        FOR volumePage ← lace, volumePage + interlace WHILE volumePage<bound DO
            BEGIN
            IF volumePage>=volume.vfmStart THEN
                BEGIN
                label: FilePageLabel.Label ← LabelTransfer.ReadLabel[FileInternal.Descriptor[, vID, local[,,,]], 0, volumePage];
                type: File.Type ← FilePageLabel.GetType[@label];
```

```
                IF type = FileTypes.tFreePage OR type = FileTypes.tVolumeFileMap THEN
                    IF label.fileID = volume.free --AND otherwise a good free label-- THEN NULL
                    ELSE LabelTransfer.WriteLabels[freeFileD, FileInternal.PageGroup[volumePage, volumePage,
                            volumePage + 1]]
                ELSE GOTO Busy
                END
            ELSE GOTO Busy
            EXITS Busy => IF AccessVAM[@volume.vID, volumePage, TRUE, FALSE] THEN ERROR
            END
            ENDLOOP
        ENDLOOP
    END;

Close: PUBLIC PROCEDURE [volume: LogicalVolume.Handle, final: BOOLEAN] =
    -- flushes page cache
    BEGIN
    IF saveBufferPage = LogicalVolume.nullVolumePage THEN RETURN;
    IF final THEN
        BEGIN
        SimpleSpace.Unmap[bufferHandle];
        saveBufferPage ← LogicalVolume.nullVolumePage;
        volumeP↑ ← NIL;
        END
    ELSE SimpleSpace.ForceOut[bufferHandle];
    END;

AllocPageGroup: PUBLIC PROCEDURE [filePtr: FileInternal.FilePtr, groupPtr: LogicalVolume.GroupPtr, createFile: BOOLEAN] =
    -- allocates a group of pages and writes labels (group is a modifiable hint)
    BEGIN OPEN groupPtr;
    skip: [0..1] = IF ~createFile AND filePage = 0 THEN 1 ELSE 0;
    offset: LONG CARDINAL;
    startPage: LogicalVolume.PageNumber = MAX[volumePage, lowerBound];
    IF createFile OR filePage # 0 THEN
        BEGIN
        volumePage ← startPage;
        WHILE AccessVAM[@filePtr.volumeID, volumePage, TRUE, FALSE] DO        --search for unbusy page
            volumePage ← (IF volumePage = volume.volumeSize-1 THEN lowerBound ELSE volumePage + 1);
            IF volumePage = startPage THEN ERROR --Volume.InsufficientSpace--;      --avoid rather than back out of
            ENDLOOP;
        END;
    FOR offset ← 0, offset + 1 WHILE offset < nextFilePage-filePage DO
        IF volumePage + offset > = volume.volumeSize THEN EXIT;
        IF offset # 0 THEN IF AccessVAM[@filePtr.volumeID, volumePage + offset, TRUE, FALSE] THEN EXIT;        --first is set
        ENDLOOP;
    WITH filePtr SELECT FROM
        local => IF type IN FileTypes.PilotVFileType THEN filePage ← volumePage;
        ENDCASE;
    nextFilePage ← filePage + offset;    -- side effect
    IF createFile AND nextFilePage = 0 THEN offset ← 1;        -- creating only the attribute label page
    IF NOT LabelTransfer.VerifyLabels[filePtr↑, FileInternal.PageGroup[0, volumePage, skip]] THEN SIGNAL LabelError;
    IF NOT LabelTransfer.VerifyLabels[
        freeFileD,
        FileInternal.PageGroup[volumePage + skip, volumePage + skip, volumePage + offset]] THEN SIGNAL LabelError;
    WITH filePtr SELECT FROM
        local => size ← nextFilePage;    -- new size (to get labels right)
        ENDCASE => ERROR;
    LabelTransfer.WriteLabels[filePtr↑, FileInternal.PageGroup[filePage, volumePage, filePage + offset]];
    END;

FreePageGroup: PUBLIC PROCEDURE [filePtr: FileInternal.FilePtr, groupPtr: LogicalVolume.GroupPtr, deleteFile: BOOLEAN] =
    -- free a page group (deleting file if filePage = 0 and deleteFile = TRUE);
    BEGIN OPEN groupPtr;
    skip: [0..1] = IF ~deleteFile AND filePage = 0 THEN 1 ELSE 0;
    offset: LONG CARDINAL;
    FOR offset ← 0, offset + 1 WHILE offset < nextFilePage-filePage DO
        IF filePage + offset # 0 OR deleteFile THEN [] ← AccessVAM[@filePtr.volumeID, volumePage + offset, FALSE, TRUE];
        ENDLOOP;
    IF NOT LabelTransfer.VerifyLabels[filePtr↑, groupPtr↑] THEN SIGNAL LabelError;
    WITH filePtr SELECT FROM
        local => size ← filePage;    -- new size (to get labels right)
        ENDCASE => ERROR;
    LabelTransfer.WriteLabels[filePtr↑, FileInternal.PageGroup[0, volumePage, skip]];
```

```
    LabelTransfer.WriteLabels[
        freeFileD,
        FileInternal.PageGroup[volumePage + skip, volumePage + skip, volumePage + nextFilePage-filePage]];
    END;

lowerBound: LogicalVolume.PageNumber ← 0;   -- speedup KLUDGE

AccessVAM: PUBLIC ENTRY PROCEDURE [volumePtr: LONG POINTER TO Volume.ID, volumePage: LogicalVolume.PageNumber, set:
        BOOLEAN, clear: BOOLEAN] RETURNS [busy: BOOLEAN] =
    -- set, clear or read one bit of vam, always returns the (previous) value;
    BEGIN OPEN Inline;
    bit: WORD;
    word: LONG POINTER TO WORD;
    IF volume = NIL THEN ERROR;
    IF saveBufferPage # volume.vamStart + Utilities.ShortCARDINAL[volumePage]/bitsPerPage THEN
        BEGIN
        IF saveBufferPage # LogicalVolume.nullVolumePage THEN SimpleSpace.Unmap[bufferHandle];
        saveBufferPage ← volume.vamStart + Utilities.ShortCARDINAL[volumePage]/bitsPerPage;
        SimpleSpace.Map[bufferHandle, Space.WindowOrigin[
                File.Capability[volume.vam, FileInternal.maxPermissions],
                saveBufferPage], FALSE];
        END;
    word ← bufferPointer + (Utilities.ShortCARDINAL[volumePage] MOD bitsPerPage)/Environment.bitsPerWord;
    bit ← BITSHIFT[1, Utilities.ShortCARDINAL[volumePage] MOD Environment.bitsPerWord];
    busy ← (0 # BITAND[word↑, bit]);
    IF (busy AND clear) OR (~busy AND set) THEN
        BEGIN
        word↑ ← BITXOR[word↑, bit];
        volume.freePageCount ← volume.freePageCount + (IF busy THEN 1 ELSE -1);
        END;
    IF clear THEN lowerBound ← MIN[volumePage, lowerBound];        -- keep lowerBound to help allocator
    IF set AND volumePage ≥ lowerBound THEN lowerBound ← lowerBound + 1;  -- keep upperBound to help allocator
    IF set AND volumePage ≥ volume.upperBound THEN        -- keep upperBound to help scavenger speed up
        BEGIN
        volume.upperBound ← volumePage + 100;
        SimpleSpace.ForceOut[bufferHandle];        -- see that vam stays fairly current on volume
        END;
    END;

LabelError: SIGNAL = CODE;


END.
```

LOG

Time: April 13, 1978 3:32 PM By: Purcell Action: Created file
Time: June 23, 1978 12:27 PM By: Purcell Action: page 0 special case in alloc/free group
Time: September 27, 1978 5:15 PM By: Purcell Action: don't raise signals and limit bound
Time: October 19, 1978 2:45 PM By: Purcell Action: CR 20.103: Use FileTypes directly
Time: March 20, 1979 9:30 PM By: Redell Action: Convert to Mesa 5.0
Time: August 1, 1979 3:10 PM By: Redell Action: Convert to use FilePageLabel
Time: August 8, 1979 12:05 PM By: Redell Action: Bug fix: wraparound condition in AllocPageGroup was off by one: ran off end of volume.

```
DIRECTORY
    File: FROM "File" USING [PageNumber],
    FileInternal: FROM "FileInternal" USING [Descriptor, FilePtr, PageGroup],
    LogicalVolume: FROM "LogicalVolume" USING [FileAccessProc, GroupPtr, Handle, Interval, Key, Level, PageNumber];

VolFileMap: DEFINITIONS =

BEGIN


Key: TYPE = LogicalVolume.Key;
Interval: TYPE = LogicalVolume.Interval;
Level: TYPE = LogicalVolume.Level;


-- public interfaces
Open: PROCEDURE [volume: LogicalVolume.Handle, initFlag, rebuildFlag: BOOLEAN];
Close: PROCEDURE [volume: LogicalVolume.Handle, final: BOOLEAN];

-- GetFileDescriptor: PROCEDURE [file: File.ID] RETURNS [FileInternal.Descriptor];
-- InsertFileDescriptor: PROCEDURE [volume: LogicalVolume.Handle, file: File.ID, volumePage: LogicalVolume.PageNumber];
-- DeleteFileDescriptor: PROCEDURE [file: File.ID] RETURNS [page: LogicalVolume.PageNumber];

GetNextFileProc: LogicalVolume.FileAccessProc;
    -- enumerates files on a volume (called through FileAccess)

GetPageGroup: PROCEDURE [filePtr: FileInternal.FilePtr, filePage: File.PageNumber] RETURNS [success: BOOLEAN, group:
        FileInternal.PageGroup];
    -- finds page group containing key (filePage = nextFilePage = size when off end of file)
InsertPageGroup: PROCEDURE [filePtr: FileInternal.FilePtr, groupPtr: LogicalVolume.GroupPtr];
    -- inserts a pageGroup into B-tree (unordered inserts are merged for rebuild)
DeletePageGroup: PROCEDURE [filePtr: FileInternal.FilePtr, groupPtr: LogicalVolume.GroupPtr];
    -- deletes a pageGroup suffix (leaving zeroGroup unless explicitly asked)


-- unmonitored debugging interfaces (use with risk)

Get: PROCEDURE [key: Key, level: Level] RETURNS [Interval];
Insert: PROCEDURE [key: Key, volumePage: LogicalVolume.PageNumber, level: Level];
Delete: PROCEDURE [key: Key, level: Level];
Lower: PROCEDURE [a,b: Key] RETURNS [BOOLEAN];

END.
```

LOG

Time: April 13, 1978 2:10 PM By: SP Action: Created file
Time: timeStamp By: initials Action: shortDescription

```
DIRECTORY
    Environment: FROM "Environment" USING [wordsPerPage],
    File: FROM "File" USING [Capability, lastPageNumber, PageNumber],
    FileInternal: FROM "FileInternal" USING [Descriptor, FilePtr, maxPermissions, PageGroup],
    FilePageLabel: FROM "FilePageLabel" USING [GetFilePage, GetType, Label, nullID],
    FileTypes: FROM "FileTypes" USING [tFreePage, tVolumeFileMap],
    FMPrograms: FROM "FMPrograms",
    Inline: FROM "Inline" USING [LongCOPY],
    LabelTransfer: FROM "LabelTransfer" USING [ReadLabel],
    LogicalVolume: FROM "LogicalVolume" USING [FileAccessProc, GroupPtr, Handle, Interval, Key, Level, maxKey, maxID,
        nullInterval, nullKey, nullVolumePage, PageNumber],
    SimpleSpace: FROM "SimpleSpace" USING [Create, ForceOut, Map, Page, Unmap],
    Space: FROM "Space" USING [Handle, WindowOrigin],
    Utilities: FROM "Utilities" USING [LongPointerFromPage],
    VolAllocMap: FROM "VolAllocMap" USING [AccessVAM, AllocPageGroup, FreePageGroup],
    VolFileMap: FROM "VolFileMap";

VolFileMapImpl: MONITOR
        IMPORTS FilePageLabel, Inline, LabelTransfer, SimpleSpace, Utilities, VolAllocMap
        EXPORTS FMPrograms, VolFileMap
        SHARES File =
BEGIN OPEN LogicalVolume;


-- data types for the vfm

Buffer: TYPE = RECORD [  --This is a B-tree page
    -- fixed length, uncompressed indexes are stored now now but in the future compressed indexes will have variable lengths.
            Compressed format is defined by VolFileMap.PutNext and VolFileMap.ReadNext
    -- data: PACKED ARRAY [0..pageSize-1) OF UNSPECIFIED,--          --can be array of Bytes
    data: ARRAY [0..indexInBuffer) OF Index,
    used: BufferPtr];

BufferPtr: TYPE = INTEGER;

Index: TYPE = RECORD [
    key: Key,
    volumePage: PageNumber,
    ptr: BufferPtr];

indexInBuffer: CARDINAL = (Environment.wordsPerPage-SIZE[BufferPtr])/SIZE[Index];
maxReadPtr: CARDINAL = SIZE[Buffer]-SIZE[BufferPtr];
maxIndex: Index = Index[maxKey, nullVolumePage, 0];
nullIndex: Index = Index[nullKey, nullVolumePage, 0];

Context: TYPE = RECORD [
    buffer: LONG POINTER TO Buffer,
    tempBuffer: LONG POINTER TO Buffer,
    xtraBuffer: LONG POINTER TO Buffer,
    interval: Interval,
    old: Index,
    low: Index,
    high: Index];

-- global state for communication between Find, ReadNext, PutNext, Insert, Delete
    volumeP: POINTER TO LogicalVolume.Handle ← @volume;  --name scope hack
    volume: LogicalVolume.Handle ← NIL;  --set on entry
    bufferHandle: Space.Handle = SimpleSpace.Create[1, hyperspace];
    xtraHandle: Space.Handle = SimpleSpace.Create[1, hyperspace];
    tBuf: Buffer;  --used by insert to avoid overlapped copy
    context: Context ← Context[
        buffer: Utilities.LongPointerFromPage[SimpleSpace.Page[bufferHandle]],
        tempBuffer: @tBuf,
        xtraBuffer: Utilities.LongPointerFromPage[SimpleSpace.Page[xtraHandle]],
        interval: nullInterval,
        old: nullIndex,
        low: nullIndex,
        high: nullIndex];
```

```
-- Public Procedures

interlace: CARDINAL ← 1;

Open: PUBLIC PROCEDURE[volume: LogicalVolume.Handle, initFlag, rebuildFlag: BOOLEAN] =
    -- opens, inits or rebuilds the vfm assuming undamaged files (calls insertPageGroup)
    BEGIN
    label: FilePageLabel.Label;
    lace: CARDINAL;
    level: LogicalVolume.Level;
    volumePage: LogicalVolume.PageNumber;
    fileD: FileInternal.Descriptor ← [ , , local[,,,]];    -- template for scavenger
    group: FileInternal.PageGroup;

    -- open
    IF volumeP↑ ~= NIL THEN Close[volumeP↑, TRUE];
    volumeP↑ ← volume;
    IF ~initFlag AND ~rebuildFlag THEN RETURN;    -- end of open

    -- init
    IF volume.vfmStart # CreateVPage[] THEN ERROR;
    FOR level DECREASING IN [0..volume.treeLevel) DO        --init the tree
        Insert[nullKey, (IF level = 0 THEN nullVolumePage ELSE CreateVPage[]), level];
        ENDLOOP;
    IF initFlag THEN RETURN;    -- end of init

    -- rebuild
    fileD.volumeID ← volume.vID;
    FOR lace IN [0..interlace) DO    -- hack to run disk as fast as possible
        FOR volumePage ← lace, volumePage + interlace WHILE volumePage<volume.volumeSize DO
            IF VolAllocMap.AccessVAM[@volume.vID, volumePage, FALSE, FALSE] THEN
                BEGIN
                label ← LabelTransfer.ReadLabel[fileD, 0, volumePage];
                IF FilePageLabel.GetType[@label] ~IN [FileTypes.tFreePage..FileTypes.tVolumeFileMap] THEN
                    BEGIN
                    filePage: File.PageNumber = FilePageLabel.GetFilePage[@label];
                    fileD.fileID ← label.fileID;
                    group ← [filePage, volumePage, (IF filePage = 0 AND label.zeroSize THEN 0 ELSE filePage + 1)];
                    InsertPageGroup[@fileD, @group]
                    END
                END
            ENDLOOP
        ENDLOOP
    END;


Close: PUBLIC ENTRY PROCEDURE [volume: LogicalVolume.Handle, final: BOOLEAN] =
    -- flushes page cache
    BEGIN
    IF saveBufferPage = LogicalVolume.nullVolumePage THEN RETURN;
    IF final THEN
        BEGIN
        SimpleSpace.Unmap[bufferHandle];
        saveBufferPage ← LogicalVolume.nullVolumePage;
        volumeP↑ ← NIL;
        END
    ELSE SimpleSpace.ForceOut[bufferHandle];
    END;

GetPageGroup: PUBLIC ENTRY PROCEDURE [filePtr: FileInternal.FilePtr, filePage: File.PageNumber] RETURNS [success: BOOLEAN,
        group: FileInternal.PageGroup] =
    -- finds page group containing key (filePage = nextFilePage = size when off end of file)
    BEGIN
    interval: Interval = Get[Key[filePtr.fileID, filePage], 0];
    RETURN[interval.key.fileID = filePtr.fileID, FileInternal.PageGroup[
        filePage: interval.key.filePage,
        volumePage: interval.volumePage,
        nextFilePage: (IF interval.nextKey.fileID = filePtr.fileID THEN interval.nextKey ELSE interval.key).filePage]];    --covers page
                zero and size requests
    END;
```

```
InsertPageGroup: PUBLIC ENTRY PROCEDURE [filePtr: FileInternal.FilePtr, groupPtr: LogicalVolume.GroupPtr] =
    -- inserts a pageGroup into B-tree (unordered inserts are merged for rebuild)
    BEGIN
    key: Key ← [filePtr.fileID, groupPtr.filePage];
    interval: Interval ← Get[key, 0];
    IF interval.key = key THEN BEGIN Delete[key, 0]; interval ← Get[key, 0]; END;
    IF key.filePage-interval.key.filePage # groupPtr.volumePage-interval.volumePage
        OR key.fileID # interval.key.fileID THEN Insert[key, groupPtr.volumePage, 0];      --merge with previous
    key.filePage ← groupPtr.nextFilePage;
    IF interval.nextKey # key AND groupPtr.filePage # groupPtr.nextFilePage THEN Insert[key, nullVolumePage, 0];
    IF interval.nextKey = key AND Get[key, 0].volumePage = interval.volumePage + interval.nextKey.filePage-interval.key.filePage
            THEN Delete[key, 0];    --merge with following
    END;

DeletePageGroup: PUBLIC ENTRY PROCEDURE [filePtr: FileInternal.FilePtr, groupPtr: LogicalVolume.GroupPtr] =
    -- deletes a pageGroup suffix (leaving zeroPageGroup unless explicitly asked) returns group deleted into groupPtr
    BEGIN
    key: Key ← [filePtr.fileID, groupPtr.nextFilePage-(IF groupPtr.nextFilePage = 0 THEN 0 ELSE 1)];
    interval: Interval ← Get[key, 0];
    IF ----interval.key.fileID # filePtr.fileID OR---- interval.volumePage = nullVolumePage THEN ERROR;
    groupPtr.filePage ← key.filePage ← MAX[interval.key.filePage, groupPtr.filePage];
    groupPtr.volumePage ← interval.volumePage + groupPtr.filePage - interval.key.filePage;
    IF groupPtr.filePage # 0 THEN
        BEGIN
        IF interval.key = key THEN Delete[key, 0];
        Insert[key, nullVolumePage, 0];        -- no check for merging since preceding is assumed present
        END;
    key.filePage ← groupPtr.nextFilePage;
    Delete[key, 0];      -- no check for merging since following is assumed vacant
    END;

GetNextFileProc: PUBLIC LogicalVolume.FileAccessProc =
    -- enumerates files on a volume (called through FileAccess)
    BEGIN
    filePtr.fileID ← Get[Key[filePtr.fileID, File.lastPageNumber], 0].nextKey.fileID;
    IF filePtr.fileID = LogicalVolume.maxID THEN filePtr.fileID ← FilePageLabel.nullID;      -- end with null
    END;


-- Internal B-tree procedures (half public for testing)

Get: PUBLIC --INTERNAL-- PROCEDURE [key: Key, level: Level] RETURNS [interval: Interval] =
    BEGIN
    -- returns value without reading a page when possible
    Get1: FindProc =
        BEGIN OPEN context;
        volume.interval[level] ← Interval[low.key, high.volumePage, high.key];
        END;
    IF level > volume.treeLevel OR level > LAST[Level] THEN ERROR;
    IF level = volume.treeLevel THEN RETURN[Interval[nullKey, volume.vfmStart, maxKey]];
    interval ← volume.interval[level];        -- try the cached intry
    IF Lower[key, interval.key] OR ~Lower[key, interval.nextKey] THEN Find[key, level, Get1];
    RETURN[volume.interval[level]];
    END;

Insert: PUBLIC --INTERNAL-- PROCEDURE [key: Key, volumePage: LogicalVolume.PageNumber, level: Level] =
    -- inserts (key, volumePage (including duplicates) calling split if necessary
    BEGIN
    splitFlag: BOOLEAN;
    Insert1: FindProc =
        BEGIN OPEN context;
        IF (splitFlag ← (buffer.used > maxReadPtr-SIZE[Index])) THEN RETURN;
        Inline.LongCOPY[buffer+high.ptr, buffer.used - high.ptr, tempBuffer];
        IF low.ptr < buffer.used THEN PutNext[key, high.volumePage, level];
        PutNext[high.key, volumePage, level];
        Inline.LongCOPY[tempBuffer, buffer.used - high.ptr, buffer+low.ptr];
        buffer.used ← low.ptr + buffer.used - high.ptr;
        END;
    Find[key, level, Insert1];
    IF splitFlag THEN
        BEGIN
        Split[key, level];
        Find[key, level, Insert1];
```

```
        END;
    END;

DeleteError: SIGNAL = CODE;

Delete: PUBLIC --INTERNAL-- PROCEDURE [key: Key, level: Level] =
    -- deletes the index <= the key, error if no index (merge is called from find)
    BEGIN
    firstFlag, lastFlag: BOOLEAN;
    volumePage: LogicalVolume.PageNumber;   -- page holding key (delete if firstFlag AND lastFlag)
    nextKey: Key;       -- the following key must be slid down over deleted key
    Delete1: FindProc =
        BEGIN OPEN context;
        firstFlag ← (low.ptr = 0);
        lastFlag ← (high.ptr = buffer.used);
        volumePage ← interval.volumePage;
        nextKey ← high.key;
        IF low.key~ = key THEN SIGNAL DeleteError;
        IF firstFlag AND NOT lastFlag THEN Inline.LongCOPY[buffer + high.ptr, (buffer.used ← buffer.used - high.ptr), buffer];
        END;
    Delete2: FindProc =
        BEGIN OPEN context;
        high ← Index[nextKey, low.volumePage, high.ptr];
        low ← old;
        Inline.LongCOPY[buffer + high.ptr, buffer.used - high.ptr, tempBuffer];
        PutNext[high.key, high.volumePage, level];
        Inline.LongCOPY[tempBuffer, buffer.used - high.ptr, buffer + low.ptr];
        buffer.used ← low.ptr + buffer.used - high.ptr;
        END;
    Find[key, level, Delete1];      -- get the preceeding index
    IF firstFlag THEN
        BEGIN
        Delete[key, level + 1];
        IF lastFlag THEN FreeVPage[volumePage]  --free after remaping
            ELSE Insert[nextKey, volumePage, level + 1];
        END;
    Find[key, level, Delete2];      -- get the preceeding index
    END;

Lower: PUBLIC --INTERNAL-- PROCEDURE [a,b: Key] RETURNS [BOOLEAN] =
    -- compares two keys for ordering
    BEGIN
    x: POINTER TO ARRAY [0..4) OF CARDINAL = LOOPHOLE[@a.fileID];
    y: POINTER TO ARRAY [0..4) OF CARDINAL = LOOPHOLE[@b.fileID];
    i: [0..4);
    FOR i IN [0..4) DO
        SELECT TRUE FROM
            (x[i] < y[i]) => RETURN[TRUE];
            (x[i] > y[i]) => RETURN[FALSE];
            ENDCASE;        -- if equal keep checking
        ENDLOOP;
    RETURN[a.filePage < b.filePage OR b = maxKey];      -- maxKey < maxKey, to close key space
    END;

FindProc: TYPE = PROCEDURE;

saveBufferPage: LogicalVolume.PageNumber ← LogicalVolume.nullVolumePage;

Find: --INTERNAL-- PROCEDURE [key: Key, level: Level, proc: FindProc] =
    -- executes proc with context (buffer, low, high) surrounding key (merges too)
    BEGIN OPEN context;
    context.interval ← Get[key, level + 1];
    IF interval.volumePage # saveBufferPage THEN
        BEGIN
        IF saveBufferPage # LogicalVolume.nullVolumePage THEN SimpleSpace.Unmap[bufferHandle];
        SimpleSpace.Map[bufferHandle, Space.WindowOrigin[File.Capability[volume.vfm, FileInternal.maxPermissions],
                interval.volumePage], FALSE];
        saveBufferPage ← interval.volumePage;
        END;
    old ← low ← high ← Index[interval.key, nullVolumePage, 0];      --init reader
    UNTIL Lower[key, high.key] DO ReadNext[]; ENDLOOP;          --scan this page till key is passed
    proc[];     -- test before 'unmap'. Merge after; postpone Unmap until later to optimize out
    IF buffer.used <= SIZE[Buffer]/3 AND interval.nextKey # maxKey THEN Merge[old.key, level]; --is interval.key correct?
            (consider merge)
    END;
```

```
Split: --INTERNAL-- PROCEDURE [key: Key, level: Level] =
    -- moves half of buffer (or root) to xtraBuffer, creating new page of tree
    BEGIN
    keyStone: LogicalVolume.Key;        -- half way mark
    page: LogicalVolume.PageNumber ← CreateVPage[];
    Split1: FindProc =
        BEGIN
        Xtra[page, Split2];
        END;
    Split2: XtraProc =
        BEGIN OPEN context;
        high ← Index[interval.key, nullVolumePage, 0];        -- quick readReset
        UNTIL high.ptr > buffer.used/2 DO ReadNext[]; ENDLOOP;
        Inline.LongCOPY[buffer+low.ptr, (xtraBuffer.used ← buffer.used-low.ptr), xtraBuffer];        --split
        buffer.used ← low.ptr;
        --volume.interval[level+1].nextKey ← low.key;--        -- interval of buffer is smaller (must keep high key above right)????
        keyStone ← low.key;
        END;
    Find[key, level, Split1];
    Insert[keyStone, page, level+1];
    END;

Merge: --INTERNAL-- PROCEDURE [key: Key, level: Level] =
    -- tries to merge page of oldInterval with next page at same level or with root
    -- cannot merge last page of any level except rootLevel
    BEGIN
    mergeFlag: BOOLEAN;
    leftInterval, rightInterval: Interval;
    Merge1: FindProc =
        BEGIN OPEN context;
        rightInterval ← interval;
        Xtra[leftInterval.volumePage, Merge2];
        END;
    Merge2: XtraProc =
        BEGIN OPEN context;
        xtraBufferUsed: INTEGER ← xtraBuffer.used;        -- used to solve stack modeling error
        IF level = volume.treeLevel-1 THEN xtraBuffer.used ← 0;    --clear now instead of deleteing later
        IF (mergeFlag ← (buffer.used + xtraBuffer.used < SIZE[Buffer])) THEN --is merging possible
            BEGIN    --merge pages
            Inline.LongCOPY[buffer, buffer.used, xtraBuffer + xtraBufferUsed];    --merge buffer with xtra
            xtraBuffer.used ← xtraBuffer.used + buffer.used;
            -- buffer.used remains to prevent Find from attempting a merge
            --interval.volumePage ← ??Interval.volumePage;--        --update cache with new page for deleted contents
            END
        ELSE
            BEGIN    --balance pages simply to provide hysteresis against futil merge attempts
            WHILE low.ptr < (buffer.used-xtraBuffer.used)/2 DO ReadNext[]; ENDLOOP;--find middle
            Inline.LongCOPY[buffer, low.ptr, xtraBuffer + xtraBufferUsed];    --move first of buffer to xtra
            Inline.LongCOPY[buffer+low.ptr, buffer.used-low.ptr, buffer];    --slide down the rest of buffer
            xtraBuffer.used ← xtraBuffer.used + low.ptr;
            buffer.used ← buffer.used - low.ptr;        --use low to insert while it is still valid
            rightInterval.key ← low.key;
            END;
        END;
    leftInterval ← Get[key, level+1];        -- so as to get a valid volumePage
    Find[leftInterval.nextKey, level, Merge1];        --beware the merging
    Delete[leftInterval.nextKey, level+1];
    IF mergeFlag THEN FreeVPage[rightInterval.volumePage] ELSE
            Insert[rightInterval.key, rightInterval.volumePage, level+1];        -- insert new index:
    END;

ReadNext: --INTERNAL-- PROCEDURE =
    -- decompresses item at high to become low & bumps high
    -- Note the side effect on low and high ! !
    -- no compression is implemented in this version
    BEGIN OPEN context;
    IF high.ptr > buffer.used THEN ERROR;
    old ← low;
    low ← high;
    high ← (IF high.ptr < buffer.used
        THEN LOOPHOLE[buffer+high.ptr, LONG POINTER TO Index]↑
        ELSE Index[maxKey, nullVolumePage, 0]);
    high.ptr ← high.ptr + low.ptr;    --high.ptr was just an increment
```

```
    END;

PutNext: --INTERNAL-- PROCEDURE [key: Key, volumePage: LogicalVolume.PageNumber, level: Level] =
    -- compresses item in the context of low
    -- Note the side effect on low 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
    BEGIN OPEN context;
        -- merge page groups when possible
    old ← low;
    low ← (LOOPHOLE[buffer + low.ptr, LONG POINTER TO Index]↑ ← Index[key, volumePage, SIZE[Index]]);   --.ptr is just an
            increment
    low.ptr ← low.ptr + old.ptr;    --low.ptr was just an increment
    volume.interval[level] ← Interval[old.key, low.volumePage, low.key];   -- keep cache up to date in the face of changes
    END;

CreateVPage: --INTERNAL-- PROCEDURE RETURNS [LogicalVolume.PageNumber] =
    -- calls VolAllocMap.AllocPageGroup to get a new page for the vfm B-tree
    BEGIN
    group: FileInternal.PageGroup ← [0, 0, 1];
    vfmFileD: FileInternal.Descriptor ← FileInternal.Descriptor[volume.vfm, volume.vID, local[FALSE, FALSE, volume.volumeSize,
            FileTypes.tVolumeFileMap]];
    VolAllocMap.AllocPageGroup[@vfmFileD, @group, TRUE];
    RETURN[group.volumePage];
    END;

FreeVPage: --INTERNAL-- PROCEDURE [volumePage: LogicalVolume.PageNumber] =
    -- calls VolAllocMap.FreePageGroup to free a page of the vfm B-tree
    BEGIN
    group: FileInternal.PageGroup ← [volumePage, volumePage, volumePage + 1];
    vfmFileD: FileInternal.Descriptor ← FileInternal.Descriptor[volume.vfm, volume.vID, local[FALSE, FALSE, volume.volumeSize,
            FileTypes.tVolumeFileMap]];
    VolAllocMap.FreePageGroup[@vfmFileD, @group, TRUE];
    END;

XtraProc: TYPE = PROCEDURE;

Xtra: --INTERNAL-- PROCEDURE [page: LogicalVolume.PageNumber, proc: XtraProc] =
    -- maps, operates, unmaps xtraBuffer
    BEGIN
    SimpleSpace.Map[xtraHandle, Space.WindowOrigin[File.Capability[volume.vfm, FileInternal.maxPermissions], page], FALSE];
    proc[];
    SimpleSpace.Unmap[xtraHandle];
    END;

END.
```

LOG

Time: April 13, 1978 1:41 PM By: Purcell Action: Created file
Time: April 28, 1978 4:17 PM By: Purcell Action: Compiled
Time: May 8, 1978 11:23 AM By: Purcell Action: added testing
Time: May 10, 1978 3:32 PM By: Purcell Action: first test
Time: May 11, 1978 7:46 PM By: Purcell Action: insert/split works
Time: May 12, 1978 2:19 PM By: Purcell Action: revised delete/merge
Time: May 18, 1978 8:06 PM By: Purcell Action: 65,000 successful tests
Time: May 23, 1978 2:34 PM By: Purcell Action: changed delete to be inverse of insert
Time: June 22, 1978 3:39 PM By: Purcell Action: forgot to close buffer in merge (1 case)
Time: June 29, 1978 10:49 PM By: Purcell Action: use well nested buffer access
Time: July 6, 1978 2:36 PM By: Purcell Action: added rebuild and init
Time: September 26, 1978 12:49 AM By: Purcell Action: CR fix to InsertPageGroup to merge correctly with following group
Time: September 26, 1978 12:50 AM By: Purcell Action: CR fix to Merge to slide the right number of words when balancing
Time: October 19, 1978 1:32 PM By: Purcell Action: CR 90: try interlace = 1
Time: October 19, 1978 2:49 PM By: Purcell Action: CR 103: Use FileTypes directly
Time: March 20, 1979 10:13 PM By: Redell Action: Convert to Mesa 5.0
Time: August 2, 1979 11:08 AM By: Redell Action: Update to use FilePageLabel

```
DIRECTORY
    Boot: FROM "Boot" USING [Location, LVBootFiles, nullDiskFileID, VolumeType],
    DiskChannel: FROM "DiskChannel" USING [Create, DiskPageCount, Drive, GetAttributes, GetDriveAttributes,
        GetNextDrive, Handle, nullDrive, SetDriveTag],
    Environment: FROM "Environment" USING [bitsPerWord, wordsPerPage],
    File: FROM "File" USING [Capability, delete, grow, ID, nullCapability, nullID, PageCount, shrink, Type],
    FileCache: FROM "FileCache" USING [SetFile, SetPageGroup, FlushFile],
    FileInternal: FROM "FileInternal" USING [Descriptor, maxPermissions, PageGroup],
    FilePageLabel: FROM "FilePageLabel" USING [GetFilePage, GetType, Label, nullID],
    FMPrograms: FROM "FMPrograms",
    Inline: FROM "Inline" USING [BITAND],
    KernelFile: FROM "KernelFile",
    LabelTransfer: FROM "LabelTransfer" USING [LabelStatus, ReadLabel, ReadRootLabel, WriteLabels],
    LogicalVolume: FROM "LogicalVolume" USING [create, currentVersion, FileAccessProc, Free, Handle,
        LogicalSubvolumeMarker, LSMCurrentVersion, LSMSeal, noOp, nullID, nullRootFileIDs, Operations, PageNumber,
        rootPageNumber, seal, SetFree, SetVam, SetVfm, setVolumeAttributes, Vam, Vfm],
    OISProcessorFace: FROM "OisProcessorFace" USING [DeviceType],
    PhysicalVolume: FROM "PhysicalVolume" USING [currentVersion, Descriptor, Handle, ID, IDCheckSum, PageCount,
        PageNumber, PhysicalSubvolumeMarker, PSMCurrentVersion, PSMSeal, rootPageNumber, seal, SubVolumeDesc],
    PilotFileTypes: FROM "PilotFileTypes" USING [PilotRootFileType, tFreePage, tLogicalVolumeRootPage,
        tPhysicalVolumeRootPage, tVolumeAllocationMap, tVolumeFileMap, tSubVolumeMarkerPage],
    PilotSwitches: FROM "PilotSwitches" USING [switches -- E, L, S, U, X, Z --],
    Runtime: FROM "Runtime" USING [CallDebugger],
    SimpleSpace: FROM "SimpleSpace" USING [Create, ForceOut, Handle, Map, Page, Unmap],
    StoragePrograms: FROM "StoragePrograms" USING [nullDevice],
    SubVolume: FROM "SubVolume" USING [completion, Find, GetNext, Handle, OnLine],
    System: FROM "System" USING [GetUniversalID, NetworkAddress, UniversalID],
    Utilities: FROM "Utilities" USING [LongPointerFromPage, ShortCARDINAL],
    VolAllocMap: FROM "VolAllocMap" USING [Open, Close],
    VolFileMap: FROM "VolFileMap" USING [Open, Close],
    Volume: FROM "Volume" USING [ID, nullID, PageCount];

VolumeImpl:
    MONITOR [bootFile: POINTER TO disk Boot.Location, pLVBootFiles: POINTER TO Boot.LVBootFiles] RETURNS [debugClass:
    Boot.VolumeType, debuggerDevice: UNSPECIFIED --OISProcessorFace.DeviceHandle--]
    IMPORTS DiskChannel, FileCache, FilePageLabel, Inline, LabelTransfer, LogicalVolume, PhysicalVolume, PilotSwitches,
    Runtime, SimpleSpace, SubVolume, System, Utilities, VolAllocMap, VolFileMap
    EXPORTS FMPrograms, KernelFile, LogicalVolume, Volume
    SHARES File =
BEGIN

BarePvID: TYPE = System.UniversalID; -- basic component of Phys ID

FileAccessProc: TYPE = LogicalVolume.FileAccessProc;

-- Someday, this may live in a seperate file.  The Pads are so each part may grow somewhat independently without trashing the other on existing
volumes.  This may be only of marginal use, but it doesn't cost much
SubVolumeFill1: TYPE = [SIZE[LogicalVolume.LogicalSubvolumeMarker]..Environment.wordsPerPage/2);
SubVolumeFill2: TYPE =
    [Environment.wordsPerPage/2 + SIZE[PhysicalVolume.PhysicalSubvolumeMarker]..Environment.wordsPerPage-1);

SubVolumeMarkerPage: TYPE = MACHINE DEPENDENT RECORD[
    logical: LogicalVolume.LogicalSubvolumeMarker,
    fill1: ARRAY SubVolumeFill1 OF WORD ← NULL,
    physical: PhysicalVolume.PhysicalSubvolumeMarker,
    fill2: ARRAY SubVolumeFill2 OF WORD ← NULL,
    checksum: CARDINAL ← 0];

clientType: Boot.VolumeType; -- set from program paramaters
debuggerIDsSet: BOOLEAN ← FALSE; -- if false, we haven't found a debugger yet
systemID: PUBLIC Volume.ID ← Volume.nullID; -- currently only a single volume is supported
cautious: BOOLEAN ← TRUE; -- force out after each file operation

-- The "active volume" whose LogicalVolume.Descriptor (root page) is buffered in memory
logicalRootPageBuffer: SimpleSpace.Handle = SimpleSpace.Create[1, hyperspace];
activeVolume: LogicalVolume.Handle = Utilities.LongPointerFromPage[SimpleSpace.Page[logicalRootPageBuffer]];
activeVID: Volume.ID ← Volume.nullID;  -- active volume (nullID => no active volume)

-- Buffer variables for accessing physical volume root page
physicalRootPageBuffer: SimpleSpace.Handle = SimpleSpace.Create[1, hyperspace];
physicalVolume: PhysicalVolume.Handle = Utilities.LongPointerFromPage[SimpleSpace.Page[physicalRootPageBuffer]];
```

```
-- Buffer variables for accessing marker pages
markerPageBuffer: SimpleSpace.Handle = SimpleSpace.Create[1, hyperspace];
marker: LONG POINTER TO SubVolumeMarkerPage = Utilities.LongPointerFromPage[SimpleSpace.Page[markerPageBuffer]];

VolImplError: ERROR[{notFoundInOffLine, lvTableFull, notFoundVolumeInActivate}] = CODE;
InsufficientSpace: PUBLIC ERROR = CODE;
Unknown: PUBLIC ERROR [volume: Volume.ID] = CODE;
UnOpened: PUBLIC ERROR [volume: Volume.ID] = CODE; -- Someday Actually Exported

-- Monitor External Procedures

GetAttributes: PUBLIC PROCEDURE [volume: Volume.ID] RETURNS [volumeSize, freePageCount: Volume.PageCount, rootFile:
File.Capability] =
    BEGIN
    GetAttributes1: PROCEDURE [vol: LogicalVolume.Handle] =
        BEGIN
        volumeSize ← vol.volumeSize;
        -- ??? free age count junk ???
        freePageCount ← vol.freePageCount-Utilities.ShortCARDINAL[vol.freePageCount/16];
        freePageCount ← freePageCount-MIN[freePageCount, 5]; -- keep freePageCount non negative
        rootFile ← vol.clientRootFile;
        END;
    VolumeAccess[@volume, GetAttributes1, FALSE];
    END;

GetLabelString: PUBLIC PROCEDURE [volume: Volume.ID, s: STRING] =
    BEGIN
    GetLabelString1: PROCEDURE [vol: LogicalVolume.Handle] =
        BEGIN
        i: CARDINAL;
        s.length ← MIN[s.maxlength, vol.labelLength, LENGTH[vol.label]];
        FOR i IN [0..s.length) DO s[i] ← vol.label[i]; ENDLOOP;
        END;
    IF s#NIL THEN VolumeAccess[@volume, GetLabelString1, FALSE];
    END;

-- get file with maxpermissions of given (Root) type from volume root page
GetRootFile: PUBLIC PROCEDURE [type: File.Type, volume: Volume.ID] RETURNS [file: File.Capability] =
    BEGIN
    GetRootFile1: PROCEDURE [vol: LogicalVolume.Handle] =
        BEGIN file ← [vol.rootFileID[type], FileInternal.maxPermissions]; END;
    IF type NOT IN PilotFileTypes.PilotRootFileType THEN ERROR;
    VolumeAccess[@volume, GetRootFile1, FALSE];
    IF file.fID = FilePageLabel.nullID THEN file ← File.nullCapability;
    END;

IsOnServer: PUBLIC PROCEDURE [volume: Volume.ID, netAddress: System.NetworkAddress] =
    BEGIN
    --not implemented
    END;

-- This may have to get smarter when we have boot messages
IsMyDebuggerVolume: PROCEDURE [vol: LogicalVolume.Handle] RETURNS [BOOLEAN] =
    BEGIN
    IF vol.seal#LogicalVolume.seal OR vol.version<=1 OR debuggerIDsSet OR vol.bootingInfo[debugger]=Boot.nullDiskFileID OR
       vol.bootingInfo[debuggee]=Boot.nullDiskFileID THEN RETURN[FALSE];
    SELECT clientType FROM
        nonPilot, debuggerDebugger => RETURN[FALSE]; -- these Guys have no debugger
        normal => RETURN[vol.type=debugger];
        debugger => RETURN[vol.type=debuggerDebugger];
        ENDCASE => ERROR;
    END;

PhysicalRootLabelCheck: PROCEDURE[label: FilePageLabel.Label] RETURNS [BarePvID, BOOLEAN] =
    BEGIN OPEN label;
    -- could also check pad2...
    RETURN[fileID, fileID#FilePageLabel.nullID AND
       FilePageLabel.GetFilePage[@label]=0 AND
       ~immutable AND ~temporary AND ~zeroSize AND pad1=0 AND
       FilePageLabel.GetType[@label]=PilotFileTypes.tPhysicalVolumeRootPage];
    END;
```

```
PutRootFile: PUBLIC PROCEDURE [vol: LogicalVolume.Handle, type: File.Type, file: POINTER TO READONLY File.Capability] =
    BEGIN
    IF type NOT IN PilotFileTypes.PilotRootFileType THEN ERROR;
    vol.rootFileID[type] ← file.fID;
    -- This is a hack that has to be fixed if we change to multiple RootPageBuffers.....
    IF vol = activeVolume THEN SimpleSpace.ForceOut[logicalRootPageBuffer];
    END;

ReadAndCheckLogicalRootLabel: PROCEDURE [lvID: Volume.ID] RETURNS [BOOLEAN] =
    BEGIN
    label: FilePageLabel.Label;
    label ← LabelTransfer.ReadLabel[[LOOPHOLE[lvID], lvID, local[FALSE, FALSE, 1, PilotFileTypes.tLogicalVolumeRootPage]], 0,
      LogicalVolume.rootPageNumber];
    BEGIN OPEN label;
    -- could also check pad2...
    RETURN[fileID = LOOPHOLE[lvID] AND
        FilePageLabel.GetFilePage[@label] = 0 AND
        ~immutable AND ~temporary AND ~zeroSize AND pad1 = 0 AND
        FilePageLabel.GetType[@label] = PilotFileTypes.tLogicalVolumeRootPage];
    END; END;

ReadAndCheckMarkerPageLabel: PROCEDURE[pvID: BarePvID, page: PhysicalVolume.PageNumber] RETURNS [File.ID, BOOLEAN] =
    BEGIN
    label: FilePageLabel.Label ← LabelTransfer.ReadRootLabel[DiskChannel.GetAttributes[SubVolume.Find[[pvID],
      PhysicalVolume.rootPageNumber].subVolume.channel], page].label;
    BEGIN OPEN label;
     -- could also check pad2...
    RETURN[fileID, fileID # FilePageLabel.nullID AND
    FilePageLabel.GetFilePage[@label] = page AND
    ~immutable AND ~temporary AND ~zeroSize AND pad1 = 0 AND
    FilePageLabel.GetType[@label] = PilotFileTypes.tSubVolumeMarkerPage];
    END; END;

-- Set up caches for reading the physical root page
RegisterPhysicalInformation: PROCEDURE[pvID: BarePvID, drive: DiskChannel.Drive, driveSize: DiskChannel.DiskPageCount] =
    BEGIN
    rootPage: PhysicalVolume.PageNumber = PhysicalVolume.rootPageNumber;
    DiskChannel.SetDriveTag[drive, PhysicalVolume.IDCheckSum[[pvID]]];
    SubVolume.OnLine[[lvID: [pvID], lvSize: driveSize, lvPage: rootPage, pvPage: rootPage, nPages: driveSize],
      DiskChannel.Create[drive, SubVolume.completion]];
    FileCache.SetFile[[[pvID], [pvID], local[FALSE, FALSE, 1, PilotFileTypes.tPhysicalVolumeRootPage]], TRUE];
    FileCache.SetPageGroup[[pvID], [rootPage, rootPage, rootPage + 1], TRUE];
    END;

-- sets client root file
SetRootFile: PUBLIC PROCEDURE [volume: Volume.ID, file: File.Capability] =
    BEGIN
    SetRootFile1: PROCEDURE [vol: LogicalVolume.Handle] =
        BEGIN
        vol.clientRootFile ← file;
        vol.updateLogicalSubvolumeMarker ← TRUE;
        END;
    VolumeAccess[@volume, SetRootFile1, TRUE];
    END;

-- Temporary to give names to volumes
SetString: PROCEDURE [dest: LONG POINTER TO PACKED ARRAY [0..40) OF CHARACTER, pages: LONG CARDINAL, which:
{physical, logical}, ifLogical: Boot.VolumeType ← normal] RETURNS [size: CARDINAL [0..40]] =
    BEGIN
    i: CARDINAL;
    s: STRING;
    IF which = physical THEN s ← IF pages>5000 THEN "Sa4000"L ELSE "Diablo 31"L
    ELSE s ← IF pages > 5000 THEN SELECT ifLogical FROM
        normal => "RD0.pilotVolume"L, debugger => "RD0.Debugger"L, debuggerDebugger => "RD0.DebuggerDebugger"L,
        ENDCASE => "RD0.nonPilot"L
    ELSE s ← SELECT ifLogical FROM
        normal => "Normal"L, debugger => "Debugger"L, debuggerDebugger => "DebuggerDebugger"L, ENDCASE =>
        "nonPilot"L;
    size ← MIN[s.length, 40];
    FOR i IN [0..size) DO dest[i] ← s[i]; ENDLOOP;
    END;
```

```
-- This should duplicate code in FIleVolumeAccess, not call it !!
-- gives proc exclusive access to volume (optionally forces out rootPage)
VolumeAccess: PUBLIC PROCEDURE [pVID: POINTER TO READONLY Volume.ID,
proc: PROCEDURE [volume: LogicalVolume.Handle], forceOut: BOOLEAN] =
    BEGIN
    VolumeAccess1: FileAccessProc -- [vol, filePtr, op, newSizeP]-- = BEGIN proc[vol]; END;
    fileD: local FileInternal.Descriptor ← [FilePageLabel.nullID, pVID↑, local[,,,]];
    FileVolumeAccess[VolumeAccess1, @fileD, (IF forceOut THEN LogicalVolume.setVolumeAttributes ELSE
        LogicalVolume.noOp), NIL];
    END;

-- Entry Procedures

Close: PUBLIC ENTRY PROCEDURE[ID: Volume.ID] =
    BEGIN
    IF ID = Volume.nullID OR ~LogicalVolumeFind[@ID].found THEN RETURN WITH ERROR Unknown[ID];
    Activate[@ID, close];
    activeVolume.open ← FALSE;
    Deactivate[flush];
    END;

-- Gives proc exclusive access to volume. Someday, there should be multiple buffers, and the capability to have several volumes
cooking at once. Someday.
FileVolumeAccess: PUBLIC ENTRY PROCEDURE [proc: FileAccessProc, filePtr: POINTER TO local FileInternal.Descriptor, op:
LogicalVolume.Operations, newSize: POINTER TO READONLY File.PageCount] =
    BEGIN
    modifyVolume: BOOLEAN = Inline.BITAND[op, File.grow + File.shrink + File.delete + LogicalVolume.create] # LogicalVolume.noOp;
    IF op # LogicalVolume.noOp THEN FileCache.FlushFile[filePtr.fileID]; --flush if there will be side effects
    IF filePtr.volumeID = Volume.nullID OR ~LogicalVolumeFind[@filePtr.volumeID].found THEN GOTO Out;
    Activate[@filePtr.volumeID, IF modifyVolume THEN modify ELSE read ! UnOpened => GOTO Out];
    IF Inline.BITAND[op, File.grow + LogicalVolume.create] # LogicalVolume.noOp
        AND newSize↑ >= filePtr.size + activeVolume.freePageCount-Utilities.ShortCARDINAL[activeVolume.freePageCount]/16-
        5 THEN
            RETURN WITH ERROR InsufficientSpace;
    proc[activeVolume, filePtr, op, newSize];          --root page is in page buffer
    -- Update subvolume marker Pages if requested. Fit the logicalRootPage buffer gets "paged", it's update bit will be on,
    suggesting it is "correct" rathern than the marker pages. The bit is clearde (and page forced out) after all marker pages are
    updated.
    IF activeVolume.updateLogicalSubvolumeMarker THEN
        BEGIN
        UpdateLogicalSubvolumeMarkers[activeVolume];
        activeVolume.updateLogicalSubvolumeMarker ← FALSE;
        END;
    IF op = LogicalVolume.setVolumeAttributes OR (cautious AND modifyVolume) THEN Deactivate[forceout];
    EXITS
    Out => RETURN WITH ERROR Unknown[filePtr.volumeID];
    END;

GetNext: PUBLIC ENTRY PROCEDURE [volume: Volume.ID] RETURNS [Volume.ID] =
    BEGIN
    volumeFound: BOOLEAN ← volume = Volume.nullID;
    lv: LVTIndex;
    FOR lv IN LVTIndex DO
        IF volumeFound AND LVT[lv] # Volume.nullID THEN RETURN [LVT[lv]];
        IF LVT[lv] = volume THEN volumeFound ← TRUE;
        ENDLOOP;
    IF volumeFound THEN RETURN [Volume.nullID] ELSE RETURN WITH ERROR Unknown[volume];
    END;

-- Open the Logica! Volume. It expects to see only volumes of the Correct Type (debugger, client, debugdebug). The labels are
good at this point, certified by LogicalVolumeCheck
Open: PUBLIC ENTRY PROCEDURE[vID: Volume.ID] =
    BEGIN
    erase, scavenge: BOOLEAN;
    IF vID = Volume.nullID OR ~LogicalVolumeFind[@vID].found THEN RETURN WITH ERROR Unknown[vID];
    Activate[@vID, opening];
    erase ← PilotSwitches.switches.L = down OR LogicalVolume.Vam[activeVolume] = LogicalVolume.nullID OR
        LogicalVolume.Vfm[activeVolume] = LogicalVolume.nullID OR LogicalVolume.Free[activeVolume] = LogicalVolume.nullID;
    IF erase THEN
        BEGIN
        Runtime.CallDebugger["Proceed to erase logical volume"L];
```

```
        activeVolume.clientRootFile ← File.nullCapability;
        activeVolume.rootFileID ← LogicalVolume.nullRootFileIDs;
        LogicalVolume.SetVam[activeVolume, [System.GetUniversalID[]]];
        LogicalVolume.SetVfm[activeVolume, [System.GetUniversalID[]]];
        LogicalVolume.SetFree[activeVolume, [System.GetUniversalID[]]];
        UpdateLogicalSubvolumeMarkers[activeVolume];
        scavenge ← FALSE;
        END
    ELSE IF (scavenge ← activeVolume.changing OR PilotSwitches.switches.S = down) THEN
        Runtime.CallDebugger["Proceed to scavenge logical volume"L];
    activeVolume.changing ← erase OR scavenge;
    SimpleSpace.ForceOut[logicalRootPageBuffer];
    OpenVFile[activeVolume, PilotFileTypes.tVolumeAllocationMap];
    OpenVFile[activeVolume, PilotFileTypes.tVolumeFileMap];
    OpenVFile[activeVolume, PilotFileTypes.tFreePage];
    VolAllocMap.Open[activeVolume, erase, scavenge];
    VolFileMap.Open[activeVolume, erase, scavenge];
    Deactivate[forceout];
    END;

PhysicalVolumeOffLine: ENTRY PROCEDURE [pvID: Volume.ID --??--] =
    BEGIN
    -- Not yet implemented; flush the subvolumes associated with the physical volume
    -- FileCache.FlushFile[rootPageFileID]; + + currently doesn't work for pinned entries!!
    -- SubVolume.OffLine[physicalVID]; + + can't do without flushing file first
    END;

-- Examines physical volume on a drive which has just come on line.
PhysicalVolumeOnLine: ENTRY PROCEDURE [drive: DiskChannel.Drive] =
    BEGIN
    driveSize: DiskChannel.DiskPageCount = DiskChannel.GetDriveAttributes[drive].nPages;
    label: FilePageLabel.Label;
    labelStatus: LabelTransfer.LabelStatus ← invalid;
    good: BOOLEAN ← FALSE;
    rootID: BarePvID;
    i: CARDINAL;

    WHILE labelStatus # valid DO
        [label, labelStatus] ← LabelTransfer.ReadRootLabel[drive, PhysicalVolume.rootPageNumber];
        IF labelStatus # valid THEN Runtime.CallDebugger["Physical volume access error (Proceed when ready)"L];
        ENDLOOP;
    IF PilotSwitches.switches.E = up THEN [rootID, good] ← PhysicalRootLabelCheck[label];
    IF ~good THEN rootID ← System.GetUniversalID[];
    RegisterPhysicalInformation[rootID, drive, driveSize];
    IF good THEN good ← PhysicalRootPageCheck[rootID];
    IF good THEN RegisterLVFiles[rootID] ELSE InitPhysicalVolume[rootID, driveSize];
    -- Create/ check markers
    CheckAndInitMarkers[rootID, driveSize, ~good];
    PhysicalRootPageMap[rootID];
    FOR i IN [0..physicalVolume.subVolumeCount) DO
        LogicalVolumeCheck[physicalVolume.subVolumes[i].lvID]
        ENDLOOP;
    PhysicalRootPageUnmap[];
    END;

-- Internal Procedures

-- Provides access to volume files of one volume at a time
Activate: INTERNAL PROCEDURE [pvID: POINTER TO READONLY Volume.ID, access: {read, modify, opening, close} ← read] =
    BEGIN
    IF activeVID # pvID↑ THEN
        BEGIN
        IF ~LogicalVolumeFind[pvID].found THEN
            ERROR VolImplError[notFoundVolumeInActivate];
        Deactivate[flush];
        LogicalRootPageMap[pvID↑];
        activeVID ← pvID↑;
        END;
    IF ~activeVolume.open THEN
        IF access = opening THEN activeVolume.open ← TRUE
        ELSE IF access # close THEN ERROR UnOpened[pvID↑];
    -- The next two are basically Noops....
```

```
    VolAllocMap.Open[activeVolume, FALSE, FALSE];
    VolFileMap.Open[activeVolume, FALSE, FALSE];
    IF access = modify AND ~activeVolume.changing THEN
        BEGIN
        activeVolume.changing ← TRUE;
        SimpleSpace.ForceOut[logicalRootPageBuffer];
        END;
    END;

-- Check that all the volume submarkers are consistant.  If not, ask confirmation and write new ones.  IF justInit, then only write
new ones.  Register the Subvolume Marker file as a side effect
CheckAndInitMarkers: INTERNAL PROCEDURE[pvID: BarePvID, driveSize: DiskChannel.DiskPageCount, justInit: BOOLEAN] =
    BEGIN
    pRootPage: PhysicalVolume.PageNumber = PhysicalVolume.rootPageNumber;
    i: CARDINAL;
    init: BOOLEAN ← FALSE;
    markerFile: FileInternal.Descriptor ← [File.nullID, [pvID], local[FALSE, FALSE, driveSize,
    PilotFileTypes.tSubVolumeMarkerPage]];

    -- First, see if there is a consistant set of labels out there
    IF ~justInit THEN
        BEGIN
        tmpID: File.ID;
        ok: BOOLEAN;
        PhysicalRootPageMap[pvID];
        FOR i IN [0..physicalVolume.subVolumeCount) DO
            OPEN sv: physicalVolume.subVolumes[i];
            [tmpID, ok] ← ReadAndCheckMarkerPageLabel[pvID, sv.pvPage + sv.nPages];
            IF ~ok THEN BEGIN init ← TRUE; EXIT; END;
            IF i = 0 THEN markerFile.fileID ← tmpID
            ELSE IF markerFile.fileID # tmpID THEN BEGIN init ← TRUE; EXIT; END;
            ENDLOOP;
        PhysicalRootPageUnmap[];
        END;
    -- Need to generate a label here
    IF init OR justInit THEN markerFile.fileID ← [System.GetUniversalID[]];
    FileCache.SetFile[markerFile, TRUE];
    FileCache.SetPageGroup[markerFile.fileID, [pRootPage + 1, pRootPage + 1, driveSize], TRUE];
    -- Check the data now
    IF ~justInit AND ~init THEN
        BEGIN
        PhysicalRootPageMap[pvID];
        FOR i IN [0..physicalVolume.subVolumeCount) WHILE init DO
            OPEN sv: physicalVolume.subVolumes[i];
            MarkerPageMap[markerFile.fileID, sv.pvPage + sv.nPages];
            init ← marker.logical.seal = LogicalVolume.LSMSeal AND
              marker.logical.version = LogicalVolume.LSMCurrentVersion AND marker.physical.seal = PhysicalVolume.PSMSeal AND
              marker.physical.version = PhysicalVolume.PSMCurrentVersion;
            MarkerPageUnmap[];
            ENDLOOP;
        PhysicalRootPageUnmap[];
        END;
    IF init THEN Runtime.CallDebugger["Proceed to write Subvolume Marker Pages"L];
    IF init OR justInit THEN
    -- This code Doesn't work for MultiSubvolume Logical Volumes if the rootpage is offline or unknown
        BEGIN
        page: LONG CARDINAL;
        Deactivate[flush];
        PhysicalRootPageMap[pvID];
        FOR i IN [0..physicalVolume.subVolumeCount) DO
            OPEN sv: physicalVolume.subVolumes[i], volP: activeVolume, pv: physicalVolume;
            page ← sv.pvPage + sv.nPages;
            LabelTransfer.WriteLabels[file: markerFile, pageGroup: [page, page, page + 1]];
            LogicalRootPageMap[sv.lvID];
            MarkerPageMap[markerFile.fileID, page];
            marker↑ ← [
                physical: [pvID: [pvID], label: pv.label, bootingInfo:  pv.bootingInfo, labelLength: pv.labelLength,
                  svNumber: i, descriptor:  sv],
                logical: [labelLength: volP.labelLength, type: volP.type, label: volP.label, bootingInfo: volP.bootingInfo,
                  clientRootFile: volP.clientRootFile]];
            MarkerPageUnmap[];
            LogicalRootPageUnmap[];
```

```
            ENDLOOP;
        PhysicalRootPageUnmap[];
        END;
    END;

-- Deactivates the active volume
Deactivate: INTERNAL PROCEDURE [termination: {forceout, flush}] =
    BEGIN
    IF activeVID = Volume.nullID THEN RETURN;
    VolFileMap.Close[activeVolume, termination = flush];
    VolAllocMap.Close[activeVolume, termination = flush];
    IF activeVolume.changing THEN activeVolume.changing ← FALSE;
    IF termination = flush THEN LogicalRootPageUnmap[] ELSE SimpleSpace.ForceOut[logicalRootPageBuffer];
    END;

-- Make the debugger volume swat + swatee pages (800 + 800) bigger
InitPhysicalVolume: INTERNAL PROCEDURE[pvID: BarePvID, driveSize: DiskChannel.DiskPageCount] =
    BEGIN
    i: CARDINAL;
    pRootPage: PhysicalVolume.PageNumber = PhysicalVolume.rootPageNumber;
    lRootPage: LogicalVolume.PageNumber = LogicalVolume.rootPageNumber;
    userDriveSize: DiskChannel.DiskPageCount = driveSize-3;
    clientSize: PhysicalVolume.PageCount = (userDriveSize-1600)/2;
    debuggerSize: PhysicalVolume.PageCount = userDriveSize-clientSize;
    clientStart: PhysicalVolume.PageNumber = pRootPage + 1;
    clientMarker: PhysicalVolume.PageNumber = clientStart + clientSize;
    debuggerStart: PhysicalVolume.PageNumber = clientMarker + 1;
    debuggerMarker: PhysicalVolume.PageNumber = debuggerStart + debuggerSize;
    bitsPerPage: CARDINAL = Environment.bitsPerWord*Environment.wordsPerPage;

    Runtime.CallDebugger["Proceed to erase and initialize physical volume"L];
    Deactivate[flush];
    LabelTransfer.WriteLabels[
        file: [[pvID], [pvID], local[FALSE, FALSE, 1, PilotFileTypes.tPhysicalVolumeRootPage]],
        pageGroup: [pRootPage, pRootPage, pRootPage + 1]];
    PhysicalRootPageMap[pvID];
    physicalVolume↑ ← [pvID: [pvID], subVolumeCount: 2,
            subVolumes: [[[System.GetUniversalID[]], clientSize, 0, clientStart, clientSize],
            [[System.GetUniversalID[]], debuggerSize, 0, debuggerStart, debuggerSize],,,,]];
    physicalVolume.labelLength ← SetString[@physicalVolume.label, driveSize, physical];
    PhysicalRootPageUnmap[];
    -- We have a good PV Root page.  Set up Mechanisms for Accessing LVpages
    RegisterLVFiles[pvID];
    -- Create LV Root Pages, using data in PV Page
    PhysicalRootPageMap[pvID];
    FOR i IN [0..physicalVolume.subVolumeCount) DO
        OPEN sv: physicalVolume.subVolumes[i];
        LabelTransfer.WriteLabels[
            file: [LOOPHOLE[sv.lvID], sv.lvID, local[FALSE, FALSE, 1, PilotFileTypes.tLogicalVolumeRootPage]],
            pageGroup: [lRootPage, lRootPage, lRootPage + 1]];
        LogicalRootPageMap[sv.lvID];
        activeVolume↑ ← [volumeSize: sv.lvSize, vID: sv.lvID, type: IF i = 0 THEN normal ELSE debugger];
        activeVolume.labelLength ← SetString[@activeVolume.label, driveSize, logical, activeVolume.type];
        -- Set VAMStart so that there are now bad pages in the run.
        activeVolume.vfmStart ← activeVolume.vamStart + (activeVolume.volumeSize + bitsPerPage)/bitsPerPage;
        -- The vam/vfm/free ID's are set latter in OPEN. This is a hack
        LogicalRootPageUnmap[];
        ENDLOOP;
    PhysicalRootPageUnmap[];
    END;

-- I need to think about taking multiPack volume off line somemore.  Caches have been set up so mapping is cool
LogicalVolumeCheck: INTERNAL PROCEDURE [vID: Volume.ID] =
    BEGIN
    found, good: BOOLEAN;
    size: Volume.PageCount;
    svH: SubVolume.Handle;

    Deactivate[flush];
    [good, svH] ← SubVolume.Find[vID, LogicalVolume.rootPageNumber];
    IF good THEN
        IF ~(good ← ReadAndCheckLogicalRootLabel[vID]) THEN
```

```
            Runtime.CallDebugger["bad LVRootPage label; proceed to ignore this logical volume"L];
    IF good THEN
        BEGIN
        LogicalRootPageMap[vID];
        good ← activeVolume.seal = LogicalVolume.seal;
        IF ~good THEN Runtime.CallDebugger["???; bad LVRootPage; proceed to ignore this logical volume"L];
        good ← good AND activeVolume.version = LogicalVolume.currentVersion AND activeVolume.type = clientType;
        size ← activeVolume.volumeSize;
        IF IsMyDebuggerVolume[activeVolume] THEN
         -- set the debugger Boot Pointers back in Pilot Control
            BEGIN
            IF pLVBootFiles # NIL THEN pLVBootFiles↑ ← activeVolume.bootingInfo;
            debuggerDevice ← DiskChannel.GetAttributes[svH.channel].drive;
            debuggerIDsSet ← TRUE;
            END;
        LogicalRootPageUnmap[];
        END;
    WHILE good AND size # svH.lvPage + svH.nPages DO
        [good, svH] ← SubVolume.Find[vID, svH.lvPage + svH.nPages];
        ENDLOOP;
    found ← LogicalVolumeFind[@vID].found;
    IF good AND ~found THEN LogicalVolumeOnLine[vID];
    IF found AND ~good THEN LogicalVolumeOffLine[vID]; -- can this really happen???
    END;

-- Logical Volume Table Keepers (lists logical volumes which are on-line)

LVTHandle: TYPE = --LONG-- POINTER TO Volume.ID;
LVT: ARRAY LVTIndex OF Volume.ID ← ALL[Volume.nullID];
LVTIndex: TYPE = [0..maxLVs);
maxLVs: CARDINAL = 8; -- Maximum logical volumes allowed on-line at once

LogicalVolumeFind: INTERNAL PROCEDURE[vID: POINTER TO READONLY Volume.ID] RETURNS [lvtH: LVTHandle, found: BOOLEAN]
=
    BEGIN
    lv: LVTIndex;
    FOR lv IN LVTIndex DO IF LVT[lv] = vID↑ THEN RETURN[@LVT[lv], TRUE]; ENDLOOP;
    RETURN [NIL, FALSE];
    END;

LogicalVolumeOffLine: INTERNAL PROCEDURE[vID: Volume.ID] =
    BEGIN
    lv: LVTIndex;
    FOR lv IN LVTIndex DO
        IF LVT[lv] = vID THEN
            BEGIN
            LVT[lv] ← Volume.nullID;
            -- Deactivate?? What else??
            -- FileCache.FlushFile[rootPageFileID];          + + currently doesn't work for pinned entries!!
            -- SubVolume.OffLine[vID, drive];          + + can't do without flushing file first
            RETURN;
            END;
        ENDLOOP;
    ERROR VolImplError[notFoundInOffLine];
    END;

LogicalVolumeOnLine: INTERNAL PROCEDURE[vID: Volume.ID] =
    BEGIN
    lv: LVTIndex;
    FOR lv IN LVTIndex DO IF LVT[lv] = Volume.nullID THEN BEGIN LVT[lv] ← vID; RETURN; END; ENDLOOP;
    ERROR VolImplError[lvTableFull];
    END;

LogicalRootPageMap: INTERNAL PROCEDURE[ID: Volume.ID] = INLINE
    BEGIN
    SimpleSpace.Map[logicalRootPageBuffer, [[LOOPHOLE[ID], FileInternal.maxPermissions], LogicalVolume.rootPageNumber],
    FALSE];
    END;

LogicalRootPageUnmap: INTERNAL PROCEDURE =
    INLINE BEGIN SimpleSpace.Unmap[logicalRootPageBuffer]; activeVID ← Volume.nullID; END;
```

```
MarkerPageMap: INTERNAL PROCEDURE[ID: File.ID, page: LONG CARDINAL] =
    BEGIN
    SimpleSpace.Map[markerPageBuffer, [[ID, FileInternal.maxPermissions], page], FALSE];
    END;

MarkerPageUnmap: INTERNAL PROCEDURE = BEGIN SimpleSpace.Unmap[markerPageBuffer]; END;

-- Loads Filer's caches with descriptors for a volume file on the active volume.  SHould make check for  existance before adding to caches.
OpenVFile: INTERNAL PROCEDURE [vol: LogicalVolume.Handle, type: File.Type] =
    BEGIN OPEN file: vol.rootFileID[type];
    FileCache.SetFile[[file, vol.vID, local[FALSE, FALSE, vol.volumeSize, type]], TRUE];
    FileCache.SetPageGroup[file, [0, 0, vol.volumeSize], TRUE];
    END;

PhysicalRootPageCheck: INTERNAL PROCEDURE[ID: BarePvID] RETURNS[good: BOOLEAN] =
    BEGIN
    PhysicalRootPageMap[ID];
    good ← physicalVolume.seal = PhysicalVolume.seal AND
    physicalVolume.version = PhysicalVolume.currentVersion AND
    physicalVolume.pvID = PhysicalVolume.ID[ID];
    PhysicalRootPageUnmap[];
    END;

PhysicalRootPageMap: INTERNAL PROCEDURE[ID: BarePvID] =
    BEGIN SimpleSpace.Map[physicalRootPageBuffer, [[ID], FileInternal.maxPermissions], PhysicalVolume.rootPageNumber],
    FALSE]; END;

PhysicalRootPageUnmap: INTERNAL PROCEDURE = BEGIN SimpleSpace.Unmap[physicalRootPageBuffer]; END;

-- make the In-core Subvolume entry for the Physical subvolumes.  If its the root subvolume, create the Root-volume file.
RegisterLVFiles: INTERNAL PROCEDURE[pvID: BarePvID] =
    BEGIN
    i: CARDINAL;
    lRootPage: LogicalVolume.PageNumber = LogicalVolume.rootPageNumber;
    PhysicalRootPageMap[pvID];
    FOR i IN [0..physicalVolume.subVolumeCount) DO
        OPEN sv: physicalVolume.subVolumes[i];
        SubVolume.OnLine[sv, SubVolume.Find[[pvID], PhysicalVolume.rootPageNumber].subVolume.channel];
        IF sv.lvPage = lRootPage THEN
            BEGIN
            FileCache.SetFile[[LOOPHOLE[sv.lvID], sv.lvID, local[FALSE, FALSE, 1, PilotFileTypes.tLogicalVolumeRootPage]],
              TRUE];
            FileCache.SetPageGroup[LOOPHOLE[sv.lvID], [lRootPage, lRootPage, lRootPage + 1], TRUE];
            END;
        ENDLOOP;
    PhysicalRootPageUnmap[];
    END;

-- Write new info on subvolume Marker pages for the various disks.  Could be smarter in not reading labels/keeping cache.  We do not have to set pageGroups/File Ptrs, since they we pinned when we opened the volume.
UpdateLogicalSubvolumeMarkers: INTERNAL PROCEDURE [vol: LogicalVolume.Handle] =
    BEGIN
    svH: SubVolume.Handle ← NIL;
    page: LONG CARDINAL;
    WHILE (svH ← SubVolume.GetNext[svH]) # NIL DO
        IF svH.lvID # vol.vID THEN LOOP;
        page ← svH.pvPage + svH.nPages;
        MarkerPageMap[LabelTransfer.ReadRootLabel[DiskChannel.GetAttributes[svH.channel], page].label.fileID, page];
        marker.logical.bootingInfo ← vol.bootingInfo;
        marker.logical.clientRootFile ← vol.clientRootFile;
        MarkerPageUnmap[];
        ENDLOOP;
    END;

InitDisks: PROCEDURE =
    BEGIN OPEN DiskChannel;
    altoDrive: Drive = [0];
    drive: Drive;
    systemDeviceType: OISProcessorFace.DeviceType ← IF PilotSwitches.switches.X = down THEN sa4000 ELSE mod31;
```

```
    volumeID: Volume.ID ← Volume.nullID;
    -- For now find the first volume of the correct type on the "SystemDrive". Eventually we will set SystemID to the booted-from
    logical volume, and then put all volumes of the correct type on line. Code also assumes we have only one pilot drive of each
    type

    drive ← nullDrive;
    WHILE (drive ← GetNextDrive[drive]) # nullDrive DO
        IF drive = altoDrive THEN LOOP;
        IF GetDriveAttributes[drive].deviceType = systemDeviceType THEN
            BEGIN PhysicalVolumeOnLine[drive]; systemID ← GetNext[Volume.nullID]; EXIT; END;
        ENDLOOP;
    IF systemID = Volume.nullID THEN Runtime.CallDebugger["No Logical Volume on System Drive "L];
    drive ← nullDrive;
    IF PilotSwitches.switches.Z = down THEN WHILE (drive ← GetNextDrive[drive]) # nullDrive DO
        IF drive = altoDrive THEN LOOP;
        IF GetDriveAttributes[drive].deviceType # systemDeviceType THEN PhysicalVolumeOnLine[drive];
        ENDLOOP;
    WHILE (volumeID ← GetNext[volumeID]) # Volume.nullID DO Open[volumeID]; ENDLOOP;
    END;

-- Set the types temporarally from information given in bootFile.device. Eventually, this will evlove into real code in the first loop
below that looks for the drive with bootfile.device
debuggerDevice ← StoragePrograms.nullDevice;
clientType ← SELECT bootFile.device FROM 0 => normal, 1 => debugger, ENDCASE => debuggerDebugger;
IF PilotSwitches.switches.U = up THEN InitDisks[];
debuggerIDsSet ← TRUE;                    -- We will never ever set them from here on....
RETURN;
END.
```

LOG

Time: June 19, 1978 1:05 PM By: Purcell Action: Created from volAllocMapImpl.mesa
Time: June 22, 1978 4:36 PM By: Purcell Action: split up volume.init
Time: July 27, 1978 11:45 AM By: Purcell Action: fix bug in rootinit, touched unmapped simple space rootbuffer
Time: August 6, 1978 1:21 AM By: Redell/Purcell Action: Added beginnings of LOOPHOLEs needed to eliminate compilation dependencies of interfaces on RigidDisk: (LOOPHOLEd channel in RootInit)
Time: August 7, 1978 3:25 PM By: Redell/Purcell Action: Changed drive number of Pilot drive from 0 to 1.
Time: August 14, 1978 11:51 AM By: Purcell Action: moved 2 above changes by hand and removed logicalVolume.channel uses.
Time: September 26, 1978 3:13 PM By: Purcell Action: CR 20.23 & 20.24 resolve errors and signals
Time: September 28, 1978 2:54 PM By: Purcell Action: CR 20.35 improve serialization
Time: October 19, 1978 10:11 AM By: Purcell Action: CR 65, 76 Volume openning and overflow; and weed out FileType dependancies
Time: December 11, 1978 5:08 PM By: Redell Action: quadword-align label buffer in VolumeOpen.
Time: February 19, 1979 5:48 PM By: Redell Action: Convert to new DiskChannel interface and Mesa 5.0.
Time: March 26, 1979 8:45 PM By: Redell Action: Fixed bug: interdependent non-constant declarations out of order.
Time: March 30, 1979 5:40 PM By: Redell Action: Removed out-of-line procedure SystemID and changed systemVID to systemID for export.
Time: April 2, 1979 10:45 AM By: McJones Action: Made systemID PUBLIC for export
Time: July 26, 1979 11:12 AM By: Forrest Action: Changed module structure to suggested layout for monitor. Added Debugger VOlumes. Made work with new Logical Volume, subvolume, PhysicalRootPage. Added GetLabelString
Time: July 31, 1979 12:02 PM By: Forrest Action: Hacked to make work for debugger/client volumes (blech)
Time: August 1, 1979 10:47 AM By: Forrest Action: Changed to export/implement new FMProgram Interface
Time: August 10, 1979 4:42 PM By: Forrest Action: Fixed GetLabelString to match new volume.
Time: August 16, 1979 10:13 AM By: McJones Action: Changed to match new DiskChannel, PhysicalVolume, SubVolume
Time: August 27, 1979 10:32 AM By: Forrest Action: fixed getLabelString
Time: August 27, 1979 11:15 AM By: Gobbel Action: Hack for initial SA4000 debugging
*Time: August 31, 1979 11:53 AM By: Forrest Action: Fix to use new LogicalVolume, Vol*map, pilotFileTypes*
Time: September 3, 1979 4:12 PM By: Forrest Action: Correct types of VolFileAccess, PutRootFile; use LogicalVolume.[Set]Free/Vam/Vfm
Time: September 17, 1979 4:50 PM By: Forrest Action: Change to use new RootReadLabel
Time: September 18, 1979 7:17 PM By: Forrest Action: change to use Boot; write out Subvolume marker pages (with nothing in them); add X&Y switch stuff.
Time: September 25, 1979 10:02 AM By: Forrest Action: Fixed L key Bug; moved test for Utility Pilot..
Time: September 25, 1979 4:01 PM By: Forrest Action: Fixed Bug introduced by previous fix....

```
-- Swapper.config (last edited by: McJones on: August 10, 1979  2:49 PM)

PACK
    CachedRegionImpl,
    CachedSpaceImpl,
    MStoreImpl,
    PageFaultImpl,
    --UNTIL IT FINDS A BETTER HOME-- SimpleSpaceImpl,
    SwapBufferImpl,
    SwapperControl,
    SwapperExceptionImpl,
    SwapTaskImpl;

Swapper: CONFIGURATION
    IMPORTS FilePageTransfer, Process, RuntimeInternal, Utilities
    EXPORTS CachedRegion, CachedSpace,
--UNTIL IT FINDS A BETTER HOME-- SimpleSpace,
        SpecialSpace, StoragePrograms, SwapperException
    CONTROL SwapperControl =

BEGIN

CachedRegionImpl;

CachedSpaceImpl;

MStoreImpl;

PageFaultImpl;

--UNTIL IT FINDS A BETTER HOME--SimpleSpaceImpl;

SwapBufferImpl;

SwapperControl;

SwapperExceptionImpl;

SwapTaskImpl;

END.
```

LOG

Time: June 6, 1978 3:05 PM By: McJones Action: Created file
Time: June 21, 1978 9:54 AM By: McJones Action: Added SimpleSpaceImpl
Time: July 18, 1978 10:55 AM By: McJones Action: Added Process to IMPORTS
Time: July 31, 1978 5:54 PM By: McJones Action: Added RuntimeInternal to IMPORTS
Time: August 1, 1978 6:21 PM By: McJones Action: Added Utilities to IMPORTS
Time: August 3, 1978 1:20 PM By: McJones Action: Added code packing
Time: August 10, 1979 2:49 PM By: McJones Action: Added SpecialSpace to EXPORTS

```
-- Things to consider:
-- 1) Analyze dynamic usage of MStore.Relocate for possible inline cases
-- 2) Move allocation of patch table to PilotControl (so it can be used during Pilot initialization)

DIRECTORY
    CachedRegion: FROM "CachedRegion",
    CachedSpace: FROM "CachedSpace" USING [Desc, Get, Handle],
    Environment: FROM "Environment" USING [Word, wordsPerPage],
    File: FROM "File" USING [PageNumber],
    FilePageTransfer: FROM "FilePageTransfer" USING [Initiate, Wait],
    InlineDefs: FROM "InlineDefs" USING [LongCOPY],
    MStore: FROM "MStore" USING [Allocate, Deallocate, Promise, Relocate],
    PageFault: FROM "PageFault" USING [Restart],
    PageMap: FROM "PageMap" USING [Flags, flagsAll, flagsNone, flagsNotReferenced, flagsReferenced,
        flagsWriteProtected],
    Process: FROM "Process" USING [Detach],
    RuntimeInternal: FROM "RuntimeInternal" USING [WorryCallDebugger],
    SwapBuffer: FROM "SwapBuffer" USING [Allocate, Deallocate],
    SwapperPrograms: FROM "SwapperPrograms",
    SwapTask: FROM "SwapTask" USING [Advance, Fork, State],
    Utilities: FROM "Utilities" USING [LongMove, LongPointerFromPage, ShortCARDINAL],
    VM: FROM "VM" USING [Interval, PageCount, PageNumber, PageOffset],
    VMMapLog: FROM "VMMapLog" USING [Descriptor, PatchTable, PatchTableEntry, PatchTableEntryBasePointer,
        PatchTableEntryPointer];

-- NOTE: No frame allocation traps may be taken within the following monitor!

CachedRegionImpl: MONITOR [pMapLogDesc: LONG POINTER]
    IMPORTS CachedSpace, FilePageTransfer, InlineDefs, MStore, PageFault, Process, RuntimeInternal, SwapBuffer,
        SwapTask, Utilities
    EXPORTS CachedRegion, SwapperPrograms =

BEGIN OPEN CachedRegion;

inSwappableCold: InSwappable ← inSwappableColdest; -- for age Operation

BaseDesc: TYPE = LONG BASE POINTER TO RECORD [UNSPECIFIED];
PDesc: TYPE = BaseDesc RELATIVE ORDERED POINTER [0..177777B] TO Desc;

IDesc: TYPE = [0..220); -- 50 more than Pilot 2.0
rgDesc: ARRAY IDesc OF Desc;

baseDesc: BaseDesc = LOOPHOLE[LONG[@rgDesc[FIRST[IDesc]]]]; -- base of cache
pDescFirst: PDesc = FIRST[PDesc]; -- offset of first entry of cache
pDescMax: PDesc = pDescFirst + SIZE[Desc]*(LAST[IDesc]-FIRST[IDesc]); -- offset of last possible entry of cache
pDescLast: PDesc; -- offset of current last entry of cache
pDescCommutator: PDesc; -- offset of entry which replacement algorithm will consider next
pDescMru: PDesc; -- offset of entry last returned by FirstNotBefore

checkIn: CONDITION;

IPTE: TYPE = [0..50);

patchTable: MACHINE DEPENDENT RECORD [
    header: VMMapLog.PatchTable,
    body: ARRAY IPTE OF VMMapLog.PatchTableEntry];

Apply: PUBLIC PROCEDURE [pageMember: VM.PageNumber, operation: Operation]
        RETURNS [outcome: Outcome, pageNext: VM.PageNumber] =
    BEGIN
    region: Desc ← CheckOut[pageMember, operation.ifCheckedOut];
    forked: BOOLEAN ← FALSE; -- as soon as forked is set to TRUE, region must be considered read-only by this process

    -- Variables declared before here may be referenced globally by ForkRead and ForkWrite
```

```
ForkRead: PROCEDURE [stateNext: State, andDirty: BOOLEAN, oldSpace: POINTER TO CachedSpace.Desc] =
    BEGIN
    buffer, activeBuffer: VM.Interval;
    flags: PageMap.Flags;
    space: CachedSpace.Desc;
    offsetInSpace: VM.PageOffset; -- offset of region in space
    countMapped, oldCountMapped, minCountMapped, growth, shrinkage: VM.PageCount;
    in: BOOLEAN = region.state IN In;
    out: BOOLEAN = region.state = outAlive;
    dead: BOOLEAN = region.state = outDead;
    CachedSpace.Get[@space, CachedSpace.Handle[level: region.levelMapped, page: region.interval.page]];
    SELECT space.state FROM
        missing => BEGIN outcome ← [spaceDMissing[region.levelMapped]]; RETURN END;
        unmapped => ERROR; -- region/space inconsistency
        beingRemapped => IF region.beingRemapped THEN RETURN;
        ENDCASE;
    flags ← [space.writeProtected, ~space.writeProtected AND andDirty, FALSE];
    region.state ← stateNext;
    offsetInSpace ← region.interval.page-space.interval.page;
    countMapped ← MIN[region.interval.count, space.countMapped-MIN[space.countMapped, offsetInSpace]];
    oldCountMapped ← IF oldSpace = NIL THEN 0
        ELSE MIN[region.interval.count, oldSpace.countMapped-MIN[oldSpace.countMapped, offsetInSpace]];
    minCountMapped ← MIN[countMapped, oldCountMapped];
    growth ← countMapped-minCountMapped; -- growth/shrinkage of mapped subregion due to Remap
    shrinkage ← oldCountMapped-minCountMapped; -- (only one can be non-zero)
    buffer ← SwapBuffer.Allocate[region.interval.count];
    activeBuffer ← IF in THEN VM.Interval[buffer.page + oldCountMapped, growth] ELSE VM.Interval[buffer.page,
            countMapped];
    [] ← MStore.Allocate[interval: activeBuffer, mayWait: TRUE];
    IF dead THEN -- zero out active buffer area and snap into region
        BEGIN
        pFirst: LONG POINTER TO Environment.Word;
        offset: VM.PageOffset;
        region.dDirty ← TRUE;
        FOR offset IN [0..activeBuffer.count) DO
            pFirst ← Utilities.LongPointerFromPage[activeBuffer.page + offset];
            pFirst↑ ← 0;
            InlineDefs.LongCOPY[from: pFirst, nwords: Environment.wordsPerPage-1, to: pFirst + 1]
            ENDLOOP;
        [] ← MStore.Relocate[buffer, region.interval.page, PageMap.flagsNone, flags]
        END
    ELSE
        BEGIN
        filePage: File.PageNumber;
        forked ← SwapTask.Fork[region, flags, buffer.page, activeBuffer.count];
        IF out AND oldCountMapped>0 THEN -- Remap only: read in leading pages from old file
            BEGIN
            filePage ← oldSpace.window.base + offsetInSpace;
            FilePageTransfer.Initiate[[
                file: oldSpace.window.file,
                filePage: filePage,
                memoryPage: buffer.page,
                count: minCountMapped,
                opSpecific: read[priorityPage: filePage + (pageMember-region.interval.page)]]]
            END
        ELSE IF in THEN -- Remap only: truncate any trailing pages no longer in mapped sub-region
            BEGIN
            [] ← MStore.Relocate[region.interval, region.interval.page, PageMap.flagsReferenced, flags];
            MStore.Deallocate[interval: VM.Interval[page: region.interval.page + countMapped, count: shrinkage],
                    promised: FALSE]
            END;
        -- All cases: read in appropriate pages from current file
        --   (Remap: these are new pages due to growth of mapped sub-region)
        filePage ← space.window.base + offsetInSpace + oldCountMapped;
        FilePageTransfer.Initiate[[
            file: space.window.file,
            filePage: filePage,
            memoryPage: buffer.page + oldCountMapped,
            count: growth,
            opSpecific: read[priorityPage: filePage + (pageMember-region.interval.page)]]]
        END;
    IF ~forked THEN SwapBuffer.Deallocate[buffer]
```

```
END;
```

```
ForkWrite: PROCEDURE [stateNext: State, andWriteProtect: BOOLEAN] =
    BEGIN
    -- Assume: region.state IN InSwappable
    buffer: VM.Interval = SwapBuffer.Allocate[region.interval.count];
    flags: PageMap.Flags ← MStore.Relocate[region.interval, buffer.page, PageMap.flagsNone,
            PageMap.flagsWriteProtected].flags;
    flags.writeProtected ← flags.writeProtected OR andWriteProtect;
    BEGIN
    IF flags.dirty THEN
        BEGIN
        space: CachedSpace.Desc;
        offsetInSpace: VM.PageOffset; -- offset of region in space
        countMapped: VM.PageCount;
        CachedSpace.Get[@space, CachedSpace.Handle[level: region.levelMapped, page: region.interval.page]];
        SELECT space.state FROM
            missing => BEGIN outcome ← [spaceDMissing[region.levelMapped]]; GO TO Undo END;
            unmapped => ERROR;
            beingRemapped => region.beingRemapped ← FALSE;
            ENDCASE;
        offsetInSpace ← region.interval.page-space.interval.page;
        countMapped ← MIN[region.interval.count, space.countMapped-MIN[space.countMapped, offsetInSpace]];
        IF (region.state ← stateNext) ~IN In THEN
            MStore.Promise[countMapped];
        forked ← SwapTask.Fork[region, [flags.writeProtected, FALSE, flags.referenced], buffer.page, countMapped];
        FilePageTransfer.Initiate[[
            file: space.window.file,
            filePage: space.window.base + offsetInSpace,
            memoryPage: buffer.page,
            count: countMapped,
            opSpecific: write[]]]
        END
    ELSE -- ~dirty
        BEGIN
        IF (region.state ← stateNext) IN In THEN
            GO TO Undo;
        MStore.Deallocate[interval: buffer, promised: FALSE]
        END;
    EXITS Undo => [] ← MStore.Relocate[buffer, region.interval.page, PageMap.flagsNone, flags]
    END;
    IF ~forked THEN
        SwapBuffer.Deallocate[buffer]
    END;
```

```
-- Main body of CachedRegion.Apply
outcome ← [ok[]];
pageNext ← region.interval.page + region.interval.count;
SELECT region.state FROM
    = checkedOut =>
        NULL;
    = missing =>
        IF operation.ifMissing = report THEN
            outcome ← [regionDMissing[]]
        ELSE -- operation.ifMissing = skip --
            NULL;
    IN [unmapped..inPinned] =>
        BEGIN OPEN region;
        error: BOOLEAN ← FALSE;
        WITH operation SELECT FROM
            activate --[why]-- =>
                IF state IN Out THEN
                    ForkRead[stateNext: inSwappableWarmest, andDirty: FALSE, oldSpace: NIL]
                ELSE IF why = pagefault AND (state = unmapped OR (state IN InSwappable AND
                        MStore.Relocate[[pageMember, 1], pageMember, PageMap.flagsAll,
                        PageMap.flagsNone].anyVacant)) THEN
                    error ← TRUE; -- this was an address fault
            age =>
            IF state IN InSwappable THEN
                    BEGIN
                    IF MStore.Relocate[interval, interval.page, PageMap.flagsNotReferenced,
                            PageMap.flagsNone].flags.referenced THEN
                        state ← inSwappableWarmest
                    ELSE IF state>inSwappableCold THEN
                        state ← LOOPHOLE[LOOPHOLE[state, CARDINAL]-1] -- for want of PREDECESSOR operation
                    ELSE -- state = inSwappableColdest --
                        ForkWrite[stateNext: outAlive, andWriteProtect: ]
                    END;
            clean --[andWriteProcted]-- =>
                IF state IN InSwappable THEN
                    ForkWrite[stateNext: state, andWriteProtect: andWriteProtect];
            deactivate =>
                IF state IN InSwappable THEN
                    ForkWrite[stateNext: outAlive, andWriteProtect: ];
            flush =>
                IF dDirty THEN
                    outcome ← [regionDDirty[]]
                ELSE IF state IN In THEN
                    ForkWrite[stateNext: missing, andWriteProtect: ]
                ELSE -- state = unmapped OR state IN Out --
                    state ← missing;
            get --[andResetDDirty, pDescResult] -- =>
                BEGIN
                pDescResult↑ ← region;
                IF andResetDDirty THEN
                    dDirty ← FALSE
                END;
            kill =>
                IF state = outAlive THEN
                    BEGIN state ← outDead; dDirty ← TRUE END
                ELSE IF state IN InSwappable THEN
                    BEGIN
                    IF MStore.Relocate[VM.Interval[interval.page, 1], interval.page, PageMap.flagsAll,
                            PageMap.flagsNone].flags.writeProtected THEN
                        error ← TRUE
                    ELSE
                        BEGIN
                        state ← outDead; dDirty ← TRUE; MStore.Deallocate[interval: interval, promised: FALSE]
                        END
                    END;
            map --[level]-- =>
                IF state = unmapped THEN
                    BEGIN
                    state ← outAlive; levelMapped ← level; beingRemapped ← FALSE; dDirty ← TRUE
                    END
                ELSE -- state IN Mapped --
                    error ← TRUE;
```

```
            pin =>
                IF state IN Out THEN
                    ForkRead[stateNext: inPinned, andDirty: FALSE, oldSpace: NIL]
                ELSE IF state IN InSwappable THEN
                    state ← inPinned
                ELSE -- state = unmapped OR state = inPinned --
                    error ← TRUE;
            remapA --[firstClean]-- =>
                IF state = unmapped OR state = inPinned THEN
                    error ← TRUE
                ELSE
                    BEGIN
                    beingRemapped ← TRUE;
                    IF state IN InSwappable AND firstClean THEN
                        ForkWrite[stateNext: state, andWriteProtect: FALSE]
                    END;
            remapB --[from]-- =>
                IF state = unmapped THEN
                    error ← TRUE
                ELSE IF beingRemapped THEN
                    BEGIN
                    beingRemapped ← FALSE;
                    ForkRead[stateNext: inSwappableWarmest, andDirty: TRUE, oldSpace: from]
                    END;

            unmap =>
                IF state = unmapped THEN
                    error ← TRUE
                ELSE
                    BEGIN
                    pinned: BOOLEAN = state = inPinned;
                    dDirty ← TRUE;
                    IF state IN Out THEN
                        state ← unmapped
                    ELSE -- state IN In --
                        ForkWrite[stateNext: unmapped, andWriteProtect: ];
                    IF pinned AND outcome = [ok[]] THEN
                        outcome ← [notePinned[levelMax: level]]
                    END;

            unpin =>
                IF state = inPinned THEN
                    state ← inSwappableWarmest
                ELSE -- state = unmapped OR state IN Out OR state IN InSwappable --
                    error ← TRUE;
            ENDCASE => ERROR;
        IF ~forked THEN
            CheckIn[desc: region, restartFaulted: outcome.kind~ = spaceDMissing AND ~error]
        ELSE
            IF operation.afterForking = wait THEN
                AwaitNotCheckedOut[pageMember];
        IF error THEN
            outcome ← [error[region.state]]
        END;
    ENDCASE => ERROR;
END;
```

```
PageTransferProcess: PROCEDURE =
    BEGIN
    pageBuffer, pageRegion: VM.PageNumber;
    task: SwapTask.State;
    DO
        pageBuffer ← FilePageTransfer.Wait[];
        task ← SwapTask.Advance[pageBuffer];
        SELECT task.region.state FROM
            IN In =>
                BEGIN
                pageRegion ← task.region.interval.page + (pageBuffer-task.bufferPage);
                IF ~MStore.Relocate[VM.Interval[pageBuffer, 1], pageBuffer, PageMap.flagsAll,
                        PageMap.flagsNone].flags.writeProtected THEN
                    ApplyPatches[pageRegion, pageBuffer];
                [] ← MStore.Relocate[VM.Interval[pageBuffer, 1], pageRegion, PageMap.flagsNone, task.flags];
                IF task.countRemaining~ = 0 THEN
                    PageFault.Restart[interval: [pageRegion, 1]]
                END;
            ENDCASE =>
                MStore.Deallocate[interval: [pageBuffer, 1], promised: TRUE];
        IF task.countRemaining = 0 THEN
            BEGIN
            SwapBuffer.Deallocate[VM.Interval[task.bufferPage, task.region.interval.count]];
            CheckIn[desc: task.region, restartFaulted: TRUE]
            END
        ENDLOOP
    END;

ApplyPatches: PROCEDURE [page, pageBuffer: VM.PageNumber] =
    INLINE BEGIN OPEN VMMapLog, LOOPHOLE[pMapLogDesc, LONG POINTER TO Descriptor].patchTable;
    entryPtr: PatchTableEntryPointer;
    addressOfPage: LONG POINTER = Utilities.LongPointerFromPage[page];
    offset: LONG INTEGER;
    FOR entryPtr ← FIRST[PatchTableEntryPointer], entryPtr + SIZE[PatchTableEntry]
            WHILE entryPtr~ = limit DO OPEN LOOPHOLE[@entries[0], PatchTableEntryBasePointer][entryPtr];
        offset ← address-addressOfPage;
        IF offset IN [0..Environment.wordsPerPage) THEN
            (Utilities.LongPointerFromPage[pageBuffer] + offset)↑ ← value
        ENDLOOP
    END;

AwaitNotCheckedOut: ENTRY PROCEDURE [pageMember: VM.PageNumber] =
    INLINE BEGIN
    found: BOOLEAN;
    pDesc: PDesc;
    DO
        [found, pDesc] ← FirstNotBefore[pageMember];
        IF ~found OR baseDesc[pDesc].state~ = checkedOut THEN EXIT;
        WAIT checkIn
        ENDLOOP
    END;

CheckIn: ENTRY PROCEDURE [desc: Desc, restartFaulted: BOOLEAN] =
    BEGIN
    found: BOOLEAN;
    pDesc: PDesc;
    [found, pDesc] ← FirstNotBefore[desc.interval.page];
--assert--IF ~found OR baseDesc[pDesc].state~ = checkedOut THEN ERROR;
    IF desc.state = missing THEN
        -- Delete the entry
        BEGIN
--assert--IF desc.dPinned THEN ERROR;
--following must avoid avoid frame allocation!
        -- Move descriptors following pDesc one place towards first
        Utilities.LongMove[
            pSource: @baseDesc[pDesc + SIZE[Desc]],
            size: Utilities.ShortCARDINAL[pDescLast-pDesc],
            pSink: @baseDesc[pDesc]];
        pDescLast ← pDescLast-SIZE[Desc];
        IF pDesc<pDescCommutator THEN pDescCommutator ← pDescCommutator-SIZE[Desc];
        pDescMru ← pDescFirst -- ensure pDescMru< = pDescLast
        END
    ELSE
```

```
        -- Update the entry
        BEGIN
        desc.dTemperature ← LAST[DTemperature];
        baseDesc[pDesc] ← desc
        END;
--following must avoid avoid frame allocation!
    IF restartFaulted THEN
        PageFault.Restart[desc.interval];
    BROADCAST checkIn
    END;

CheckOut: ENTRY PROCEDURE [pageMember: VM.PageNumber, ifCheckedOut: ReturnWait]
        RETURNS [desc: Desc] =
    -- If returned descriptor state = missing, interval.page + interval.count gives start of next potential cached region
    INLINE BEGIN
    found: BOOLEAN;
    pDesc: PDesc;
    DO
        [found, pDesc] ← FirstNotBefore[pageMember];
        desc ← baseDesc[pDesc];
        IF found THEN
            BEGIN OPEN baseDesc[pDesc];
            IF state~ = checkedOut THEN
                BEGIN state ← checkedOut; EXIT END
            ELSE IF ifCheckedOut = return THEN
                EXIT
            ELSE
                WAIT checkIn
            END
        ELSE
            BEGIN desc.state ← missing; desc.interval.count ← 0; EXIT END
        ENDLOOP
    END;

-- rewrite the following to avoid frame allocation:

FirstNotBefore: INTERNAL PROCEDURE [page: VM.PageNumber] RETURNS [--found:--BOOLEAN, --pDesc:--PDesc] =
    BEGIN
    iDesc, iDescL, iDescU: IDesc;
    pageComp: VM.PageNumber;
--assert--IF page = pageTop THEN ERROR;
    BEGIN OPEN baseDesc[pDescMru]; -- interval.page, interval.count
    IF page IN [interval.page..interval.page + interval.count) THEN
        RETURN[TRUE, pDescMru] -- same as last call
    END;
    iDescL ← 0; iDescU ← (pDescLast-pDescFirst)/SIZE[Desc]; -- or maintain cDesc(-1) across insert/delete
    DO
        iDesc ← (iDescL + iDescU)/2;
        pageComp ← baseDesc[pDescFirst + SIZE[Desc]*iDesc].interval.page;
        IF page<pageComp THEN
            iDescU ← iDesc-1
        ELSE IF pageComp<page THEN
            iDescL ← iDesc + 1
        ELSE
            GO TO Exact;
        IF iDescU<iDescL THEN
            GO TO NotExact
        REPEAT
            Exact => RETURN[TRUE, --Cache[iDesc]--pDescFirst + SIZE[Desc]*iDesc];
            NotExact =>
                IF iDescL = 0 THEN
                    RETURN[FALSE, pDescFirst]
                ELSE
                    BEGIN
                    pDescMru ← pDescFirst + SIZE[Desc]*iDescU;
                    IF page<baseDesc[pDescMru].interval.page + baseDesc[pDescMru].interval.count THEN
                        RETURN[TRUE, pDescMru]
                    ELSE
                        RETURN[FALSE, pDescFirst + SIZE[Desc]*iDescL]
                    END
        ENDLOOP
    END;
```

```
Insert: PUBLIC ENTRY PROCEDURE [desc: Desc] RETURNS [descVictim: Desc] =
    BEGIN
    found: BOOLEAN;
    pDesc: PDesc;
    descVictim.dDirty ← FALSE; -- or set descVictim ← "descTop" ?
    IF pDescLast = pDescMax THEN
        -- Delete coldest unpinned descriptor
        BEGIN
        pDescCommOld: PDesc = pDescCommutator;
        DO OPEN baseDesc[pDescCommutator];
            IF state~ = checkedOut AND state ~IN In THEN
                BEGIN
                IF dTemperature = FIRST[DTemperature] THEN
                    BEGIN IF ~dPinned THEN EXIT END
                ELSE
                    dTemperature ← dTemperature-1;
                END;
            pDescCommutator ← IF pDescCommutator = pDescLast THEN pDescFirst ELSE pDescCommutator + SIZE[Desc];
            IF pDescCommutator = pDescCommOld THEN
                RuntimeInternal.WorryCallDebugger["RegionCacheTooSmall"]
            ENDLOOP;
        descVictim ← baseDesc[pDescCommutator];
        -- Move descriptors following pDescCommutator one place towards first
        Utilities.LongMove[
            pSource: @baseDesc[pDescCommutator + SIZE[Desc]],
            size: Utilities.ShortCARDINAL[pDescLast-pDescCommutator],
            pSink: @baseDesc[pDescCommutator]];
        pDescLast ← pDescLast-SIZE[Desc]
        END;
    [found, pDesc] ← FirstNotBefore[desc.interval.page];
--assert--IF found THEN ERROR;
    -- Move descriptors not before pDesc one place towards last
    pDescLast ← pDescLast + SIZE[Desc];
    Utilities.LongMove[
        pSource: @baseDesc[pDesc],
        size: Utilities.ShortCARDINAL[pDescLast-pDesc],
        pSink: @baseDesc[pDesc + SIZE[Desc]]];
    IF pDesc< = pDescCommutator THEN pDescCommutator ← pDescCommutator + SIZE[Desc];
    -- Insert desc
    desc.dTemperature ← FIRST[DTemperature];
    baseDesc[pDesc] ← desc
    END;

-- Initialization
--Process.DisableAbort[@checkIn];
pDescLast ← pDescCommutator ← pDescMru ← pDescFirst;
baseDesc[pDescLast] ← [
    interval: [page: pageTop, count: 0], -- page = page + count = positive infinity
    level: 0, -- don't care
    dTemperature: FIRST[DTemperature], -- don't care
    dPinned: TRUE,
    dDirty: FALSE, -- don't care
    state: missing, -- don't care
    levelMapped: 0, -- don't care
    beingRemapped: FALSE]; -- don't care

patchTable.header ← [
    limit: FIRST[VMMapLog.PatchTableEntryPointer],
    maxLimit: LOOPHOLE[SIZE[VMMapLog.PatchTableEntry]*(LAST[IPTE]-FIRST[IPTE] + 1)],
    entries: ];

LOOPHOLE[pMapLogDesc, LONG POINTER TO VMMapLog.Descriptor].patchTable ← @patchTable.header;

Process.Detach[FORK PageTransferProcess[]]

END.
```

LOG

Time: March 1978 By: McJones Action: Created file

Time: June 7, 1978 12:55 PM By: McJones Action: Apply called CheckIn even when CheckOut had failed
Time: June 7, 1978 5:27 PM By: McJones Action: CheckIn decreased commutator when it pointed to deleted entry
Time: June 16, 1978 10:56 AM By: McJones Action: forceOut didn't allocate buffer expected by PageTransferProcess
Time: June 16, 1978 5:32 PM By: McJones Action: (noMap hack) map set state even when ForkRead failed
Time: June 23, 1978 4:33 PM By: McJones Action: Added get, getResetDDirty; made CheckOut/CheckIn private; changed to new PageFault.Restart protocol.
Time: June 26, 1978 11:34 AM By: McJones Action: Apply outcome.pageNext logic in wrong place
Time: July 28, 1978 1:36 PM By: McJones Action: ForkWrite didn't update state of clean region
Time: August 4, 1978 2:06 PM By: McJones Action: Added beingRemapped logic
Time: August 28, 1978 6:38 PM By: Redell Action: Installed "gray-code" beingRemapped states in regions/spaces
Time: August 28, 1978 6:38 PM By: McJones Action: Added PatchTable, VMMode.pageFaulting tests
Time: September 5, 1978 1:35 PM By: Redell Action: Added facility to allow mapping off the end of a file
Time: September 9, 1978 2:39 PM By: McJones Action: Added notePinned from Apply unmap
Time: September 12, 1978 7:11 PM By: Redell Action: Fixed bug in ForkRead: if remapping and using only old pages that were already in, region never got set dirty
Time: September 19, 1978 6:32 PM By: McJones Action: age cleared dirty bit
Time: September 21, 1978 10:22 AM By: McJones Action: Insert replaced in region descriptor
Time: September 26, 1978 3:08 PM By: McJones CR20.25: Increased region cache size and added full message
Time: September 28, 1978 11:40 AM By: McJones CR20.31: remapB made in regions vacant
Time: September 29, 1978 3:02 PM By: Redell CR20.32: In-place Remap may set write-protect (change to 'flags' in ForkRead)
Time: October 4, 1978 5:25 PM By: McJones CR20.57: ForkWrite Promised even on forceOut
Time: January 29, 1979 11:09 AM By: McJones CR20.168: Patches were applied even on forceOut
Time: February 5, 1979 1:54 PM By: McJones CR20.171: Speeded up FirstNotBefore
Time: February 20, 1979 5:59 PM By: McJones CR20.177: Apply restarted faulted even on spaceDMissing (and error)
Time: March 7, 1979 1:31 PM By: McJones CR20.48,134: Added andWriteProtect to clean Operation

```
-- Things to consider:
-- 1) To allow multi-MDS, move global data to monitored record

DIRECTORY
    CachedSpace: FROM "CachedSpace",
    SwapperPrograms: FROM "SwapperPrograms";

CachedSpaceImpl: MONITOR EXPORTS CachedSpace, SwapperPrograms =

BEGIN OPEN CachedSpace;

CacheEntry: TYPE = RECORD [
    pCENext: PCE,
    desc: Desc];

PCE: TYPE = POINTER TO CacheEntry;

ICE: TYPE = [0..32); -- replacement algorithm assumes at least two entries

-- Monitor data:

rgCE: ARRAY ICE OF CacheEntry;

pCEMru: PCE;

pCEFree: PCE;

-- Interpretation of handles is not "strict": page may be any page of space

FindCacheEntry: INTERNAL PROCEDURE [space: Handle] RETURNS [found: BOOLEAN, pCE: PCE] =
    BEGIN
    pCECur, pCEPrev: PCE;
    FOR pCECur ← pCEMru, pCECur.pCENext WHILE pCECur~ = NIL DO OPEN pCECur;
        IF space.level = desc.level AND space.page IN [desc.interval.page..desc.interval.page + desc.interval.count) THEN
            GO TO Found;
        pCEPrev ← pCECur
        REPEAT
            Found =>
                BEGIN
                IF pCECur~ = pCEMru THEN
                    BEGIN
                    pCEPrev.pCENext ← pCENext;
                    pCENext ← pCEMru;
                    pCEMru ← pCECur
                    END;
                RETURN[TRUE, pCECur]
                END;
            FINISHED =>
                RETURN[FALSE, NIL]
        ENDLOOP
    END;

Delete: PUBLIC ENTRY PROCEDURE [space: Handle] =
    BEGIN
    found: BOOLEAN;
    pCE: PCE;
    [found, pCE] ← FindCacheEntry[space];
    -- Assume pCEMru = pCE
    IF found THEN
        BEGIN
--assert--IF pCE.desc.dPinned THEN ERROR;
        pCEMru ← pCE.pCENext;
        pCE.pCENext ← pCEFree;
        pCEFree ← pCE
        END
    END;

Get: PUBLIC ENTRY PROCEDURE [pDescResult: PDesc, space: Handle] =
```

```
    BEGIN
    found: BOOLEAN;
    pCE: PCE;
    [found, pCE] ← FindCacheEntry[space];
    IF found THEN
        pDescResult↑ ← pCE.desc
    ELSE
        pDescResult.state ← missing
    END;

Insert: PUBLIC ENTRY PROCEDURE [pDescVictim: PDesc, pDesc: PDesc] =
    BEGIN
    pCE, pCECur, pCEPrev: PCE;
--assert--IF FindCacheEntry[Handle[level: pDesc.level, page: pDesc.interval.page]].found THEN ERROR;
    pDescVictim.dDirty ← FALSE;
    IF pCEFree~ = NIL THEN
        -- Free cache entry is available; use it
        BEGIN pCE ← pCEFree; pCEFree ← pCEFree.pCENext END
    ELSE
        -- Replace lru element, which can't be same as mru since free chain is empty
        BEGIN
        pCE ← NIL;
        FOR pCECur ← pCEMru, pCECur.pCENext WHILE pCECur.pCENext~ = NIL DO
            IF ~pCECur.pCENext.desc.dPinned THEN
                BEGIN pCE ← pCECur.pCENext; pCEPrev ← pCECur END
            ENDLOOP;
--assert--IF pCE = NIL THEN ERROR; -- no unpinned cache entries
        pCEPrev.pCENext ← pCE.pCENext;
        IF pCE.desc.dDirty THEN
            pDescVictim↑ ← pCE.desc
        END;
    pCE.pCENext ← pCEMru;
    pCEMru ← pCE;
    pCE.desc ← pDesc↑
    END;

Update: PUBLIC ENTRY PROCEDURE [pDesc: PDesc] RETURNS [found: BOOLEAN] =
    BEGIN
    pCE: PCE;
    [found, pCE] ← FindCacheEntry[Handle[level: pDesc.level, page: pDesc.interval.page]];
    IF found THEN
        BEGIN
        pCE.desc ← pDesc↑;
        pCE.desc.dDirty ← TRUE
        END
    END;

BEGIN
pCE: PCE;
pCE ← pCEFree ← @rgCE[FIRST[ICE]];
THROUGH [FIRST[ICE]..LAST[ICE]) DO
    pCE.pCENext ← pCE + SIZE[CacheEntry];
    pCE ← pCE.pCENext
    ENDLOOP;
rgCE[LAST[ICE]].pCENext ← pCEMru ← NIL
END

END.
```

LOG

Time: June 1, 1978 3:00 PM By: McJones Action: Created file
Time: June 8, 1978 10:32 AM By: McJones Action: Update didn't mark cache entry dirty
Time: June 27, 1978 11:19 AM By: McJones Action: Added consistency check for no replaceable cache entries
Time: September 9, 1978 2:55 PM By: McJones Action: Suppressed clearing of dDirty in Insert

```
DIRECTORY
    PageMap: FROM "PageMap" USING [Flags],
    VM: FROM "VM" USING [Interval, PageCount, PageNumber];

MStore: DEFINITIONS =

BEGIN

Allocate: PROCEDURE [interval: VM.Interval, mayWait: BOOLEAN] RETURNS [countAllocated: VM.PageCount];

AwaitBelowThreshold: PROCEDURE;

Deallocate: PROCEDURE [interval: VM.Interval, promised: BOOLEAN];

Promise: PROCEDURE [count: VM.PageCount];

Relocate: PROCEDURE [interval: VM.Interval, pageDest: VM.PageNumber, flagsKeep, flagsAdd: PageMap.Flags]
        RETURNS [flags: PageMap.Flags, anyVacant: BOOLEAN];

SetThreshold: PROCEDURE [count: VM.PageCount];

END.
```

LOG

Time: March 1978 By: McJones Action: Created file
Time: September 5, 1978 3:40 PM By: Redell Action: Added 'anyVacant' result to Relocate.

```
-- Things to consider:
--   1) Divide allocationMap into "segments"
--   2) Drop result from SwapperPrograms.MStoreImpl (and param to SwapperPrograms.SimpleSpaceImpl)

DIRECTORY
    Environment: FROM "Environment" USING [bitsPerWord, Word],
    Inline: FROM "Inline" USING [BITAND, BITNOT, BITOR, BITSHIFT, COPY],
    MStore: FROM "MStore",
    PageMap: FROM "PageMap",
    --Process: FROM "Process" USING [DisableAbort],
    SpecialSpace: FROM "SpecialSpace",
    SwapperPrograms: FROM "SwapperPrograms",
    VM: FROM "VM" USING [Interval, PageCount, PageNumber, PageOffset];

MStoreImpl: MONITOR RETURNS [VM.PageCount] -- delete this result
    IMPORTS Inline, PageMap EXPORTS MStore, SpecialSpace, SwapperPrograms SHARES PageMap =

BEGIN OPEN Environment, MStore, PageMap, VM;

countVM: PageNumber = 37000B; -- should be from processor face?

ValueAnd: PROCEDURE [mv1, mv2: Value] RETURNS [Value] = LOOPHOLE[Inline.BITAND];

ValueOr: PROCEDURE [mv1, mv2: Value] RETURNS [Value] = LOOPHOLE[Inline.BITOR];

-- Monitor data:

-- Bit -(rp MOD bitsPerWord) of word rp/bitsPerWord of allocationMap is 1 iff real page rp is allocated.
-- Initially all real pages are allocated.
allocationMap: ARRAY [0..(maxRealPages + bitsPerWord-1)/bitsPerWord) OF Word;

countFree: PageCount ← 0;
countPromised: PageCount ← 0;
countThreshold: PageCount ← 0;

allocation, deallocation: CONDITION; -- aborts disabled

realPage: RealPageNumber ← FIRST[RealPageNumber]; -- rover for Allocate

Allocate: PUBLIC ENTRY PROCEDURE [interval: Interval, mayWait: BOOLEAN] RETURNS [countAllocated: PageCount] =
    BEGIN
    offset: PageOffset;
    w: CARDINAL;
    bit: Word;
    countAllocated ← IF mayWait THEN interval.count ELSE MIN[interval.count, countFree];
    FOR offset IN [0..countAllocated) DO
        WHILE countFree = 0 DO
            WAIT deallocation
            ENDLOOP;
        DO
            realPage ← IF realPage = LAST[RealPageNumber] THEN
                FIRST[RealPageNumber]
            ELSE
                realPage + 1;
            w ← realPage/bitsPerWord;
            bit ← Inline.BITSHIFT[1, realPage MOD bitsPerWord];
            IF Inline.BITAND[allocationMap[w], bit] = 0 THEN GO TO Found
            REPEAT Found =>
                BEGIN
                allocationMap[w] ← Inline.BITOR[allocationMap[w], bit];
                Assoc[interval.page + offset, Value[FALSE, flagsClean, realPage]]
                END
            ENDLOOP;
        countFree ← countFree-1;
        IF countFree + countPromised<countThreshold THEN BROADCAST allocation
        ENDLOOP
    END;
```

```
AwaitBelowThreshold: PUBLIC ENTRY PROCEDURE =
    BEGIN
    UNTIL countFree + countPromised<countThreshold DO
        WAIT allocation
        ENDLOOP
    END;

Deallocate: PUBLIC ENTRY PROCEDURE [interval: Interval, promised: BOOLEAN] =
    BEGIN -- Will accept an interval with holes in it and deallocate all real pages within it.
    offset: PageOffset;
    value: Value;
    w: CARDINAL;
    bit: Word;
    count: PageCount ← 0;
    FOR offset IN [0..interval.count) DO
        value ← SetF[interval.page + offset, valueVacant];
        IF ValueAnd[value, valueVacant] ~= valueVacant THEN
            BEGIN
            w ← value.realPage/bitsPerWord;
            bit ← Inline.BITSHIFT[1, value.realPage MOD bitsPerWord];
            allocationMap[w] ← Inline.BITAND[allocationMap[w], Inline.BITNOT[bit]];
            count ← count + 1
            END
        ENDLOOP;
    countFree ← countFree + count;
    IF promised THEN
        countPromised ← countPromised-count
    ELSE
        BROADCAST deallocation
    END;

Promise: PUBLIC ENTRY PROCEDURE [count: PageCount] =
    BEGIN
    countPromised ← countPromised + count
    END;

Relocate: PUBLIC ENTRY PROCEDURE [interval: Interval, pageDest: PageNumber, flagsKeep, flagsAdd: Flags]
        RETURNS [flags: Flags, anyVacant: BOOLEAN] =
    BEGIN

-- Note: the implementation of Relocate may make the affected pages temporarily "vacant". The only case in which it does not is
        when the interval is not being moved (pageDest = interval.page) and the old flags are being totally overwritten (flagsKeep
        = flagsNone). In particular, this is the only case in which it can be applied to pinned pages.
    valueKeep: Value = [FALSE, flagsKeep, 7777B];
    valueAdd: Value = [FALSE, flagsAdd, 0];
    offset: PageOffset;
    value1, value2, value3: Value;
    valueMax: Value ← [logSingleError:, flags: flagsNone, realPage: ];
    anyVacant ← FALSE;
    value2 ← IF flagsKeep = flagsNone AND interval.page = pageDest THEN valueAdd ELSE valueVacant;
    FOR offset IN [0..interval.count) DO
        value1 ← SetF[interval.page + offset, value2];
        IF ValueAnd[value1, valueVacant] = valueVacant THEN
            anyVacant ← TRUE
        ELSE
            BEGIN
            IF value2 = valueVacant THEN
                BEGIN
                value3 ← ValueOr[ValueAnd[value1, valueKeep], valueAdd];
                Assoc[pageDest + offset, value3]
                END;
            valueMax ← ValueOr[valueMax, value1];
            END
        ENDLOOP;
    flags ← valueMax.flags
    END;
```

```
SetThreshold: PUBLIC ENTRY PROCEDURE [count: PageCount] =
    BEGIN
    countThreshold ← count;
    BROADCAST allocation
    END;

-- SpecialSpace

realMemorySize: PUBLIC PageCount;

page: PageNumber;

-- Initialization
allocationMap[0] ← 177777B;
Inline.COPY[from: @allocationMap[0], to: @allocationMap[1], nwords: LENGTH[allocationMap]-1];
-- Process.DisableAbort[@allocation];
-- Process.DisableAbort[@deallocation];

realMemorySize ← 0;
FOR page IN [FIRST[PageNumber]..FIRST[PageNumber] + countVM) DO
    IF ValueAnd[SetF[page + FIRST[PageNumber], valueClean], valueVacant]~ = valueVacant THEN
        realMemorySize ← realMemorySize + 1 -- SetF of vacant page is no-op
    ENDLOOP;

RETURN[0] -- delete this result

END.
```

LOG

Time: March 1978 By: McJones Action: Created file
Time: June 7, 1978 2:25 PM By: McJones Action: LongMove, LongCopy had closed upper loop bounds
Time: September 5, 1978 5:15 PM By: Redell Action: Fixed Deallocate and Relocate to accept holes (vacant map entries) in intervals passed to them. Added 'anyVacant' result to Relocate.
Time: September 20, 1978 1:43 AM By: Redell Action: Fixed bug in detection of vacant map entries by Relocate and Deallocate. Made Relocate slightly smarter when called only to overwrite flags.
Time: January 25, 1979 11:15 AM By: McJones CR20.161: Added rover to Allocate
Time: August 10, 1979 11:15 AM By: McJones Action: Added SpecialSpace.realMemorySize
Time: August 21, 1979 6:00 PM By: McJones Action: Fixed bug in realMemorySize initialization

```
DIRECTORY
    VM: FROM "VM" USING [Interval, PageNumber];

PageFault: DEFINITIONS =

BEGIN

Await: PROCEDURE RETURNS [VM.PageNumber];

Restart: PROCEDURE [interval: VM.Interval];

END.
```

LOG

Time: March 1978 By: PMcJ Action: Created file
Time: June 23, 1978 1:27 PM By: PMcJ Action: Removed RestartQuantifier
Time: June 24, 1978 3:00 PM By: PMcJ Action: Interval, PageNumber moved to VM

```
DIRECTORY
    ControlDefs: FROM "ControlDefs" USING [Frame, GlobalFrameHandle, NullFrame, StateVector, WordPC],
    FrameOps: FROM "FrameOps" USING [GetReturnLink, MyLocalFrame],
    InlineDefs: FROM "InlineDefs" USING [COPY],
    PageFault: FROM "PageFault",
    ProcessInternal: FROM "ProcessInternal" USING [DisableInterrupts],
    ProcessOps: FROM "ProcessOps" USING [CurrentPSB, EnableAndRequeue, Queue, ReadyList, Requeue],
    PSBDefs: FROM "PSBDefs" USING [Condition, Empty, ProcessHandle],
    SDDefs: FROM "SDDefs" USING [SD, sPageFault],
    SwapperPrograms: FROM "SwapperPrograms",
    TrapOps: FROM "TrapOps" USING [ReadOTP],
    VM: FROM "VM" USING [Interval, PageNumber];

PageFaultImpl: MONITOR
        IMPORTS FrameOps, InlineDefs, ProcessInternal, ProcessOps, TrapOps
        EXPORTS PageFault, SwapperPrograms =

BEGIN

queueFaulted: ProcessOps.Queue ← NIL;
queueNoticed: ProcessOps.Queue ← NIL;
faultOccurred: CONDITION;

-- Event log (for debugging)
Event: TYPE = RECORD [
    page: VM.PageNumber,
    process: PSBDefs.ProcessHandle,
    local: POINTER TO local ControlDefs.Frame,
    global: ControlDefs.GlobalFrameHandle,
    pc: ControlDefs.WordPC];
lEvent: TYPE = [0..20);
rgEvent: ARRAY lEvent OF Event;
pEvent: POINTER TO Event ← @rgEvent[FIRST[lEvent]];

Trap: INTERNAL PROCEDURE = -- called from outside monitor!
    -- Simulate fault notification which will eventually be done by processor
    BEGIN
    page: VM.PageNumber;
    state: ControlDefs.StateVector;
    trapee: POINTER TO local ControlDefs.Frame;
    -- Capture state
    ProcessInternal.DisableInterrupts[]; -- must be first instruction
    state ← STATE; state.dest ← FrameOps.GetReturnLink[]; state.source ← ControlDefs.NullFrame;
    page ← LOOPHOLE[TrapOps.ReadOTP[]]; -- trap parameter
    LOOPHOLE[FrameOps.MyLocalFrame[], POINTER TO local ControlDefs.Frame].unused ← page; -- put it where Await can find it
    -- Log event
    trapee ← state.dest;
    pEvent↑ ← [page, ProcessOps.CurrentPSB↑, trapee, trapee.accesslink, trapee.pc];
    pEvent ← IF pEvent = @rgEvent[LAST[lEvent]] THEN @rgEvent[FIRST[lEvent]] ELSE pEvent + SIZE[Event];
    -- Simulate naked notify (recall interrupts are disabled)
    BEGIN OPEN cond: LOOPHOLE[faultOccurred, PSBDefs.Condition];
    IF cond.queue = PSBDefs.Empty THEN
        cond.wakeupWaiting ← yes
    ELSE
        NOTIFY faultOccurred
    END;
    -- Add this process to the fault queue and reschedule
    ProcessOps.EnableAndRequeue[ProcessOps.ReadyList, @queueFaulted, ProcessOps.CurrentPSB↑];
    RETURN WITH state
    END;

Await: PUBLIC ENTRY PROCEDURE RETURNS [VM.PageNumber] =
    BEGIN
    p: PSBDefs.ProcessHandle;
    DO
        IF queueFaulted~ = NIL THEN
            BEGIN
            p ← queueFaulted.link;
            ProcessOps.Requeue[@queueFaulted, @queueNoticed, p];
            WITH p.frame SELECT local FROM
```

```
                local => RETURN[unused];
                ENDCASE
            END;
        WAIT faultOccurred
        ENDLOOP
    END;

Restart: PUBLIC ENTRY PROCEDURE [interval: VM.Interval] =
    BEGIN
    pLast: PSBDefs.ProcessHandle = queueNoticed;
    p, pLink: PSBDefs.ProcessHandle;
    IF queueNoticed~ = NIL THEN
        FOR p ← queueNoticed.link, pLink DO
            pLink ← p.link;
            WITH p.frame SELECT local FROM
                local =>
                    IF unused IN [interval.page..interval.page + interval.count) THEN
                        ProcessOps.Requeue[@queueNoticed, ProcessOps.ReadyList, p];
                ENDCASE;
            IF p = pLast THEN EXIT
            ENDLOOP
    END;

SDDefs.SD[SDDefs.sPageFault] ← Trap;

rgEvent[0] ← [0, NIL, NIL, NIL, [0]];
InlineDefs.COPY[from: @rgEvent[0], nwords: SIZE[Event]*(LAST[IEvent]-FIRST[IEvent] + 1-1), to: @rgEvent[1]];

END.
```

LOG

Time: June 12, 1978 9:46 AM By: McJones Action: Created file
Time: June 20, 1978 10:53 AM By: McJones Action: Replaced "mark bit" with two queues;
Added correct simulation of naked notify in Notify
Time: June 23, 1978 1:30 PM By: McJones Action: Removed RestartQuantifier
Time: July 25, 1978 5:53 PM By: McJones Action: Trap didn't set state.dest (or .source);
Restart traversed queue incorrectly
Time: September 19, 1978 12:03 PM By: McJones Action: Added event log
Time: March 28, 1979 10:10 AM By: McJones Action: Initialization of rgEvent clobbered following words

```
-- This module is logically of "cached descriptor" type, but (DescribeSpace) must be initially resident

DIRECTORY
    CachedRegion: FROM "CachedRegion" USING [Apply, Desc, forceOut, Insert, Outcome, pin, unmap],
    CachedSpace: FROM "CachedSpace" USING [Desc, Get, Handle, Insert, Level, Update],
    Environment: FROM "Environment" USING [maxPagesInMDS, maxPagesInVM],
    File: FROM "File" USING [write],
    Inline: FROM "Inline" USING [BITAND],
    MStore: FROM "MStore" USING [Deallocate, Relocate],
    PageMap: FROM "PageMap" USING [flagsNone, flagsWriteProtected],
    SimpleSpace: FROM "SimpleSpace",
    Space: FROM "Space" USING [defaultWindow, Handle, PageCount, PageNumber, WindowOrigin],
    StoragePrograms: FROM "StoragePrograms",
    SwapperPrograms: FROM "SwapperPrograms",
    VM: FROM "VM" USING [Interval];

SimpleSpaceImpl: PROGRAM [countReal: Space.PageCount--delete this parameter--]
    IMPORTS CachedRegion, CachedSpace, Inline, MStore
    EXPORTS SimpleSpace, StoragePrograms, SwapperPrograms
    SHARES File =

BEGIN OPEN SimpleSpace;

locationData: TYPE = RECORD [ascending: BOOLEAN, total: VM.Interval, largestHole: VM.Interval];
locations: ARRAY Location OF POINTER TO locationData = [@first64K, @first64K, @hyperspace];
first64K: locationData ← [FALSE, [0, 256], [0, 0]];
hyperspace: locationData ← [TRUE, [256, Environment.maxPagesInVM-256], [256, 0]];

initializationDisabled: BOOLEAN ← FALSE;

-- StoragePrograms

DescribeSpace: PUBLIC PROCEDURE [options: StoragePrograms.SpaceOptions,
        page: Space.PageNumber, count: Space.PageCount, window: Space.WindowOrigin] =
    BEGIN
    location: Location;
    regionVictim: CachedRegion.Desc;
    space, spaceVictim: CachedSpace.Desc;
    level: CachedSpace.Level = (IF page<Environment.maxPagesInMDS THEN 2 ELSE 1)+(IF options.subspace THEN 1 ELSE 0);
            -- first64K is at level 1
    IF initializationDisabled OR options.error THEN ERROR;
    IF count=0 THEN RETURN;
    IF options.emptyInterval THEN
        FOR location IN Location DO OPEN locations[location];
            IF count>largestHole.count AND page>=total.page AND page+count<=total.page+total.count
                THEN largestHole ← [page: page, count: count]
            ENDLOOP;
    IF options.createRegion THEN
        BEGIN
        regionVictim ← CachedRegion.Insert[[
            interval: [page, count],
            level: level,
            dTemperature: ,
            dPinned: options.pinRegionD,
            dDirty: TRUE,
            state: SELECT TRUE FROM
                --options.special => special,
                options.pinned => inPinned,
                options.initiallyResident => inSwappableWarmest,
                options.mapped => outAlive,
                ENDCASE => unmapped,
            levelMapped: level-(IF options.subspace THEN 1 ELSE 0),
            beingRemapped: FALSE]];
        IF regionVictim.dDirty THEN ERROR; -- cache not large enough
        END;
    IF options.createSpace THEN
        BEGIN
        space ← [
            interval: [page, count],
            level: level,
            dPinned: options.pinSpaceD,
            dDirty: TRUE,
            pinned: options.pinned,
            state: IF options.mapped THEN mapped ELSE unmapped,
```

```
            writeProtected: 0 = Inline.BITAND[window.file.permissions, File.write],
            dataOrFile: file,
            pageRover: page,
            vp: long[window: window, countMapped: count]];
        CachedSpace.Insert[@spaceVictim, @space];
        IF spaceVictim.dDirty THEN ERROR; -- cache not large enough
        END;
    -- There should be an option for the following:
    IF options.initiallyResident AND options.createRegion AND 0 = Inline.BITAND[window.file.permissions, File.write] THEN
        -- The following depends on Relocate NOT temporarily vacating the pages
        [] ← MStore.Relocate[interval: [page, count], pageDest: page,
                flagsKeep: PageMap.flagsNone, flagsAdd: PageMap.flagsWriteProtected];
    IF options.mStoreDeallocate THEN
        MStore.Deallocate[interval: [page: page, count: count], promised: FALSE]
    END;

HandleFromPage: PUBLIC PROCEDURE [page: Space.PageNumber] RETURNS [Space.Handle] =
    BEGIN -- what about nonzero MDS & first64K?
    Handle: TYPE = RECORD [level: [0..4), page: Space.PageNumber]; -- CachedSpace.Handle!
    RETURN[LOOPHOLE[Handle[level: IF page<Environment.maxPagesInMDS THEN 2 ELSE 1, page: page]]]
    END;

SuperFromPage: PUBLIC PROCEDURE [page: Space.PageNumber] RETURNS [Space.Handle, Space.PageNumber] =
    BEGIN -- what about nonzero MDS & first64K?
    Handle: TYPE = RECORD [level: [0..4), page: Space.PageNumber]; -- CachedSpace.Handle!
    RETURN[LOOPHOLE[Handle[level: IF page<Environment.maxPagesInMDS THEN 1 ELSE 0, page: 0]], 0]
    END;

-- SimpleSpace

Create: PUBLIC PROCEDURE [count: Space.PageCount, location: SimpleSpace.Location] RETURNS [Space.Handle] =
    BEGIN OPEN locations[location];
    h: CachedSpace.Handle;
    IF count > largestHole.count THEN ERROR; -- free space exhausted
    largestHole.count ← largestHole.count-count;
    h.page ← largestHole.page + (IF ascending THEN 0 ELSE largestHole.count);
    h.level← IF h.page<Environment.maxPagesInMDS THEN 2 ELSE 1;
    IF ascending THEN largestHole.page ← largestHole.page + count;
    DescribeSpace[StoragePrograms.createSimpleSpace, h.page, count, Space.defaultWindow];
    RETURN[LOOPHOLE[h]]
    END;

DisableInitialization: PUBLIC PROCEDURE =
    BEGIN
    initializationDisabled ← TRUE
    END;

Page: PUBLIC PROCEDURE [handle: Space.Handle] RETURNS [Space.PageNumber] =
    BEGIN OPEN h: LOOPHOLE[handle, CachedSpace.Handle];
    RETURN[h.page]
    END;

Map: PUBLIC PROCEDURE [handle: Space.Handle, window: Space.WindowOrigin, andPin: BOOLEAN,
        countMapped: Space.PageCount] =
    -- Assumes window.file is writable and file doesn't end before space
    BEGIN OPEN h: LOOPHOLE[handle, CachedSpace.Handle];
    spaceD: CachedSpace.Desc;
    found: BOOLEAN;
    outcome: CachedRegion.Outcome;
    pageNext: Space.PageNumber;
    CachedSpace.Get[@spaceD, h];
--assert--IF spaceD.state~ = unmapped OR ~spaceD.dPinned THEN ERROR;
    spaceD.pinned ← andPin;
    spaceD.state ← mapped;
    spaceD.writeProtected ← (Inline.BITAND[window.file.permissions, File.write] = 0);
    spaceD.window ← window;
    spaceD.countMapped ← IF countMapped = defaultCount THEN spaceD.interval.count ELSE countMapped;
    spaceD.dataOrFile ← file;
    found ← CachedSpace.Update[@spaceD];
--assert--IF ~found THEN ERROR;
    [outcome, pageNext] ← CachedRegion.Apply[h.page, [ifMissing: report, ifCheckedOut: wait, afterForking: , vp:
            map[h.level]]];
--assert--IF outcome.kind~ = ok OR pageNext-h.page~ = spaceD.interval.count THEN ERROR;
```

```
    IF andPin THEN
        BEGIN
        [outcome, pageNext] ← CachedRegion.Apply[h.page, CachedRegion.pin];
--assert--IF outcome.kind~ = ok OR pageNext-h.page~ = spaceD.interval.count THEN ERROR
        END
    END;

Unmap: PUBLIC PROCEDURE [handle: Space.Handle] =
    BEGIN OPEN h: LOOPHOLE[handle, CachedSpace.Handle];
    spaceD: CachedSpace.Desc;
    outcome: CachedRegion.Outcome;
    pageNext: Space.PageNumber;
    found: BOOLEAN;
    CachedSpace.Get[@spaceD, h];
--assert--IF spaceD.state~ = mapped OR ~spaceD.dPinned THEN ERROR;
    [outcome, pageNext] ← CachedRegion.Apply[h.page, CachedRegion.unmap];
--assert--IF (outcome.kind~ = ok AND outcome.kind~ = notePinned) OR pageNext-h.page~ = spaceD.interval.count THEN ERROR;
    spaceD.pinned ← FALSE;
    spaceD.state ← unmapped;
    found ← CachedSpace.Update[@spaceD];
--assert--IF ~found THEN ERROR;
    END;

ForceOut: PUBLIC PROCEDURE [handle: Space.Handle] =
    BEGIN OPEN h: LOOPHOLE[handle, CachedSpace.Handle];
    outcome: CachedRegion.Outcome = CachedRegion.Apply[h.page, CachedRegion.forceOut].outcome;
--assert--IF outcome.kind~ = ok THEN ERROR;
    END;


END.
```

LOG

Time: June 19, 1978 5:55 PM By: McJones Action: Created file
Time: June 21, 1978 10:09 AM By: McJones Action: Result from Create was unassigned
Time: June 23, 1978 4:58 PM By: McJones Action: Map set space.writeProtected to TRUE
Time: June 29, 1978 4:40 PM By: McJones Action: Added free space allocation
Time: July 10, 1978 2:20 PM By: McJones Action: Made level dependent on (handle.)page (Create, Map, Unmap)
Time: August 3, 1978 9:17 AM By: McJones Action: made Map do pin to allow tests of paging
Time: August 7, 1978 5:15 PM By: Purcell Action: Added DisableCreate, parameters type to Create and andPin to Map
Time: August 10, 1978 3:56 PM By: Purcell Action: Moved generalized create to VFSPrograms.DescribeSpace, changed module parameters
Time: August 10, 1978 3:56 PM By: McJones Action: Added ForceOut
Time: September 7, 1978 3:12 PM By: Purcell Action: writeProtect with SetF, Describe swapped out resident descriptor, Create hyper vs mds
Time: September 9, 1978 3:11 PM By: McJones Action: Changes for CachedSpace.Desc pinned
Time: September 11, 1978 5:42 PM By: McJones Action: Made Unmap allow notePinned Outcome
Time: September 22, 1978 11:55 AM By: McJones Action: Made DescribeSpace do Relocate to writeProtected for any in region
Time: February 1, 1979 2:02 PM By: McJones CR20.169: Added pageRover to CachedSpace.Desc
Time: March 5, 1979 9:34 AM By: McJones Action: Changed Handle, etc.
Time: June 27, 1979 8:51 AM By: McJones Action: Deleted dependence on countReal
Time: August 15, 1979 9:26 AM By: McJones Action: Added countMapped to Map; added HandleFromPage, SuperFromPage

```
--Things to consider:
-- 1) Provision of multiple pools of buffer space (for deadlock avoidance)

DIRECTORY
    VM: FROM "VM" USING [Interval, PageCount];

SwapBuffer: DEFINITIONS =

BEGIN

Allocate: PROCEDURE [count: VM.PageCount] RETURNS [interval: VM.Interval];

Deallocate: PROCEDURE [interval: VM.Interval];

END.
```

LOG

Time: March 1978 By: McJones Action: Created file

```
DIRECTORY
    SwapBuffer: FROM "SwapBuffer",
    Environment: FROM "Environment" USING [bitsPerWord, maxPagesInMDS, Word],
    InlineDefs: FROM "InlineDefs" USING [BITAND, BITNOT, BITOR, BITSHIFT],
    --Process: FROM "Process" USING [DisableAbort],
    SwapperPrograms: FROM "SwapperPrograms",
    VM: FROM "VM" USING [Interval, PageCount, PageOffset];

SwapBufferImpl: MONITOR [intervalBuffer: VM.Interval] IMPORTS InlineDefs EXPORTS SwapBuffer, SwapperPrograms =

BEGIN

Status: TYPE = {free, busy}; -- i.e. 0 is free

allocationMap: ARRAY [0..(Environment.maxPagesInMDS + Environment.bitsPerWord-1)/Environment.bitsPerWord)
        OF Environment.Word;

deallocation: CONDITION;

Allocate: PUBLIC ENTRY PROCEDURE [count: VM.PageCount] RETURNS [interval: VM.Interval] =
    BEGIN
    offsetPage: VM.PageOffset;
    offsetOffset: VM.PageOffset;
    DO -- until enough space is available
        FOR offsetPage ← 0, offsetPage + offsetOffset + 1 WHILE offsetPage + count< = intervalBuffer.count DO
            BEGIN
            FOR offsetOffset IN [0..count) DO
                IF GetStatus[offsetPage + offsetOffset]~ = free THEN GO TO HoleTooSmall
                ENDLOOP;
            FOR offsetOffset IN [0..count) DO
                SetStatus[offsetPage + offsetOffset, busy]
                ENDLOOP;
            RETURN[[intervalBuffer.page + offsetPage, count]]
            EXITS HoleTooSmall => NULL
            END
            ENDLOOP;
        WAIT deallocation
        ENDLOOP
    END;

Deallocate: PUBLIC ENTRY PROCEDURE [interval: VM.Interval] =
    BEGIN
    offset: VM.PageOffset;
--assert--IF interval.page ~IN [intervalBuffer.page..intervalBuffer.page + intervalBuffer.count) OR interval.page + interval.count
        ~IN [intervalBuffer.page..intervalBuffer.page + intervalBuffer.count] THEN ERROR;
    FOR offset IN [0..interval.count) DO
        SetStatus[interval.page-intervalBuffer.page + offset, free]
        ENDLOOP;
    BROADCAST deallocation
    END;

GetStatus: PROCEDURE [offset: VM.PageOffset] RETURNS [Status] =
    BEGIN
    w: CARDINAL = offset/Environment.bitsPerWord;
    bit: Environment.Word = InlineDefs.BITSHIFT[1, offset MOD Environment.bitsPerWord];
    RETURN[IF InlineDefs.BITAND[allocationMap[w], bit] = 0 THEN free ELSE busy]
    END;

SetStatus: PROCEDURE [offset: VM.PageOffset, status: Status] =
    BEGIN
    w: CARDINAL = offset/Environment.bitsPerWord;
    bit: Environment.Word = InlineDefs.BITSHIFT[1, offset MOD Environment.bitsPerWord];
    allocationMap[w] ← InlineDefs.BITOR[InlineDefs.BITAND[allocationMap[w], InlineDefs.BITNOT[bit]],
            IF status = free THEN 0 ELSE bit]
    END;

-- Initialization
BEGIN
```

```
w: CARDINAL;
--assert--IF intervalBuffer.count>Environment.maxPagesInMDS THEN ERROR;
FOR w IN [0..LENGTH[allocationMap]) DO
    allocationMap[w] ← 0
    ENDLOOP;
--Process.DisableAbort[@deallocation]
END;

END.
```

LOG

Time: May 22, 1978 10:57 AM By: McJones Action: Created file
Time: August 4, 1978 9:42 AM By: McJones Action: Allocated allocationMap locally; added initialization, condition variable
Time: August 18, 1978 11:53 AM By: McJones Action: Initialization followed module END!
Time: September 7, 1978 10:42 AM By: Redell Action: Minor edit to allow allocation/deallocation of zero-length buffers
Time: September 8, 1978 2:00 PM By: McJones Action: Fixed check for end of intervalBuffer in Allocate

```
DIRECTORY
    CachedRegion: FROM "CachedRegion" USING [age, Apply, Outcome, pagefault, pageTop],
    Environment: FROM "Environment" USING [PageCount, PageNumber],
    MStore: FROM "MStore" USING [AwaitBelowThreshold, SetThreshold],
    PageFault: FROM "PageFault" USING [Await],
    Process: FROM "Process" USING [Detach],
    StoragePrograms: FROM "StoragePrograms",
    SwapperException: FROM "SwapperException" USING [Report],
    SwapperPrograms: FROM "SwapperPrograms" USING [CachedSpaceImpl, CachedRegionImpl, MStoreImpl, PageFaultImpl,
        SimpleSpaceImpl, SwapBufferImpl, SwapperExceptionImpl, SwapTaskImpl];

SwapperControl: PROGRAM [pageBuffer: Environment.PageNumber, countBuffer: Environment.PageCount,
    pMapLogDesc: LONG POINTER]
  IMPORTS CachedRegion, MStore, PageFault, Process, SwapperException, SwapperPrograms
  EXPORTS StoragePrograms =

BEGIN

PageFaultProcess: PROCEDURE =
  BEGIN
  page: Environment.PageNumber;
  outcome: CachedRegion.Outcome;
  DO
    page ← PageFault.Await[];
    outcome ← CachedRegion.Apply[page, CachedRegion.pagefault].outcome;
    WITH outcome SELECT FROM
      ok => NULL;
      regionDMissing, spaceDMissing, error => SwapperException.Report[page, CachedRegion.pagefault, outcome];
      ENDCASE => ERROR
    ENDLOOP
  END;

ReplacementProcess: PUBLIC PROCEDURE [threshold: Environment.PageCount] =
  BEGIN
  page, pageNext: Environment.PageNumber;
  outcome: CachedRegion.Outcome;
  MStore.SetThreshold[threshold];
  FOR page ← 0, IF pageNext = CachedRegion.pageTop THEN 0 ELSE pageNext DO
    MStore.AwaitBelowThreshold[];
    [outcome, pageNext] ← CachedRegion.Apply[page, CachedRegion.age];
    SELECT outcome.kind FROM
      ok => NULL;
      spaceDMissing => SwapperException.Report[page, CachedRegion.age, outcome];
      ENDCASE => ERROR
    ENDLOOP
  END;

countReal: Environment.PageCount = START SwapperPrograms.MStoreImpl;
START SwapperPrograms.SwapBufferImpl[intervalBuffer: [pageBuffer, countBuffer]];
START SwapperPrograms.CachedSpaceImpl;
START SwapperPrograms.PageFaultImpl;
START SwapperPrograms.SwapTaskImpl;
START SwapperPrograms.CachedRegionImpl[pMapLogDesc];
START SwapperPrograms.SimpleSpaceImpl[countReal: countReal];
START SwapperPrograms.SwapperExceptionImpl;

Process.Detach[FORK PageFaultProcess[]];

END.
```

LOG

Time: March 1978	*By: McJones*	*Action: Created file*
Time: June 20, 1978 10:33 AM	*By: McJones*	*Action: Added START SimpleSpaceImpl for testing*
Time: June 29, 1978 5:18 PM	*By: McJones*	*Action: Added module parameters for SimpleSpaceImpl*
Time: July 17, 1978 5:16 PM	*By: McJones*	*Action: Added countReal*
Time: July 26, 1978 4:39 PM	*By: McJones*	*Action: Added kludge Deallocate of free real memory*
Time: July 31, 1978 6:11 PM	*By: McJones*	*Action: Updated to new SwapperException interface*
Time: August 3, 1978 8:39 AM	*By: McJones*	*Action: Updated to new CachedRegion interface*
Time: August 10, 1978 7:04 PM	*By: Purcell*	*Action: Passed new parameters through to SimpleSpaceImpl*
Time: August 29, 1978 1:46 PM	*By: McJones*	*Action: Added pMapLogDesc to CachedRegionImpl, etc.*

```
DIRECTORY
    CachedRegion: FROM "CachedRegion" USING [Operation, Outcome],
    FrameOps: FROM "FrameOps" USING [Alloc, Free],
    RuntimeInternal: FROM "RuntimeInternal" USING [MakeFsi],
    SwapperException: FROM "SwapperException",
    SwapperPrograms: FROM "SwapperPrograms",
    VM: FROM "VM" USING [PageNumber];

SwapperExceptionImpl: MONITOR IMPORTS FrameOps, RuntimeInternal EXPORTS SwapperException, SwapperPrograms =

BEGIN OPEN SwapperException;

QE: TYPE = RECORD [
    pQEPrev: POINTER TO QE,
    page: VM.PageNumber,
    operation: CachedRegion.Operation,
    outcome: CachedRegion.Outcome];

PQE: TYPE = POINTER TO QE;

fsiQE: CARDINAL = RuntimeInternal.MakeFsi[SIZE[QE]];

pQELast: PQE ← NIL; -- points to last of fifo queue of exception reports

report: CONDITION;

Await: PUBLIC ENTRY PROCEDURE RETURNS [page: VM.PageNumber, operation: CachedRegion.Operation,
        outcome: CachedRegion.Outcome] =
    BEGIN
    pQE, pQENext: PQE;
    -- Wait for queue nonempty
    WHILE pQELast = NIL DO WAIT report ENDLOOP;
    -- Remove first report from queue and return it
    FOR pQE ← pQELast, pQE.pQEPrev WHILE pQE.pQEPrev~ = NIL DO
        pQENext ← pQE
        ENDLOOP;
    [, page, operation, outcome] ← pQE↑;
    IF pQE = pQELast THEN
        pQELast ← NIL
    ELSE
        pQENext.pQEPrev ← NIL;
    FrameOps.Free[pQE]
    END;

Report: PUBLIC ENTRY PROCEDURE [page: VM.PageNumber, operation: CachedRegion.Operation,
        outcome: CachedRegion.Outcome] =
    BEGIN
    -- Add report to end of queue
    pQE: PQE = LOOPHOLE[FrameOps.Alloc[fsiQE], PQE];
    pQE↑ ← [pQELast, page, operation, outcome];
    pQELast ← pQE;
    NOTIFY report
    END;

END.
```

LOG

Time: June 6, 1978 3:31 PM By: McJones Action: Created file (stub only)
Time: July 31, 1978 10:53 AM By: McJones Action: Replace stubs with real implementation

```
DIRECTORY
    Environment: FROM "Environment" USING [PageCount],
    VM: FROM "VM" USING [Interval, PageCount];

SwapperPrograms: DEFINITIONS =

BEGIN

CachedRegionImpl: PROGRAM [pMapLogDesc: LONG POINTER]; -- for patch table

CachedSpaceImpl: PROGRAM;

MStoreImpl: PROGRAM RETURNS [countReal: VM.PageCount];

PageFaultImpl: PROGRAM;

SwapBufferImpl: PROGRAM [intervalBuffer: VM.Interval];

SwapperExceptionImpl: PROGRAM;

SwapTaskImpl: PROGRAM;

SimpleSpaceImpl: PROGRAM [countReal: Environment.PageCount];

END.
```

LOG

Time: June 6, 1978 4:05 PM By: McJones Action: Created file
Time: June 20, 1978 10:31 AM By: McJones Action: Added SimpleSpaceImpl
Time: June 29, 1978 3:40 PM By: McJones Action: Added parameters to SimpleSpaceImpl
Time: July 17, 1978 4:42 PM By: McJones Action: Added countReal to MStoreImpl
Time: August 10, 1978 7:09 PM By: Purcell Action: New SimpleSpace interface
Time: August 29, 1978 1:38 PM By: McJones Action: Added pMapLogDesc to CachedRegionImpl

```
DIRECTORY
    CachedRegion: FROM "CachedRegion" USING [Desc],
    PageMap: FROM "PageMap" USING [Flags],
    VM: FROM "VM" USING [PageCount, PageNumber];

SwapTask: DEFINITIONS =

BEGIN

State: TYPE = RECORD [
    next: PRIVATE PState,
    region: CachedRegion.Desc,
    flags: PageMap.Flags,
    bufferPage: VM.PageNumber,
    countRemaining: VM.PageCount];

PState: PRIVATE TYPE = POINTER TO State;

Fork: PROCEDURE [region: CachedRegion.Desc, flags: PageMap.Flags, bufferPage: VM.PageNumber, swapCount: VM.PageCount]
        RETURNS [forked: BOOLEAN];

Advance: PROCEDURE [page: VM.PageNumber] RETURNS [state: State];

END.
```

LOG

Time: March 1978 By: McJones Action: Created file
Time: June 29, 1978 5:50 PM By: McJones Action: Moved PageCount, PageNumber to VM
Time: September 5, 1978 2:44 PM By: Redell Action: Replaced over-complex 'Action' record with 'flags' field in swaptask 'State' record. Added 'swapCount' argument and 'forked' result to Fork.

```
-- Things to consider:
-- 1) When converting to multi-MDS, PState must be long pointer; may only free frame from same MDS as allocated it!
-- 2) Change Advance to return region, action, bufferPage, and count (or lastFlag) separately

DIRECTORY
    CachedRegion: FROM "CachedRegion" USING [Desc],
    FrameOps: FROM "FrameOps" USING [Alloc, Free],
    PageMap: FROM "PageMap" USING [Flags],
    RuntimeInternal: FROM "RuntimeInternal" USING [MakeFsi],
    SwapperPrograms: FROM "SwapperPrograms",
    SwapTask: FROM "SwapTask",
    VM: FROM "VM" USING [PageNumber, PageCount];

SwapTaskImpl: MONITOR
    IMPORTS FrameOps, RuntimeInternal
    EXPORTS SwapperPrograms, SwapTask
    SHARES SwapTask =

BEGIN OPEN SwapTask;

fsiState: CARDINAL = RuntimeInternal.MakeFsi[SIZE[State]];

swapTaskList: PState ← NIL;

Fork: PUBLIC ENTRY PROCEDURE [region: CachedRegion.Desc, flags: PageMap.Flags, bufferPage: VM.PageNumber, swapCount:
        VM.PageCount] RETURNS [forked: BOOLEAN] =
    BEGIN
    pState: PState;
    IF ~(forked ← swapCount>0) THEN RETURN;
    pState ← LOOPHOLE[FrameOps.Alloc[fsiState], PState];
    pState↑ ← [swapTaskList, region, flags, bufferPage, swapCount];
    swapTaskList ← pState
    END;

Advance: PUBLIC ENTRY PROCEDURE [page: VM.PageNumber] RETURNS [state: State] =
    BEGIN
    pState, pStatePrev: PState;
    pState ← swapTaskList;
    DO OPEN pState;
--assert--IF pState = NIL THEN ERROR;
        IF page IN [bufferPage..bufferPage + region.interval.count) THEN EXIT;
        pStatePrev ← pState;
        pState ← next
        ENDLOOP;
    pState.countRemaining ← pState.countRemaining-1;
    state ← pState↑;
    IF pState.countRemaining = 0 THEN
        BEGIN
        IF pState = swapTaskList THEN swapTaskList ← pState.next ELSE pStatePrev.next ← pState.next;
        FrameOps.Free[pState]
        END
    END;

END.
```

LOG

Time: March 1978 By: McJones Action: Created file
Time: September 5, 1978 4:39 PM By: Redell Action: Removed 'Action' records. Added 'swapCount' argument and 'forked' result to Fork.

```
PACK
    AltoDiskImpl,
    FileCacheImpl,
    FilerControl,
    FilerExceptionImpl,
    FilerTransferImpl,
    FileTaskImpl,
    RemotePageTransferImpl,
    SubVolumeImpl;

Filer: CONFIGURATION
    IMPORTS DiskChannel, Process, ResidentMemory
    EXPORTS AltoDisk, FileCache, FilePageTransfer, FilerException, LabelTransfer, StoragePrograms, SubVolume =
BEGIN

AltoDiskImpl;

FileCacheImpl;

FilerControl;

FilerExceptionImpl;

FilerTransferImpl;

FileTaskImpl;

RemotePageTransferImpl;

SubVolumeImpl;

END.
```

LOG

Time: February 26, 1979 5:29 PM Action: Created file from old FilePageTransferrer.config
Time: March 23, 1979 2:39 PM Action: Converted to Mesa 5.0
Time: April 11, 1979 1:50 PM Action: Added AltoDisk to EXPORTs (temporary, for MesaRuntime)

```
AltoDiskController: DEFINITIONS =

BEGIN
Acquire, Release: PROCEDURE;
END.
```

LOG

Time: August 4, 1978 1:27 PM By: Redell Action: Created file

```
-- Lowest level Alto file access module, used by Filer. (Modified copy of Alto/Mesa module "DiskIO")

DIRECTORY
  AltoDefs: FROM "AltoDefs" USING [BYTE, PageNumber, PageSize],
  AltoFileDefs: FROM "AltoFileDefs" USING [
    DISK, eofDA, fillinDA, SN, vDA, vDC],
  AltoDisk: FROM "AltoDisk" USING [
    CB, CBinit, CBptr, CBZ, CBZptr, DA, DC, DDC, DiskPageDesc, DiskRequest,
    DL, DS, DSfakeStatus, DSfreeStatus, DSgoodStatus, DSmaskStatus, FID,
    InvalidDA, ICBZ, nCB, nDisks, nHeads, nSectors, nTracks, RetryCount],
  InlineDefs: FROM "InlineDefs" USING [BITAND, COPY, DIVMOD],
  ProcessInternal: FROM "ProcessInternal" USING [DisableInterrupts, EnableInterrupts];

AltoDiskImpl: PROGRAM
   IMPORTS InlineDefs, ProcessInternal
   EXPORTS AltoDisk
   SHARES AltoDisk =

BEGIN OPEN AltoDisk;

  PageNumber: TYPE = AltoDefs.PageNumber;
  DISK: TYPE = AltoFileDefs.DISK;
  SN: TYPE = AltoFileDefs.SN;
  vDA: TYPE = AltoFileDefs.vDA;
  vDC: TYPE = AltoFileDefs.vDC;
  BYTE: TYPE = AltoDefs.BYTE;

  driveNumber: PUBLIC [0..1] ← 0;
  sysdisk: DISK ← DISK[nDisks,nTracks,nHeads,nSectors];
  disk: POINTER TO DISK = @sysdisk;

  Zero: PUBLIC PROCEDURE [p:POINTER, l:CARDINAL] =
    BEGIN
    IF l=0 THEN RETURN;  p↑ ← 0;
    InlineDefs.COPY [from:p, to:p+1, nwords:l-1];
    RETURN
    END;

  SetDisk: PUBLIC PROCEDURE [d:POINTER TO DISK] =
    BEGIN disk↑ ← d↑; RETURN END;

  GetDisk: PUBLIC PROCEDURE RETURNS [POINTER TO DISK] =
    BEGIN RETURN[disk] END;

  ResetDisk: PUBLIC PROCEDURE RETURNS [POINTER TO DISK] =
    BEGIN
    disk↑ ← DISK[nDisks,nTracks,nHeads,nSectors];
    RETURN[disk]
    END;

  VirtualDA: PUBLIC PROCEDURE [da:DA] RETURNS [vDA] =
    BEGIN
    RETURN[IF da = DA[0,0,0,0,0] THEN AltoFileDefs.eofDA ELSE vDA [
      ((da.disk*disk.tracks+da.track)*disk.heads+
       da.head)*disk.sectors+da.sector]];
    END;

  RealDA: PUBLIC PROCEDURE [v:vDA] RETURNS [da:DA] =
    BEGIN
    i: CARDINAL ← v;
    da ← DA[0,0,0,0,0];
    IF v # AltoFileDefs.eofDA THEN
      BEGIN
      [i,da.sector] ← InlineDefs.DIVMOD[i,disk.sectors];
      [i,da.head] ← InlineDefs.DIVMOD[i,disk.heads];
      [i,da.track] ← InlineDefs.DIVMOD[i,disk.tracks];
      [i,da.disk] ← InlineDefs.DIVMOD[i,disk.disks];
      IF i # 0 THEN da ← InvalidDA;
      END;
    RETURN
    END;
```

```
-- Disk transfer "process"

DCseal: AltoDefs.BYTE = 111B;

DCs: ARRAY vDC OF DC = [
  DC[DCseal,DiskRead, DiskRead, DiskRead, 0,0],  -- ReadHLD
  DC[DCseal,DiskCheck,DiskRead, DiskRead, 0,0],  -- ReadLD
  DC[DCseal,DiskCheck,DiskCheck,DiskRead, 0,0],  -- ReadD
  DC[DCseal,DiskWrite,DiskWrite,DiskWrite,0,0],  -- WriteHLD
  DC[DCseal,DiskCheck,DiskWrite,DiskWrite,0,0],  -- WriteLD
  DC[DCseal,DiskCheck,DiskCheck,DiskWrite,0,0],  -- WriteD
  DC[DCseal,DiskCheck,DiskCheck,DiskCheck,1,0],  -- SeekOnly
  DC[DCseal,DiskCheck,DiskCheck,DiskCheck,0,0]]; -- DoNothing

nextDiskCommand: POINTER TO CBptr = LOOPHOLE[521B];
diskStatus: POINTER TO DS = LOOPHOLE[522B];
lastDiskAddress: POINTER TO DA = LOOPHOLE[523B];
sectorInterrupts: POINTER TO CARDINAL = LOOPHOLE[524B];

-- DoDiskCommand assumes that the version number in a FID (and
-- in an FP) will never be used (is always one).  It further
-- assumes that if fp is NIL, a FreePageFID was meant;
-- this allows the rest of the world to use short (3 word) FPs.

FreePageFID: FID = FID[LAST[CARDINAL], SN[1,1,1,17777B, LAST[CARDINAL]]];

NonZeroWaitCell: WORD ← 1;
waitCell: POINTER TO WORD ← @NonZeroWaitCell;

ResetWaitCell: PUBLIC PROCEDURE =
  BEGIN
  waitCell ← @NonZeroWaitCell;
  END;

SetWaitCell: PUBLIC PROCEDURE [p: POINTER TO WORD] RETURNS [preval: POINTER TO WORD] =
  BEGIN
  ProcessInternal.DisableInterrupts[];
  preval ← waitCell;
  waitCell ← p;
  ProcessInternal.EnableInterrupts[];
  RETURN;
  END;

DoDiskCommand: PUBLIC PROCEDURE [arg:POINTER TO DDC] =
  BEGIN OPEN arg;
  ptr, next, prev: CBptr;
  la: POINTER TO DL;
  zone: CBZptr = cb.zone;
  cb.headerAddress ← @cb.header;
  IF (la ← GetLabelAddress[cb]) = NIL THEN
    SetLabelAddress[cb, la ← @cb.label];
  SetDataAddress[cb, ca];
  IF cb.normalWakeups = 0 THEN cb.normalWakeups ← zone.normalWakeups;
  IF cb.errorWakeups = 0 THEN cb.errorWakeups ← zone.errorWakeups;
  IF fp = NIL THEN la.fileID ← FreePageFID
  ELSE la.fileID ← FID[1,fp.serial];
  la.page ← cb.page ← page;
  IF da # AltoFileDefs.fillinDA THEN cb.header.diskAddress ← RealDA[da];
  IF restore THEN cb.header.diskAddress.restore ← 1;
  cb.command ← DCs[action];
  cb.command.exchange ← driveNumber;
  prev ← PrevCB[zone];
  -- Put the command on the disk controller's queue
  UNTIL waitCell↑ # 0 DO NULL ENDLOOP; -- Wait for Trident to finish
  ProcessInternal.DisableInterrupts[];
  IF (next ← nextDiskCommand↑) # NIL THEN
    BEGIN
    DO  ptr ← next;  next ← ptr.nextCB;
      IF next = NIL THEN EXIT;
      ENDLOOP;
```

```
      ptr.nextCB ← cb;
      END;
    -- Take care of a possible race with disk controller.  The disk
    -- may have gone idle (perhaps due to an error) even as we were
    -- adding a command to the chain.  To make sure there was no
    -- error, we check the status of the previous cb in this zone.
    IF nextDiskCommand↑ = NIL THEN
      SELECT MaskDS[prev.status, DSmaskStatus] FROM
        DSfreeStatus, DSgoodStatus => nextDiskCommand↑ ← cb;
        ENDCASE;
    ProcessInternal.EnableInterrupts[];
    EnqueueCB[zone,cb];
    RETURN
    END;

GetLabelAddress: PROCEDURE [cb: CBptr] RETURNS [la: POINTER TO DL] =
  BEGIN
  la ← LOOPHOLE[cb.labelAddressLow];
  END;

SetLabelAddress: PROCEDURE [cb: CBptr, la: POINTER TO DL] =
  BEGIN
  cb.labelAddressLow ← LOOPHOLE[la];
  cb.labelAddressHigh ← 0;
  END;

LongPointer: TYPE = MACHINE DEPENDENT RECORD[low: WORD, fill: BYTE, high: BYTE];

GetDataAddress: PROCEDURE [cb: CBptr] RETURNS [ca: LONG POINTER] =
  BEGIN OPEN LOOPHOLE[ca, LongPointer];
  low ← cb.dataAddressLow;
  high ← cb.dataAddressHigh;
  fill ← 0;
  END;

SetDataAddress: PROCEDURE [cb: CBptr, ca: LONG POINTER] =
  BEGIN OPEN LOOPHOLE[ca, LongPointer];
  cb.dataAddressLow ← low;
  cb.dataAddressHigh ← high;
  END;



  -- Disk command block queue

  InitializeCBstorage: PUBLIC PROCEDURE [
    zone:CBZptr, nCBs:CARDINAL, page:PageNumber, init:CBinit] =
    BEGIN
    cb: CBptr;
    i: CARDINAL;
    nq: CARDINAL = nCBs + 1;
    length: CARDINAL = SIZE[CBZ] + nCBs*(SIZE[CB] + SIZE[CBptr]);
    queue: DESCRIPTOR FOR ARRAY OF CBptr ←
      DESCRIPTOR[@zone.queueVec,nq];
    cbVector: DESCRIPTOR FOR ARRAY OF CB ←
      DESCRIPTOR[@zone.queueVec + SIZE[CBptr]*nq,nCBs];
    IF init = clear THEN Zero[zone,length];
    zone.currentPage ← page;  zone.cbQueue ← queue;
    zone.qTail ← 0;  zone.qHead ← 1;
    queue[0] ← NIL; -- end of queue;
    FOR i IN [1..nCBs] DO
      queue[i] ← cb ← @cbVector[i-1];
      cb.zone ← zone;  cb.status ← DSfreeStatus;
      ENDLOOP;
    RETURN
    END;

  NumCBs: PROCEDURE [zone:CBZptr] RETURNS [CARDINAL] =
    BEGIN
    RETURN[LENGTH[zone.cbQueue]-1]
    END;
```

```
ClearCB: PROCEDURE [cb:CBptr] =
  BEGIN
  zone: CBZptr = cb.zone;
  Zero[cb,SIZE[CB]];
  cb.zone ← zone;
  RETURN
  END;

EnqueueCB: PROCEDURE [zone:CBZptr, cb:CBptr] =
  BEGIN  i: CARDINAL ← zone.qTail;
  IF zone.cbQueue[i] # NIL THEN ERROR;
  zone.cbQueue[i] ← cb;
  IF (i ← i+1) = LENGTH[zone.cbQueue] THEN i ← 0;
  zone.qTail ← i;
  RETURN
  END;

DequeueCB: PROCEDURE [zone:CBZptr] RETURNS [cb:CBptr] =
  BEGIN  i: CARDINAL ← zone.qHead;
  IF (cb ← zone.cbQueue[i]) = NIL THEN ERROR;
  zone.cbQueue[i] ← NIL;
  IF (i ← i+1) = LENGTH[zone.cbQueue] THEN i ← 0;
  zone.qHead ← i;
  RETURN
  END;

PrevCB: PROCEDURE [zone:CBZptr] RETURNS [cb:CBptr] =
  BEGIN  i: CARDINAL ← zone.qTail;
  i ← (IF i=0 THEN LENGTH[zone.cbQueue] ELSE i) - 1;
  IF (cb ← zone.cbQueue[i]) = NIL THEN ERROR;
  RETURN
  END;

CleanupCBqueue: PUBLIC PROCEDURE [zone:CBZptr] =
  BEGIN
  cb: CBptr;
  UNTIL zone.cbQueue[zone.qHead] = NIL DO
    cb ← GetCB[zone,dontClear];
    ENDLOOP;
  RETURN
  END;


-- Removing CBs from the queue.  If for some reason the disk has
-- gone idle without executing the command, we fake an error
-- in it so that the entire zone of CBs will get retryed.

RetryableDiskError: PUBLIC SIGNAL [cb:CBptr] = CODE;
UnrecoverableDiskError: PUBLIC SIGNAL [cb:CBptr] = CODE;

MaskDS: PROCEDURE [DS, DS] RETURNS [DS] = LOOPHOLE[InlineDefs.BITAND];

GetCB: PUBLIC PROCEDURE [zone:CBZptr, init:CBinit] RETURNS [cb:CBptr] =
  BEGIN
  s:DS;  da:vDA;  ddc:DDC;  ec:CARDINAL;
  cb ← DequeueCB[zone];
  UNTIL cb.status.done # 0 DO
    -- not zero means done or fake or free
    IF nextDiskCommand↑ = NIL
    AND cb.status.done = 0 THEN
      cb.status ← DSfakeStatus;
    ENDLOOP;
  cb.command.seal ← 0; -- remove command seal
  s ← MaskDS[cb.status, DSmaskStatus];
  SELECT s FROM
    DSgoodStatus =>
      BEGIN
      IF cb.header.diskAddress.restore=0 THEN
        BEGIN
        zone.errorCount ← 0;
        zone.currentBytes ← GetLabelAddress[cb].bytes;
        IF zone.cleanup # LOOPHOLE[0] THEN zone.cleanup[cb];
```

```
      END;
    IF init = clear THEN ClearCB[cb];
    END;
  DSfreeStatus =>
    ClearCB[cb]; -- really means DSneverBeenUsed
  ENDCASE =>
    BEGIN -- some error occurred
    -- busy wait until disk controller is idle
    UNTIL nextDiskCommand↑ = NIL DO NULL ENDLOOP;
    ec ← zone.errorCount ← zone.errorCount + 1;
    IF ec >= RetryCount THEN ERROR UnrecoverableDiskError[cb];
    da ← zone.errorDA ← VirtualDA[cb.header.diskAddress];
    IF cb.status.finalStatus = CheckError THEN zone.checkError ← TRUE;
    InitializeCBstorage [
      zone,NumCBs[zone],cb.page,dontClear];
    IF ec > RetryCount/2 THEN
      BEGIN -- start a restore before signalling the error
      lastDiskAddress↑ ← InvalidDA;
      ddc ← DDC[GetCB[zone,clear],NIL,da,0,NIL,TRUE,SeekOnly];
      DoDiskCommand[@ddc];
      END;
    ERROR RetryableDiskError[cb];
    END;
RETURN
END;

-- Don't all Cleanup procedures need to be locked?
```

```
-- Segment swapper

-- Note that each CB is used twice: first to hold the disk label
-- for page i-1, and then to hold the DCB for page i.  It isn't
-- reused until the DCB for page i-1 is correctly done, which
-- is guaranteed to be after the disk label for page i-1 is no
-- longer needed, since things are done strictly sequentially by
-- page number.

-- Currently, DiskRequest.lastAction is not used by SwapPages.

DiskCheckError: PUBLIC SIGNAL [page:PageNumber] = CODE;

SwapPages: PUBLIC PROCEDURE [arg:POINTER TO swap DiskRequest]
  RETURNS [PageNumber, CARDINAL] =
  BEGIN OPEN arg;
  i: PageNumber;
  cb, nextcb: CBptr;
  cbzone: ARRAY [0..ICBZ) OF UNSPECIFIED;
  zone: CBZptr = @cbzone[0];
  ddc: DDC ← DDC[,ca,da↑,,fp,FALSE,action];
  InitializeCBstorage[zone,nCB,firstPage,clear];
  IF desc # NIL THEN
    BEGIN zone.info ← desc;
    zone.cleanup ← GetDiskPageDesc;
    END;
  BEGIN
    ENABLE RetryableDiskError --[cb]-- =>
      BEGIN
      ddc.da ← zone.errorDA;
      ddc.ca ← GetDataAddress[cb];
      RETRY END;
    cb ← GetCB[zone,clear];
    FOR i ← zone.currentPage, i+1 UNTIL i=lastPage+1 DO
      IF ddc.da = AltoFileDefs.eofDA THEN EXIT;
      IF signalCheckError AND zone.errorCount = RetryCount/2
        THEN SIGNAL DiskCheckError[i];
      nextcb ← GetCB[zone,clear];
      SetLabelAddress[cb, LOOPHOLE[@nextcb.header.diskAddress]];
      ddc.cb ← cb;  ddc.page ← i;
      IF i # zone.currentPage THEN ddc.da ← AltoFileDefs.fillinDA;
      DoDiskCommand[@ddc];
      IF ~fixedCA THEN ddc.ca ← ddc.ca+AltoDefs.PageSize;
      cb ← nextcb;
      ENDLOOP;
    CleanupCBqueue[zone];
    END; -- of enable block
  RETURN[i-1,zone.currentBytes]
  END;

  GetDiskPageDesc: PROCEDURE [cb:CBptr] =
    BEGIN
    la: POINTER TO DL = GetLabelAddress[cb];
    desc: POINTER TO DiskPageDesc ← cb.zone.info;
    desc↑ ← DiskPageDesc [
      VirtualDA[la.prev],
      VirtualDA[cb.header.diskAddress],
      VirtualDA[la.next],
      la.page, la.bytes];
    RETURN
    END;

END..
```

LOG

Time: September 14, 1978 5:36 PM By: Redell Action: Created File from Alto/Mesa DiskIO converted to LONG POINTER disk controller.
Time: September 19, 1978 11:29 PM By: Redell Action: Enabled LONG POINTER disk controller.
Time: March 7, 1979 11:58 AM By: Redell Action: Added IMPORTS of InlineDefs and ProcessInternal, as required by Mesa 5.0.
Time: July 31, 1979 11:26 AM By: Knutsen Action: Made a Pilot-standard module header paragraph.

```
DIRECTORY
    File: FROM "File",
    FileCache: FROM "FileCache",
    FileInternal: FROM "FileInternal",
    FilerPrograms: FROM "FilerPrograms";

FileCacheImpl: MONITOR
    EXPORTS FileCache, FilerPrograms =

BEGIN OPEN FileInternal;

-- General cache mechanism --

Cache: TYPE = POINTER TO CacheRecord;

CacheRecord: TYPE = RECORD [
    cacheType: CacheType,
    mru: CEptr, -- most recently used entry (head of lru chain)
    free: CEptr, -- head of free chain
    CleanUp: CacheCleanUpProc]; -- proc to cleanup when an entry is replaced

CacheType: TYPE = {file, pageGroup};

CacheEntry: TYPE = RECORD [
    next: CEptr, -- next entry in lru chain
    refCnt: CARDINAL, -- number of readlocks set by FindCacheEntry
    body: SELECT COMPUTED CacheType FROM
        file => [fd: Descriptor],
        pageGroup => [group: PageGroup, fileCacheEntry: CEptr],
        ENDCASE];

CEptr: TYPE = POINTER TO CacheEntry;

CacheCleanUpProc: TYPE = PROCEDURE [cePtr: CEptr];

NoCleanUp: CacheCleanUpProc; -- null case indicator (unbound control link)

returned: CONDITION;

-- Key for cache searches --

CacheKey: TYPE = RECORD[
    SELECT OVERLAID CacheType FROM
        file => [fileID: File.ID],
        pageGroup => [fileCacheEntry: CEptr, filePage: File.PageNumber],
        ENDCASE];

-- File descriptor cache --

FCache: Cache;
FCacheRecord: CacheRecord;
FCacheArray: ARRAY FCacheIndex OF file CacheEntry;
FCacheIndex: TYPE = [0..FCacheSize);
FCacheSize: CARDINAL = 32; -- must be 2 or greater
FCacheCleanUp: INTERNAL CacheCleanUpProc =
    BEGIN -- flush corresponding entries from page group cache
    WHILE RemoveCacheEntry[PGCache, cePtr] DO NULL ENDLOOP
    END;

-- Page Group cache --

PGCache: Cache;
PGCacheRecord: CacheRecord;
PGCacheArray: ARRAY PGCacheIndex OF pageGroup CacheEntry;
PGCacheIndex: TYPE = [0..PGCacheSize);
PGCacheSize: CARDINAL = 32; -- must be 2 or greater
```

```
GetFilePtrs: PUBLIC ENTRY PROCEDURE [count: CARDINAL, fileID: File.ID] RETURNS [success: BOOLEAN, fD: POINTER TO
Descriptor] =
    BEGIN
    fEntry: CEptr;
    [success, fEntry] ← FindCacheEntry[get, count, FCache, CacheKey[file[fileID]]];
    IF success THEN WITH fEntry SELECT file FROM file => fD ← @fd; ENDCASE;
    END;

ReturnFilePtrs: PUBLIC ENTRY PROCEDURE [count: CARDINAL, fD: POINTER TO Descriptor] =
    BEGIN
    [] ← FindCacheEntry[return, count, FCache, CacheKey[file[fD.fileID]]];
    END;

SetFile: PUBLIC ENTRY PROCEDURE [fd: Descriptor, pinned: BOOLEAN] =
    BEGIN
    SetCacheEntry[FCache, CacheEntry[NIL, IF pinned THEN 1 ELSE 0, file[fd]]]
    END;

FlushFile: PUBLIC ENTRY PROCEDURE [fileID: File.ID] =
    BEGIN
    found: BOOLEAN;
    fEntry: CEptr;
    [found, fEntry] ← FindCacheEntry[locate, 0, FCache, CacheKey[file[fileID]]];
    IF found THEN []←RemoveCacheEntry[FCache,fEntry];
    END;

GetPageGroup: PUBLIC ENTRY PROCEDURE [fileID: File.ID, filePage: File.PageNumber]
  RETURNS [success: BOOLEAN, pg: PageGroup] =
    BEGIN
    fEntry, pgEntry: CEptr;
    [success, fEntry] ← FindCacheEntry[locate, 0, FCache, CacheKey[file[fileID]]];
    IF ~success THEN RETURN;
    [success, pgEntry] ← FindCacheEntry[locate, 0, PGCache, CacheKey[pageGroup[fEntry, filePage]]];
    IF success THEN WITH pgEntry SELECT pageGroup FROM pageGroup => pg ← group; ENDCASE;
    END;

SetPageGroup: PUBLIC ENTRY PROCEDURE [fileID: File.ID, group: PageGroup, pinned: BOOLEAN] =
    BEGIN
    success: BOOLEAN;
    fEntry: CEptr;
    [success, fEntry] ← FindCacheEntry[locate, 0, FCache, CacheKey[file[fileID]]];
    IF ~success THEN ERROR;
    SetCacheEntry[PGCache, CacheEntry[NIL, IF pinned THEN 1 ELSE 0, pageGroup[group, fEntry]]];
    END;
```

```
FindCacheEntry: INTERNAL PROCEDURE [op: {get, return, locate}, count: CARDINAL, cache: Cache, key: CacheKey] RETURNS
[success: BOOLEAN, ceptr: CEptr] =

-- FindCacheEntry searches for an entry in the indicated cache having the indicated key, and returns the specified number of
   pointers to it (i.e. a pointer that may be copied that many times). If the pointers will be passed outside the monitor, op = get,
   and a later matching call with op = return must occur when the client discards the pointers; these adjust the refCnt of the
   entry appropriately. If only the contents of the entry are passed outside the monitor, op = locate, and the refCnt is not
   incremented.

    BEGIN OPEN cache;
    current: CEptr ← mru; -- start search with most-recently-used
    previous: CEptr;
    WHILE current ~= NIL DO OPEN current;
        IF WITH current SELECT cacheType FROM
            file => key.fileID = fd.fileID,
            pageGroup => key.fileCacheEntry = fileCacheEntry AND
                key.filePage >= group.filePage AND key.filePage < group.nextFilePage,
            ENDCASE => FALSE -- (never happens)
        THEN -- found the matching cache entry
            BEGIN
            IF current ~= mru THEN -- promote it to mru
                BEGIN previous.next ← next; next ← mru; mru ← current END;
            SELECT op FROM
                get => refCnt ← refCnt+count;
                return => IF (refCnt ← refCnt-count) = 0 THEN BROADCAST returned;
                ENDCASE;
            RETURN [TRUE, current]
            END;
        previous ← current; current ← next; -- advance down LRU chain
        ENDLOOP;
    RETURN [FALSE, NIL]; -- search failed
    END;

SetCacheEntry: INTERNAL PROCEDURE [cache: Cache, newEntry: CacheEntry] =
    BEGIN OPEN cache;
    found: BOOLEAN;
    p, entry, previous: CEptr;
    key: CacheKey;
    WITH newEntry SELECT cacheType FROM
        file => key ← CacheKey[file[fd.fileID]];
        pageGroup => key ← CacheKey[pageGroup[fileCacheEntry, group.filePage]];
        ENDCASE;
    [found, entry] ← FindCacheEntry[locate, 0, cache, key];
    IF ~found THEN -- must find a slot in the cache for the new entry
        BEGIN
        IF free ~= NIL THEN BEGIN entry ← free; free ← free.next; entry.refCnt ← 0; END -- try to use a free entry
        ELSE -- replace lru entry (can't be same as mru since free-chain known empty)
            BEGIN
            p ← mru;
            WHILE p.next ~= NIL DO
                IF p.next.refCnt = 0 THEN BEGIN entry ← p.next; previous ← p END; -- eligible for replacement
                p ← p.next;
                ENDLOOP;
            IF entry = NIL THEN ERROR; -- cache full of pinned entries!
            IF CleanUp ~= NoCleanUp THEN CleanUp[entry]; -- clean up for removal
            previous.next ← entry.next; -- remove last eligible entry from lru chain
            END;
        entry.next ← mru; mru ← entry; -- splice entry in as head of lru chain
        END;
    entry.refCnt ← entry.refCnt+newEntry.refCnt;
    WITH newEntry SELECT cacheType FROM
        file => entry.body ← file[fd];
        pageGroup => entry.body ← pageGroup[group, fileCacheEntry];
        ENDCASE;
    END;

RemoveCacheEntry: INTERNAL PROCEDURE [cache: Cache, fce: CEptr] RETURNS [success: BOOLEAN] =
    BEGIN OPEN cache;
    current, previous: CEptr;
    DO -- Note: This loop will never terminate if attempting to remove a pinned cache entry !
        current ← mru; -- start search with most-recently-used
        WHILE current ~= NIL DO
            IF WITH current SELECT cacheType FROM
```

```
            file => current = fce,
            pageGroup => fileCacheEntry = fce,
            ENDCASE => FALSE -- (never happens)
          THEN EXIT;
          previous ← current; current ← current.next; -- advance down LRU chain
          REPEAT
          FINISHED => RETURN [FALSE]; -- search failed
          ENDLOOP;
      IF current.refCnt = 0 THEN EXIT;
      WAIT returned; -- entry busy: try again when activity subsides
      ENDLOOP;
  IF CleanUp ~= NoCleanUp THEN CleanUp[current]; -- clean up entry for removal
  IF current = mru THEN mru ← current.next ELSE previous.next ← current.next; -- remove from lru chain
  current.next ← free; free ← current; -- put on free chain
  RETURN [TRUE];
  END;
```

```
-- Initialization of file caches --

i: CARDINAL;

-- Process.DisableAbort[@returned];

FCache ← @FCacheRecord;
FCache.cacheType ← file;
FCache.mru ← NIL;
FCache.free ← @FCacheArray[FIRST[FCacheIndex]];
FCache.CleanUp ← FCacheCleanUp;
FOR i IN FCacheIndex DO -- set up free chain
    FCacheArray[i].next ← IF i = LAST[FCacheIndex] THEN NIL ELSE @FCacheArray[i + 1]
    ENDLOOP;

PGCache ← @PGCacheRecord;
PGCache.cacheType ← pageGroup;
PGCache.mru ← NIL;
PGCache.free ← @PGCacheArray[FIRST[PGCacheIndex]];
PGCache.CleanUp ← NoCleanUp;
FOR i IN PGCacheIndex DO -- set up free chain
    PGCacheArray[i].next ← IF i = LAST[PGCacheIndex] THEN NIL ELSE @PGCacheArray[i + 1]
    ENDLOOP;

END.
```

LOG

Time: April 20, 1978 3:09 PM By: Redell Action: Created file
Time: May 4, 1978 1:32 PM By: Redell Action: bug: get free entry => refCnt ← 0
Time: July 25, 1978 1:31 PM By: Redell Action: clean crash if cache fills up with pinned entries.
Time: September 11, 1978 5:42 PM By: Redell Action: Fixed flush to wait if I/O in progress. Re-did lost edits to expand cache size to 32 entries(!)

```
-- FilerControl.mesa (last edited by: Redell on: March 7, 1979  4:45 PM)

DIRECTORY
    FilerPrograms: FROM "FilerPrograms",
    StoragePrograms: FROM "StoragePrograms";

FilerControl: PROGRAM
    IMPORTS FilerPrograms
    EXPORTS StoragePrograms =

BEGIN OPEN FilerPrograms;
START FileCacheImpl;
START FilerExceptionImpl;
START FilerTransferImpl;
START FileTaskImpl;
START RemotePageTransferImpl;
START SubVolumeImpl;

END.
```

LOG

Time: February 28, 1979 11:39 AM Action: Created file from old InitializeFPT.mesa

```
DIRECTORY
    FilePageTransfer: FROM "FilePageTransfer" USING [Request],
    FilerException: FROM "FilerException",
    FilerPrograms: FROM "FilerPrograms",
    Process: FROM "Process" --USING [DisableAborts]--;

FilerExceptionImpl: MONITOR
        --IMPORTS Process
        EXPORTS FilerException, FilerPrograms =
BEGIN

FET: ARRAY FETindex OF FETentry; -- Filer Exception Table
FETentry: TYPE = RECORD [
    state: {free, report},
    req: FilePageTransfer.Request];
FETindex: TYPE = [0..FETsize);
FETsize: CARDINAL = 10;
report, free: CONDITION;
i: CARDINAL;

Await: PUBLIC ENTRY PROCEDURE RETURNS [FilePageTransfer.Request] =
    -- Waits for a Filer exception to occur
    BEGIN
    DO
        FOR i IN FETindex DO OPEN FET[i];
            IF state = report THEN
                BEGIN state ← free; NOTIFY free; RETURN[req] END;
            ENDLOOP;
        WAIT report;
        ENDLOOP;
    END;

Report: PUBLIC ENTRY PROCEDURE [rwReq: FilePageTransfer.Request] =
    -- Reports a Filer exception and describes the remaining work (rwReq)
    BEGIN
    DO
        FOR i IN FETindex DO OPEN FET[i];
            IF state = free THEN
                BEGIN req ← rwReq; state ← report; NOTIFY report; RETURN END;
            ENDLOOP;
        WAIT free;
        ENDLOOP;
    END;

-- Initialize Filer Exception Table --

--Process.DisableAborts[@report];
--Process.DisableAborts[@free];

FOR i IN FETindex DO FET[i].state ← free ENDLOOP;

END.
```

LOG

Time: February 27, 1979 9:51 PM By: Redell Action: Created file from old CacheMissImpl.mesa
Time: timeStamp By: yourName Action: shortDescription

-- **FilerPrograms**.mesa *(last edited by: Redell on: March 7, 1979 3:15 PM)* --

```
FilerPrograms: DEFINITIONS =
BEGIN

FileCacheImpl: PROGRAM;

FilerExceptionImpl: PROGRAM;

FilerTransferImpl: PROGRAM;

FileTaskImpl: PROGRAM;

RemotePageTransferImpl: PROGRAM;

SubVolumeImpl: PROGRAM;

END.
```

LOG

Time: February 27, 1979 10:07 PM By: Redell Action: Created file from old InitializeFPT.mesa

```
-- FilerTransferImpl.mesa (last edited by: Redell on: August 19, 1979  1:51 PM)

--Things to consider:
-- 1) Large XWire request (>Transfer.maxConcurrency) will hang forever; should split up as with label requests (must
        deal with multiple processes however...simply making this module a monitor would seem to introduce
        deadlocks ...?)
-- 2) Main routines for data and label transfers should be combined. Note that combined routine will be an external
        procedure and will execute within monitor for label transfers but outside it for data...

DIRECTORY
    DiskChannel: FROM "DiskChannel" USING [Address, CompletionHandle, Create, CreateCompletionObject, Delete, Drive,
      Handle, InitiateIO, IORequest, Label, PLabel, WaitAny],
    Environment: FROM "Environment" USING [PageNumber, wordsPerPage],
    File: FROM "File" USING [PageCount, PageNumber],
    FileCache: FROM "FileCache" USING [GetFilePtrs, GetPageGroup, ReturnFilePtrs],
    FileInternal: FROM "FileInternal" USING [Descriptor, FilePtr, PageGroup],
    FilePageTransfer: FROM "FilePageTransfer" USING [Request],
    FilePageLabel: FROM "FilePageLabel" USING [Label],
    FilerException: FROM "FilerException" USING [Report],
    FilerPrograms: FROM "FilerPrograms",
    FileTask: FROM "FileTask" USING [LabelWait, maxConcurrency, XWireStart],
    Inline: FROM "Inline" USING [LongCOPY],
    LabelTransfer: FROM "LabelTransfer" USING [LabelType, Operation],
    RemotePageTransfer: FROM "RemotePageTransfer" USING [Initiate, Suggest],
    ResidentMemory: FROM "ResidentMemory" USING [Allocate],
    SubVolume: FROM "SubVolume" USING [Find, GetPageAddress, Handle, PageNumber, StartIO],
    System: FROM "System" USING [nullNetworkAddress],
    SystemInternal: FROM "SystemInternal" USING [altoFPSeries, UniversalID],
    Utilities: FROM "Utilities" USING [ShortCARDINAL],
    VolumeInternal: FROM "VolumeInternal" USING [altoVolume, PageNumber];

FilerTransferImpl: MONITOR
    IMPORTS
      DiskChannel,
      FileCache,
      FilerException,
      FileTask,
      Inline,
      RemotePageTransfer,
      ResidentMemory,
      SubVolume,
      Utilities
    EXPORTS FilePageTransfer, FilerPrograms, LabelTransfer
    SHARES File, SystemInternal =

BEGIN

Initiate: PUBLIC PROCEDURE [req: FilePageTransfer.Request] =
    BEGIN OPEN FilePageTransfer, req;
    fileFound, groupFound: BOOLEAN;
    fileP: FileInternal.FilePtr;
    fileD: FileInternal.Descriptor;
    rReq: Request ← req;
    group: FileInternal.PageGroup;
    IF count = 0 THEN RETURN;
    IF LOOPHOLE[file.fID, SystemInternal.UniversalID].series = SystemInternal.altoFPSeries THEN -- Alto files are special remote files
        BEGIN
        fileP ← @fileD;
        fileD ← FileInternal.Descriptor[file.fID, VolumeInternal.altoVolume, remote[]];
        END
    ELSE
        BEGIN
        [fileFound, fileP] ← FileCache.GetFilePtrs[count, file.fID];
        IF ~fileFound THEN BEGIN FilerException.Report[req]; RETURN END;
        END;
    WHILE count > 0 DO -- handle each requested file page
        WITH fileP SELECT FROM
            local =>
                BEGIN
                svH: SubVolume.Handle;
                svFound: BOOLEAN;
                svPage: SubVolume.PageNumber;
                [groupFound, group] ← FileCache.GetPageGroup[file.fID, filePage];
                IF ~groupFound THEN
```

-- **FilerTransferImpl**.mesa *(last edited by: Redell on: August 19, 1979 1:51 PM)*

```
                    BEGIN FileCache.ReturnFilePtrs[count, fileP]; FilerException.Report[req]; EXIT END;
                [svFound, svH] ← SubVolume.Find[fileP.volumeID, group.volumePage];
                svPage ← group.volumePage-svH.lvPage + (filePage-group.filePage);
                SubVolume.StartIO[operation, svH, svPage, memoryPage, fileP, filePage];
                END;
            remote =>
                BEGIN
                FileTask.XWireStart[memoryPage, fileP];
                RemotePageTransfer.Suggest[[operation, file.fID, filePage, memoryPage]]; -- Alto file hack: remove later
                END;
            ENDCASE;
        filePage ← filePage + 1; memoryPage ← memoryPage + 1; count ← count-1;
        ENDLOOP;
    WITH fileP SELECT FROM remote => RemotePageTransfer.Initiate[rReq, System.nullNetworkAddress]; ENDCASE;
    END;


-- Start of monitor implementing LabelTransfer interface

-- Empty page is source of zeros (sink for garbage) when writing (reading) labels.
emptyPagePtr: LONG POINTER = ResidentMemory.Allocate[hyperspace, 1];
emptyPage: Environment.PageNumber = Utilities.ShortCARDINAL[LOOPHOLE[emptyPagePtr, LONG
CARDINAL]/Environment.wordsPerPage];
completion: DiskChannel.CompletionHandle ← DiskChannel.CreateCompletionObject[];

ReadLabel: PUBLIC ENTRY PROCEDURE [
  file: FileInternal.Descriptor,
  filePage: File.PageNumber,
  volumePage: VolumeInternal.PageNumber] RETURNS [label: FilePageLabel.Label] =
    BEGIN
    pageGroup: FileInternal.PageGroup ← [filePage, volumePage, filePage + 1];
    [label,] ← Perform[readLabel, @file, @pageGroup, emptyPage];
    END;

ReadRootLabel: PUBLIC ENTRY PROCEDURE [
  drive: DiskChannel.Drive,
  rootPage: VolumeInternal.PageNumber] RETURNS [label: FilePageLabel.Label, labelValid: BOOLEAN] =
    BEGIN OPEN DiskChannel;
    rootChannel: Handle ← Create[drive, completion]; -- temporary channel for root page
    request: IORequest;
    pLabel: PLabel;
    labelStorage: ARRAY [0..SIZE[Label] + 3) OF WORD; -- space for label plus extra for quad alignment
    pLabel ← LOOPHOLE[((LOOPHOLE[LONG[@labelStorage[0]], LONG INTEGER] + 3)/4)*4]; --quad-align label
    BEGIN OPEN request; -- Can't use constructor because of blasted PRIVATE fields...
    channel ← rootChannel;
    diskPage ← rootPage;
    memoryPage ← emptyPage;
    count ← 1;
    direction ← get;
    label ← pLabel;
    verifyLabel ← 0;
    END;

    InitiateIO[@request];

    labelValid ← (WaitAny[completion].status = goodCompletion);
    Delete[rootChannel];
    label ← LOOPHOLE[pLabel↑];
    END;

WriteLabels: PUBLIC ENTRY PROCEDURE [
  file: FileInternal.Descriptor,
  pageGroup: FileInternal.PageGroup] =
    BEGIN OPEN pageGroup;
    emptyPagePtr↑ ← 0;
    Inline.LongCOPY[emptyPagePtr, Environment.wordsPerPage-1, emptyPagePtr + 1];
    [] ← Perform[writeLabel, @file, @pageGroup, emptyPage];
    END;

VerifyLabels: PUBLIC ENTRY PROCEDURE [
  file: FileInternal.Descriptor,
  pageGroup: FileInternal.PageGroup] RETURNS [labelsValid: BOOLEAN] =
    BEGIN OPEN pageGroup;
    [,labelsValid] ← Perform[verifyLabel, @file, @pageGroup, emptyPage];
```

-- **FilerTransferImpl**.mesa *(last edited by: Redell on: August 19, 1979 1:51 PM)*

```
    END;

ReadLabelAndData: PUBLIC ENTRY PROCEDURE [
  file: FileInternal.Descriptor,
  filePage: File.PageNumber,
  volumePage: VolumeInternal.PageNumber,
  memoryPage: Environment.PageNumber] RETURNS [label: FilePageLabel.Label] =
    BEGIN
    pageGroup: FileInternal.PageGroup ← [filePage, volumePage, filePage + 1];
    [label,] ← Perform[readLabelAndData, @file, @pageGroup, memoryPage];
    END;

WriteLabelsAndData: PUBLIC ENTRY PROCEDURE [
  file: FileInternal.Descriptor,
  pageGroup: FileInternal.PageGroup,
  memoryPage: Environment.PageNumber,
  type: LabelTransfer.LabelType ← normal,
  lastLink: DiskChannel.Address ← NULL] =
    BEGIN
    [] ← Perform[writeLabelsAndData, @file, @pageGroup, memoryPage, type, lastLink];
    END;

Perform: INTERNAL PROCEDURE [ -- Should be combined with Initiate (above)
  operation: LabelTransfer.Operation,
  filePtr: FileInternal.FilePtr,
  pageGroupPtr: POINTER TO FileInternal.PageGroup,
  memoryPage: Environment.PageNumber,
  labelType: LabelTransfer.LabelType ← normal,
  lastLink: DiskChannel.Address ← NULL]
  RETURNS [label: FilePageLabel.Label, labelsValid: BOOLEAN] =
    BEGIN OPEN pageGroupPtr;
    count: File.PageCount ← nextFilePage-filePage;
    subVolume: SubVolume.Handle;
    subVolumeFound: BOOLEAN;
    batchSize: CARDINAL;
    thisLabelValid: BOOLEAN;
    [subVolumeFound, subVolume] ← SubVolume.Find[filePtr.volumeID, volumePage];
    IF ~subVolumeFound THEN ERROR; -- local volume must be found
    WITH filePtr SELECT FROM
        remote => ERROR; -- can't transfer remote labels
        local =>
            BEGIN
            i: CARDINAL;
            link: DiskChannel.Address;
            labelsValid ← TRUE;
            WHILE count > 0 DO -- handle each requested label transfer
                batchSize ← MIN[Utilities.ShortCARDINAL[count], FileTask.maxConcurrency];
                FOR i IN [1..batchSize] DO -- start a batch
                    IF labelType = chained THEN link ← IF i = count THEN lastLink ELSE
                      SubVolume.GetPageAddress[filePtr.volumeID, volumePage + 1].address;
                    SubVolume.StartIO[operation, subVolume, volumePage, memoryPage, filePtr, filePage,
                      labelType = chained, link];
                    filePage ← filePage + 1; volumePage ← volumePage + 1;
                    IF operation = writeLabelsAndData THEN memoryPage ← memoryPage + 1;
                    ENDLOOP;
                THROUGH [1..batchSize] DO -- wait for the batch to finish
                    [label, thisLabelValid] ← FileTask.LabelWait[];
                    labelsValid ← labelsValid AND thisLabelValid;
                    ENDLOOP;
                count ← count-batchSize;
                ENDLOOP;
            END;
        ENDCASE;
    END;

END.
```

LOG

Time: February 28, 1979 9:28 AM By: Redell Action: Created file by merging old FilePageTransferImpl.mesa and

-- **FilerTransferImpl**.mesa *(last edited by: Redell on: August 19, 1979 1:51 PM)*

LabelTransferImpl.mesa
Time: March 21, 1979 12:01 PM By: Redell Action: Converted to Mesa 5.0
Time: August 13, 1979 3:59 PM By: Redell Action: Added interim boot-chain machinery; must be cleaned up and pushed down into disk driver.

```
DIRECTORY
    DiskChannel: FROM "DiskChannel" USING [IORequestHandle],
    Environment: FROM "Environment" USING [PageNumber],
    FileInternal: FROM "FileInternal" USING [FilePtr, Operation],
    FilePageLabel: FROM "FilePageLabel" USING [Label];

FileTask: DEFINITIONS =

BEGIN

maxConcurrency: CARDINAL = 40; -- maximum File Tasks (oustanding page transfers) that can exist at any one time.

DiskStart: PROCEDURE [mPage: Environment.PageNumber, fPtr: FileInternal.FilePtr, operation: FileInternal.Operation] RETURNS
  [DiskChannel.IORequestHandle];

DiskFinish: PROCEDURE [reqH: DiskChannel.IORequestHandle, labelValid: BOOLEAN];

XWireStart: PROCEDURE [mPage: Environment.PageNumber, fPtr: FileInternal.FilePtr] ; --RETURNS [??];

XWireFinish: PROCEDURE [ mPage: Environment.PageNumber --other args--];

LabelWait: PROCEDURE RETURNS [label: FilePageLabel.Label, labelValid: BOOLEAN];

END.
```

LOG

Time: February 28, 1979 10:35 AM By: Redell Action: Created file from old Transfer.mesa
Time: March 22, 1979 5:51 PM By: Redell Action: Increased size of RequestCell from 12 to 14 words.
Time: July 31, 1979 6:25 PM By: Gobbel Action: Increased size of RequestCell from 14 to 17 words.
Time: August 17, 1979 4:09 PM By: Redell Action: Changed to use FilePageLabel and DiskChannel.

```
DIRECTORY
    DiskChannel: FROM "DiskChannel" USING [IORequest, IORequestHandle, Label],
    Environment: FROM "Environment" USING [PageNumber],
    FilePageTransfer: FROM "FilePageTransfer",
    FilePageLabel: FROM "FilePageLabel" USING [Label],
    FileInternal: FROM "FileInternal" USING [FilePtr, Operation],
    FileCache: FROM "FileCache" USING [ReturnFilePtrs],
    FilerPrograms: FROM "FilerPrograms",
    FileTask: FROM "FileTask",
    LabelTransfer: FROM "LabelTransfer" USING [Operation],
    --Process: FROM "Process" USING [DisableAborts],
    RuntimeInternal: FROM "RuntimeInternal" USING [WorryCallDebugger];

FileTaskImpl: MONITOR
    IMPORTS FileCache, --Process,-- RuntimeInternal
    EXPORTS FileTask, FilePageTransfer, FilerPrograms =

BEGIN
ErrorHalt: PROCEDURE = BEGIN RuntimeInternal.WorryCallDebugger["Error in FileTaskImpl"L]; END;
FTT: ARRAY [0..FTTsize) OF FTTentry; -- File Task Table

FTTlabelStorage: ARRAY [0..FTTsize] OF DiskChannel.Label; -- parallel table of labels (w/ extra slot to align)
FTTlabels: POINTER TO ARRAY OF DiskChannel.Label =
    LOOPHOLE[((LOOPHOLE[@FTTlabelStorage[0], CARDINAL] + 3)/4)*4]; -- quadword aligned
FTTindex: TYPE = [0..FTTsize);
FTTsize: CARDINAL = FileTask.maxConcurrency;
FTTentry: TYPE = MACHINE DEPENDENT RECORD [ -- must be mult of quadword to retain alignment
    operationClass: OperationClass,
    state: {pending, complete, free},
    labelError: BOOLEAN,
    pad: [0..7777B],
    memPage: Environment.PageNumber,
    filePtr: FileInternal.FilePtr,
    var: SELECT OVERLAID * FROM
        disk => [request: DiskChannel.IORequest],
        xwire => NULL, --??--
        ENDCASE];
OperationClass: TYPE = {dataOperation, labelOperation};
complete, free: CONDITION;

DiskStart: PUBLIC ENTRY PROCEDURE [mPage: Environment.PageNumber, fPtr: FileInternal.FilePtr, operation: FileInternal.Operation]
  RETURNS [DiskChannel.IORequestHandle] =
    BEGIN
    i: CARDINAL;
    Error: PROCEDURE RETURNS [OperationClass];
    DO
        FOR i IN FTTindex DO WITH FTT[i] SELECT disk FROM disk =>
            IF state = free THEN
                BEGIN
                operationClass ← SELECT operation FROM
                    IN FilePageTransfer.Operation => dataOperation,
                    IN LabelTransfer.Operation => labelOperation,
                    ENDCASE => Error[];
                state ← pending;
                memPage ← mPage;
                filePtr ← fPtr;
                RETURN [@request]
                END;
            ENDCASE;
            ENDLOOP;
        WAIT free; -- no free slots; wait for one and try again
        ENDLOOP;
    END;

DiskFinish: PUBLIC ENTRY PROCEDURE [reqH: DiskChannel.IORequestHandle, labelValid: BOOLEAN] =
    BEGIN
    i: CARDINAL;
    FOR i IN FTTindex DO OPEN FTT[i];
        IF state = pending AND reqH = @request THEN
            BEGIN
            IF operationClass = dataOperation THEN
                IF labelValid THEN FileCache.ReturnFilePtrs[1, filePtr]
                ELSE RuntimeInternal.WorryCallDebugger["Unrecoverable disk error"]; -- Should be non-fatal somehow
            labelError ← ~labelValid;
```

Disk label

```
            state ← complete;
            BROADCAST complete;
            RETURN;
            END;
        ENDLOOP;
    ErrorHalt[]; -- never heard of this page transfer
    END;

XWireStart: PUBLIC --ENTRY-- PROCEDURE [mPage: Environment.PageNumber, fPtr: FileInternal.FilePtr] =
    BEGIN
    -- Real version must handle duplicates, retransmissions, etc. Needs much fancier key...
    [] ← DiskStart[mPage, fPtr, read]; -- this will do for now (for pseudo-remote Alto files)
    END;

XWireFinish: PUBLIC ENTRY PROCEDURE [mPage: Environment.PageNumber --other args--] =
    BEGIN
    -- Real version must handle duplicates, retransmissions, etc. Needs much fancier key... This version works only for pseudo-
    remote Alto-file transfers
    i: CARDINAL;
    FOR i IN FTTindex DO OPEN FTT[i];
        IF state = pending AND memPage = mPage THEN
            BEGIN
            -- FileCache.ReturnFilePtrs[1, filePtr]; ...disabled for Alto files, which aren't in file cache
            labelError ← FALSE;
            state ← complete;
            BROADCAST complete;
            RETURN;
            END;
        ENDLOOP;
    ErrorHalt[]; -- never heard of this page transfer
    END;

Wait: PUBLIC ENTRY PROCEDURE RETURNS [Environment.PageNumber] =
    BEGIN
    i: CARDINAL;
    DO
        FOR i IN FTTindex DO OPEN FTT[i];
            IF state = complete AND operationClass = dataOperation THEN
                BEGIN
                state ← free;
                NOTIFY free;
                RETURN[memPage]
                END;
            ENDLOOP;
        WAIT complete; -- no completed transfers: wait for some transfer to complete
        ENDLOOP;
    END;

LabelWait: PUBLIC ENTRY PROCEDURE RETURNS [label: FilePageLabel.Label, labelValid: BOOLEAN] =
    BEGIN
    i: CARDINAL;
    DO
        FOR i IN FTTindex DO OPEN FTT[i];
            IF state = complete AND operationClass = labelOperation THEN
                BEGIN
                labelValid ← ~labelError;
                label ← LOOPHOLE[request.label↑, FilePageLabel.Label];
                state ← free;
                NOTIFY free;
                RETURN;
                END;
            ENDLOOP;
        WAIT complete; -- no completed transfers: wait for some transfer to complete
        ENDLOOP;
    END;

-- Initialize Page Transfer Table --

BEGIN
i: CARDINAL;

--Process.DisableAborts[@complete];
--Process.DisableAborts[@free];
```

```
FOR i IN FTTindex DO
    FTT[i].state ← free;
    FTT[i].request.label ← @FTTlabels[i];
    ENDLOOP;
END

END.
```

LOG

Time: February 28, 1979 11:33 AM By: Redell Action: Created file from old TransferImpl.mesa
Time: March 7, 1979 6:04 PM By: Redell Action: Converted to Mesa 5.0; removed reference to old FilePageTransferInternal.
Time: March 26, 1979 4:59 PM By: Redell Action: Added check for label error on data transfer (For now, fatal error => Call Debugger).
Time: August 20, 1979 12:19 PM By: Redell Action: Removed LOOPHOLEs to DiskChannel types.

```
DIRECTORY
    Environment: FROM "Environment" USING [PageNumber],
    FilePageTransfer: FROM "FilePageTransfer" USING [Operation, Request],
    File: FROM "File" USING [ID, PageNumber],
    System: FROM "System" USING [NetworkAddress];

RemotePageTransfer: DEFINITIONS =

BEGIN

PageOp: TYPE = RECORD[
    op: FilePageTransfer.Operation,
    file: File.ID,
    fPage: File.PageNumber,
    mPage: Environment.PageNumber];

Suggest: PROCEDURE [sPop: PageOp];

Initiate: PROCEDURE [FilePageTransfer.Request, System.NetworkAddress];

END.
```

LOG

Time: March 20, 1978 9:16 AM By: Redell Action: Created file
Time: September 22, 1978 4:08 PM By: Redell Action: Added 'Suggest' mechanism to eliminate Alto file bottleneck.
Time: March 1, 1979 10:56 AM By: Redell Action: Changed name from "Server" to "RemotePageTransfer".

```
DIRECTORY
    AltoDiskController: FROM "AltoDiskController" USING [Acquire, Release],
    AltoDefs: FROM "AltoDefs" USING [BYTE, MaxFilePage, PageNumber],
    AltoFileDefs: FROM "AltoFileDefs" USING [FA, FP, LD, NullFP, vDA, vDC],
    AltoDisk: FROM "AltoDisk" USING [DiskCheckError, DiskPageDesc, DiskRequest, SwapPages],
    Environment: FROM "Environment" USING [Word, wordsPerPage],
    File: FROM "File" USING [PageNumber],
    FilePageTransfer: FROM "FilePageTransfer" USING [Operation, Request],
    FilerPrograms: FROM "FilerPrograms",
    FileTask: FROM "FileTask" USING [XWireFinish],
    Process: FROM "Process" USING [Detach, --DisableAborts,-- SetTimeout],
    RemotePageTransfer: FROM "RemotePageTransfer" USING [PageOp],
    Runtime: FROM "Runtime" USING [CallDebugger],
    System: FROM "System" USING [NetworkAddress],
    Utilities: FROM "Utilities" USING [ShortCARDINAL],
    VM: FROM "VM" USING [PageNumber, PageOffset];

RemotePageTransferImpl: MONITOR
    IMPORTS
        AltoDiskController,
        AltoDisk,
        FileTask,
        Process,
        Runtime,
        Utilities
    EXPORTS FilerPrograms, RemotePageTransfer
    SHARES File =
BEGIN
-- This module simulates communication with a page-level server by short-circuiting the entire remote swapping protocol and
        directly accessing Alto files via the low-level AltoDisk interface (adapted from the Alto/Mesa DiskDefs interface). It is a monitor
        to avoid any possible conflicts due to the non-reentrancy of AltoDiskImpl (adapted from the Alto/Mesa DiskIO module).
        RemotePageTransferImpl must be replaced (or modified) when real remote swapping is desired instead of (or in addition
        to) pseudo-remote access to Alto files.

hints: ARRAY [0..1] OF Hint ← [nullHint, nullHint];
Hint: TYPE = RECORD [
    fp: AltoFileDefs.FP,
    filePage: File.PageNumber,
    vDA: AltoFileDefs.vDA];
nullHint: Hint = [AltoFileDefs.NullFP, 0, [0]];
hOldest: CARDINAL ← 0;

workToDo, workDone: CONDITION;
wPop: RemotePageTransfer.PageOp;
wCount: CARDINAL;

Suggest: PUBLIC ENTRY PROCEDURE [sPop: RemotePageTransfer.PageOp] =
    BEGIN OPEN wPop; -- This is really just a hack for Alto files
    IF wCount > 0 THEN
        IF sPop = RemotePageTransfer.PageOp[op, file, fPage + wCount, mPage + wCount] THEN
            BEGIN wCount ← wCount + 1; RETURN; END
        ELSE
            WHILE wCount > 0 DO
                WAIT workDone
                ENDLOOP;
    wPop ← sPop; wCount ← 1;
    END;

Initiate: PUBLIC ENTRY PROCEDURE [req: FilePageTransfer.Request, netAddr: System.NetworkAddress] =
    BEGIN
    NOTIFY workToDo;
    END;

AltoFileProcess: ENTRY PROCEDURE =
    BEGIN OPEN wPop;
    offset: VM.PageOffset;
    AltoFileID: TYPE = RECORD[series: CARDINAL, fp: AltoFileDefs.FP];
    DO
        WHILE wCount = 0 DO
            WAIT workToDo
            ENDLOOP;
        AltoDiskController.Acquire[];
        TransferAltoPages[LOOPHOLE[file, AltoFileID].fp, fPage, mPage, wCount, op];
        AltoDiskController.Release[];
```

```
        FOR offset IN [0..wCount) DO
            FileTask.XWireFinish[mPage + offset];
            ENDLOOP;
        wCount ← 0;
        NOTIFY workDone;
        ENDLOOP;
    END;

TransferAltoPages: INTERNAL PROCEDURE [fp: AltoFileDefs.FP, filePage: File.PageNumber, memoryPage: VM.PageNumber, count:
        CARDINAL, operation: FilePageTransfer.Operation] =
    BEGIN
    firstFilePage: AltoDefs.PageNumber ← Utilities.ShortCARDINAL[filePage];
    lastFilePage: AltoDefs.PageNumber ← firstFilePage + count-1;
    firstDA: AltoFileDefs.vDA;
    page: AltoDefs.PageNumber;
    byte: AltoDefs.BYTE;
    memoryAddr: LONG POINTER = LOOPHOLE[LONG[memoryPage]*256];
    leaderPage: LONG POINTER TO AltoFileDefs.LD = LOOPHOLE[memoryAddr];
    dReq: swap AltoDisk.DiskRequest;
    dpd: AltoDisk.DiskPageDesc;
    Error: PROCEDURE RETURNS [AltoFileDefs.vDC];
    BEGIN ENABLE
        AltoDisk.DiskCheckError =>
            IF page = 0 AND filePage = 0 THEN
                BEGIN
                leaderPage.eofFA ← AltoFileDefs.FA[[0], 0, 0]; -- (tells file mgr that file doesn't exist)
                GO TO Return
                END
            ELSE
                DO Runtime.CallDebugger["Malformed Alto file: run the scavenger"] ENDLOOP;
    IF fp = hints[0].fp AND filePage = hints[0].filePage THEN
        firstDA ← hints[0].vDA
    ELSE IF fp = hints[1].fp AND filePage = hints[1].filePage THEN
        firstDA ← hints[1].vDA
    ELSE IF firstFilePage = 0 THEN
        firstDA ← fp.leaderDA
    ELSE
        BEGIN
        dReq ← AltoDisk.DiskRequest[
                ca: @scanBuffer,
                da: @fp.leaderDA,
                firstPage: 0,
                lastPage: firstFilePage-1,
                fp: @fp,
                fixedCA: TRUE,
                action: ReadD,
                lastAction: ,
                signalCheckError: TRUE,
                option: swap[@dpd]];
        [page,] ← AltoDisk.SwapPages[@dReq];
        IF page ~= firstFilePage-1 THEN ERROR; -- File too short
        firstDA ← dpd.next;
        END;
    dReq ← AltoDisk.DiskRequest[
            ca: memoryAddr,
            da: @firstDA,
            firstPage: firstFilePage,
            lastPage: lastFilePage,
            fp: @fp,
            fixedCA: FALSE,
            action: SELECT operation FROM read => ReadD, write => WriteD, ENDCASE => Error[],
            lastAction: ,
            signalCheckError: TRUE,
            option: swap[@dpd]];
    [page,] ← AltoDisk.SwapPages[@dReq];
    IF page ~= lastFilePage THEN ERROR; -- File too short
    IF filePage = 0 THEN -- (client reading leader page: check length hint)
        BEGIN
        pastEOF: BOOLEAN ← FALSE; -- Has bad EOF hint sent us diving off past end-of-file?
        hintDA: AltoFileDefs.vDA ← leaderPage.eofFA.da;
        eofFA: AltoFileDefs.FA;
        dReq ← AltoDisk.DiskRequest[
```

```
                ca: @scanBuffer,
                da: @hintDA,
                firstPage: leaderPage.eofFA.page, -- start at alleged end of file...
                lastPage: AltoDefs.MaxFilePage,        -- ...and read as much as there is.
                fp: @fp,
                fixedCA: TRUE,
                action: ReadD,
                lastAction: ,
                signalCheckError: TRUE,
                option: swap[@dpd]];
        [page, byte] ← AltoDisk.SwapPages[@dReq ! AltoDisk.DiskCheckError => BEGIN pastEOF ← TRUE; CONTINUE END];
        IF pastEOF THEN -- go back to beginning of file and read as much as there is
            BEGIN
            dReq ← AltoDisk.DiskRequest[
                        ca: @scanBuffer,
                        da: @firstDA,
                        firstPage: firstFilePage,
                        lastPage: AltoDefs.MaxFilePage,
                        fp: @fp,
                        fixedCA: TRUE,
                        action: ReadD,
                        lastAction: ,
                        signalCheckError: TRUE,
                        option: swap[@dpd]];
            [page, byte] ← AltoDisk.SwapPages[@dReq];
            END;
        eofFA ← AltoFileDefs.FA[dpd.this, page, byte]; -- truth about end of file
        IF leaderPage.eofFA ~= eofFA THEN
            BEGIN
            leaderPage.eofFA ← eofFA; -- fix hint
            dReq ← AltoDisk.DiskRequest[
                        ca: leaderPage,
                        da: @firstDA,
                        firstPage: firstFilePage,
                        lastPage: firstFilePage,
                        fp: @fp,
                        fixedCA: TRUE,
                        action: WriteD,
                        lastAction: ,
                        signalCheckError: TRUE,
                        option: swap[@dpd]];
            [page, byte] ← AltoDisk.SwapPages[@dReq]; -- should't ever get check error here...
            END
        END
    ELSE -- (enter hint in cache)
        BEGIN
        hints[hOldest] ← Hint[fp, filePage + count, dpd.next];
        hOldest ← (hOldest + 1) MOD 2;
        END
    END -- enable-block for DiskCheckError
    EXITS
        Return => RETURN;
    END;
scanBuffer: ARRAY [0..Environment.wordsPerPage) OF Environment.Word;

--Process.DisableAborts[@workToDo];
--Process.DisableAborts[@workDone];
Process.SetTimeout[@workToDo, 10];
wCount ← 0;
Process.Detach[FORK AltoFileProcess[]];

END.
```

LOG

Time: June 7, 1978 11:01 AM By: Redell Action: Created file
Time: June 14, 1978 6:30 PM By: Redell Action: Added call to Transfer.XWireFinish to complete each transfer.
Time: June 14, 1978 6:30 PM By: Redell Action: Added private scanBuffer for DiskIO label scan.

Time: July 21, 1978 11:37 AM By: Redell Action: Bug: could not access page 0.
Time: August 4, 1978 5:03 PM By: Redell Action: Added acquire/release of shared alto disk controller
Time: August 30, 1978 5:27 PM By: Redell Action: Added two-item disk address cache for Alto files (should catch code remapping and directory searching)
Time: September 1, 1978 4:56 PM By: McJones Action: Set signalCheckError ← TRUE
Time: September 2, 1978 11:11 AM By: McJones Action: Added count to hint filePage
Time: September 14, 1978 11:28 PM By: Redell Action: Changed to fix hint in leader page. Converted to LONG POINTER disk controller.
Time: September 22, 1978 3:57 PM By: Redell Action: Added extra process to allow arbitrary-sized Alto file transfers without danger of lockup.
Time: March 1, 1979 11:09 AM By: Redell Action: Changed name from "ServerImpl" to "RemotePageTransferImpl".
Time: March 1, 1979 11:09 AM By: Redell Action: Converted to Mesa 5.0.

```
--Things to consider:
--1) Taking subvolumes offline needs more synchronization. Can probably cover subvolumes with filecache
        reference counts since holder of a SubVolume.Handle usually holds a FilePtr for a file on the subvolume.
        Offline must then flush file cache before removing subvolume (allows pending I/O to complete).

DIRECTORY
    AltoDiskController: FROM "AltoDiskController",
    DiskChannel: FROM "DiskChannel" USING [Address, CompletionHandle, CreateCompletionObject, Direction,
      GetPageAddress, Handle, Idle, InitiateIO, IORequestHandle, Restart, WaitAny],
    Environment: FROM "Environment" USING [PageNumber],
    File: FROM "File" USING [PageNumber],
    FileInternal: FROM "FileInternal" USING [FilePtr, Operation],
    FilePageLabel: FROM "FilePageLabel" USING [Label, nullLabel, SetFilePage, SetType],
    FilerPrograms: FROM "FilerPrograms",
    FileTask: FROM "FileTask" USING [DiskFinish, DiskStart],
    PhysicalVolume: FROM "PhysicalVolume" USING [SubVolumeDesc],
    PilotFileTypes: FROM "PilotFileTypes" USING [tBootFile],
    Process: FROM "Process" USING [Detach],
    RuntimeInternal: FROM "RuntimeInternal" USING [WorryCallDebugger],
    SubVolume: FROM "SubVolume" USING [Descriptor, Handle, PageNumber],
    Volume: FROM "Volume" USING [ID],
    VolumeInternal: FROM "VolumeInternal" USING [PageNumber];

SubVolumeImpl: MONITOR
    IMPORTS DiskChannel, FilePageLabel, FileTask, Process, RuntimeInternal
    EXPORTS AltoDiskController, SubVolume, FilerPrograms =
BEGIN OPEN DiskChannel;

-- Preliminary version of subvolume cache
cacheSize: CARDINAL = 8; -- (2 subvolumes + physical Root Hack)*2 disks + 2 for fudge
CacheIndex: TYPE = [1..cacheSize];
cache: ARRAY CacheIndex OF CacheEntry;
CacheEntry: TYPE = RECORD [occupied: BOOLEAN, svDesc: SubVolume.Descriptor];
CePtr: TYPE = POINTER TO CacheEntry;
ceFirst: CePtr = @cache[FIRST[CacheIndex]];

ErrorHalt: PROCEDURE = BEGIN RuntimeInternal.WorryCallDebugger["Error in SubVolumeImpl"L]; END;

Find: PUBLIC ENTRY PROCEDURE [vID: Volume.ID, page: VolumeInternal.PageNumber] RETURNS [success: BOOLEAN, subVolume:
        SubVolume.Handle] =
    BEGIN
    [success, subVolume] ← FindSV[@vID, page]
    END;

FindSV: INTERNAL PROCEDURE [vID: POINTER TO Volume.ID, page: VolumeInternal.PageNumber] RETURNS [BOOLEAN,
        SubVolume.Handle] =
    INLINE
    BEGIN
    cePtr: CePtr ← ceFirst;
    svH: SubVolume.Handle;
    THROUGH CacheIndex DO
        IF cePtr.occupied THEN
            BEGIN
            svH ← @cePtr.svDesc;
            IF vID↑ = svH.lvID AND page IN [svH.lvPage..svH.lvPage + svH.nPages) THEN RETURN [TRUE, svH];
            END;
        cePtr ← cePtr + SIZE[CacheEntry];
        ENDLOOP;
    RETURN [FALSE, NIL]
    END;

GetPageAddress: PUBLIC ENTRY PROCEDURE [vID: Volume.ID, page: VolumeInternal.PageNumber] RETURNS [channel:
  DiskChannel.Handle, address: DiskChannel.Address] =
    BEGIN
    found: BOOLEAN;
    svH: SubVolume.Handle;
    [found, svH] ← FindSV[@vID, page];
    IF NOT found THEN ErrorHalt[];
    RETURN [svH.channel, DiskChannel.GetPageAddress[svH.channel, svH.pvPage + (page-svH.lvPage)]]
    END;
```

```
OnLine: PUBLIC ENTRY PROCEDURE [subVolume: PhysicalVolume.SubVolumeDesc, channel: DiskChannel.Handle] =
    BEGIN OPEN subVolume;
    cePtr: CePtr ← ceFirst;
    THROUGH CacheIndex DO OPEN cePtr;
        IF ~occupied THEN
            BEGIN
            occupied ← TRUE;
            svDesc ← SubVolume.Descriptor[
                        lvID: lvID,
                        lvPage: lvPage,
                        pvPage: pvPage,
                        nPages: nPages,
                        channel: channel];
            RETURN;
            END;
        cePtr ← cePtr + SIZE[CacheEntry];
        ENDLOOP;
    ErrorHalt[]; -- subvolume cache overflow
    END;

OffLine: PUBLIC ENTRY PROCEDURE [vID: Volume.ID] =
    BEGIN -- Flushes all subvolumes of a given logical volume
    cePtr: CePtr ← ceFirst;
    THROUGH CacheIndex DO OPEN cePtr;
        IF svDesc.lvID = vID THEN occupied ← FALSE;
        cePtr ← cePtr + SIZE[CacheEntry];
        ENDLOOP;
    END;

Acquire: PUBLIC ENTRY PROCEDURE =
    BEGIN -- Idles all Pilot subvolumes to allow Alto file I/O
    cePtr: CePtr ← ceFirst;
    THROUGH CacheIndex DO OPEN cePtr;
        IF occupied THEN Idle[svDesc.channel];
        cePtr ← cePtr + SIZE[CacheEntry];
        ENDLOOP;
    END;

Release: PUBLIC ENTRY PROCEDURE =
    BEGIN -- Restarts all Pilot subvolumes when Alto file I/O is completed
    cePtr: CePtr ← ceFirst;
    THROUGH CacheIndex DO OPEN cePtr;
        IF occupied THEN Restart[svDesc.channel];
        cePtr ← cePtr + SIZE[CacheEntry];
        ENDLOOP;
    END;

StartIO: PUBLIC --ENTRY-- PROCEDURE [op: FileInternal.Operation, subVolume: SubVolume.Handle, subVolumePage:
    SubVolume.PageNumber, memPage: Environment.PageNumber, filePtr: FileInternal.FilePtr, filePage: File.PageNumber, chained:
    BOOLEAN ← FALSE, link: DiskChannel.Address ← NULL] =
    BEGIN
    labelSize: CARDINAL = SIZE [FilePageLabel.Label];
    VerifyCount: TYPE = [0..labelSize];
    verify: VerifyCount ← labelSize;
    noVerify: VerifyCount = 0;
    Action: TYPE = RECORD [verifyLabel: VerifyCount, getOrPut: Direction];
    action: Action;
    ErrorHalt1: PROCEDURE RETURNS [Action] = LOOPHOLE[ErrorHalt];
    requestHandle: IORequestHandle;
    properLabel: FilePageLabel.Label ← FilePageLabel.nullLabel;
    properLabel.fileID ← filePtr.fileID;
    FilePageLabel.SetFilePage[@properLabel, filePage];
    IF chained THEN properLabel.bootChainLink ← LOOPHOLE[link];
    WITH filePtr SELECT FROM -- couldn't his be done at a higher level???
        local =>
            BEGIN
            FilePageLabel.SetType[@properLabel, type];
            IF type = PilotFileTypes.tBootFile THEN verify ← verify-2; -- can't verify bootchain link words
            IF filePage = 0 THEN
                BEGIN
                properLabel.immutable ← immutable;
                properLabel.temporary ← temporary;
                properLabel.zeroSize ← (size = 0);
```

```
DIRECTORY
    AltoDisk: FROM "AltoDisk" USING [DA, DL, DiskPageDesc, DiskRequest, RealDA, SwapPages, VirtualDA],
    AltoFileDefs: FROM "AltoFileDefs" USING [FP],
    Boot: FROM "Boot" USING [Location, LP, LVBootFiles, nullDiskFileID, VolumeType],
    BootSwap: FROM "BootSwap" USING [countSkip, GetPStartListHeader, GetSelf, Initialize, InitializeMDS, mdsiGerm],
    CommunicationPrograms: FROM "CommunicationPrograms" USING [CommunicationControl],
    ControlDefs: FROM "ControlDefs" USING [GlobalFrameHandle, StateVector, TraceOff, TrapLink],
    ControlPrograms: FROM "ControlPrograms" USING [SystemImpl],
    CoreSwapDefs: FROM "CoreSwapDefs" USING [PuntInfo],
    DebuggerSwap: FROM "DebuggerSwap" USING [DebuggerMicrocode, Parameters],
    DriverPrograms: FROM "DriverPrograms" USING [DriverControl, StartIODrivers],
    Environment: FROM "Environment" USING [maxPagesInMDS, PageCount, PageNumber, PageOffset, wordsPerPage],
    FacePrograms: FROM "FacePrograms" USING [FaceControl],
    File: FROM "File" USING [Capability, Create, ID, MakePermanent, nullID, Permissions, read, SetSize, write],
    FrameOps: FROM "FrameOps" USING [Free, GetReturnFrame, SetReturnLink],
    Inline: FROM "Inline" USING [BITAND, LongCOPY],
    KernelFile: FROM "KernelFile" USING [DeleteTemps, GetRootFile, Pin],
    Keys: FROM "Keys" USING [KeyBits],
    MiscPrograms: FROM "MiscPrograms" USING [PPDataImpl, ResidentHeapImpl, StreamImpl, UtilitiesImpl, ZoneImpl],
    Mopcodes: FROM "Mopcodes" USING [zLI4, zMISC, zRFS, zSHIFT, zWFS],
    PageMap: FROM "PageMap" USING [Assoc, SetF, Value, valueClean, valueVacant],
    PerformancePrograms: FROM "PerformancePrograms" USING [PilotCounter, PilotPerfMonitor],
    PilotClient: FROM "PilotClient" USING [Run],
    PilotMP: FROM "PilotMP" USING [Code, cBadBootFile, cClient, cCommunication, cDeleteTemps, cDrivers, cMap,
        cRuntime, cStorage],
    PilotFileTypes: FROM "PilotFileTypes" USING [tCodeFile],
    PPData: FROM "PPData" USING [Initialize, on, TurnOn],
    Process: FROM "Process" USING [Detach],
    ProcessInternal: FROM "ProcessInternal" USING [DisableInterrupts, EnableInterrupts],
    ResidentMemory: FROM "ResidentMemory",
    Runtime: FROM "Runtime" USING [CallDebugger],
    RuntimeInternal: FROM "RuntimeInternal" USING [Codebase, WorryCallDebugger],
    RuntimePrograms: FROM "RuntimePrograms" USING [Frames, Instructions, InterruptKey, PilotNub, Processes, Signals,
        SnapshotImpl, Start, Traps],
    SDDefs: FROM "SDDefs" USING [sFirstFree, SD],
    Space: FROM "Space" USING [Create, defaultWindow, Delete, Handle, MakeReadOnly, Map, Remap, WindowOrigin],
    SpecialSpace: FROM "SpecialSpace" USING [MakeSwappable],
    StartList: FROM "StartList" USING [Entry, FileIndex, Index, Header, NullFileIndex, SelfFileIndex, NullSpaceIndex,
        SpaceIndex, SpaceType, StartIndex, TableBase, VersionID],
    StoragePrograms: FROM "StoragePrograms",
    System: FROM "System" USING [UniversalID],
    SystemInternal: FROM "SystemInternal" USING [UniversalID, altoFPSeries],
    TrapOps: FROM "TrapOps" USING [WriteXTS],
    VMMapLog: FROM "VMMapLog" USING [Descriptor, EntryPointer, PatchTable, PatchTableEntry, PatchTableEntryPointer],
    PilotSwitches: FROM "PilotSwitches",
    Volume: FROM "Volume" USING [ID, SystemID];

PilotControl: PROGRAM
    IMPORTS AltoDisk, Boot, BootSwap, CommunicationPrograms, ControlPrograms, DebuggerSwap, FacePrograms,
        DriverPrograms, File, FrameOps, Inline, KernelFile, MiscPrograms, PageMap, PerformancePrograms, PilotClient,
        PPData, Process, ProcessInternal, Runtime, RuntimeInternal, RuntimePrograms, Space, SpecialSpace,
        StoragePrograms, TrapOps, Volume
    EXPORTS DebuggerSwap, PilotSwitches, ResidentMemory
    SHARES AltoDisk, ControlDefs, File, PageMap, RuntimePrograms, System, SystemInternal =
BEGIN OPEN Environment;

countVM: PageCount = 16384-256-256; -- leave room for germ, old saved map entries
pageSavedMap: PageNumber = countVM-1; -- only if altoDebugger (need an allocator for this)

  -- Options:
switches: PUBLIC Keys.KeyBits; -- exported through PilotSwitches
debuggerVolume: BOOLEAN; -- Pilot was bootLoaded from a debugger or a client volume?

  -- Parameters:
```

```
countResidentMemory: PageCount ← 30; -- for ResidentMemory implementation
countBuffer: PageCount; -- for SwapBuffer implementation (set just before key stop 0)
countThreshold: PageCount ← 4; -- for replacement algorithm (MStore.AwaitBelowThreshold)

  -- Assorted global variables:
h: POINTER TO StartList.Header; tableBase: StartList.TableBase;
pageFirstReal: PageNumber ← 0; -- should be field in StartList.Header
indexResidentMemory: StartList.SpaceIndex ← StartList.NullSpaceIndex;
fcSelf: File.Capability; -- file from which we were booted
countLoaded: PageCount; -- prefix of countVirtual allocated by StartPilot
pageGerm: PageNumber; -- the one table-compiled into Pilot (if any)
pageResidentMemory: PageNumber; -- currently within the first64K, which is also the MDS
pageBuffer: PageNumber;
  -- Exported to DebuggerSwap:
canSwap: PUBLIC BOOLEAN ← FALSE; -- can we get to the debugger
parameters: PUBLIC DebuggerSwap.Parameters;
altoGermCopy: ARRAY [0..wordsPerPage) OF UNSPECIFIED;
altoBootLabelCopy: AltoDisk.DL;

  -- Pilot code swapping file:
fcSwap: File.Capability;
countSwap: PageCount;
offsetSwap: PageOffset;

  -- Map log and patch table descriptor (see also CachedRegionImpl):
mapLogDescriptor: VMMapLog.Descriptor ← [
  self: ,
  writer: FIRST[VMMapLog.EntryPointer],
  reader: FIRST[VMMapLog.EntryPointer],
  limit: FIRST[VMMapLog.EntryPointer],
  patchTable: @dummyPatchTable.patchTable]; -- overwritten by CachedRegionImpl
DummyPatchTable: TYPE = RECORD [patchTable: VMMapLog.PatchTable, entries: ARRAY [0..5) OF VMMapLog.PatchTableEntry];
dummyPatchTable: DummyPatchTable ← [[
  limit: FIRST[VMMapLog.PatchTableEntryPointer],
  maxLimit: FIRST[VMMapLog.PatchTableEntryPointer] + 5*SIZE[VMMapLog.PatchTableEntry],
  entries: ], ];

-- Heart of PilotControl
ProcessStartList: PROCEDURE =
  -- Assume called from main body
  BEGIN
  pKeys: LONG POINTER TO Keys.KeyBits ← LOOPHOLE[LONG[177033B]];
  bootFile: disk Boot.Location;
  lvBootFiles: Boot.LVBootFiles;
  debugClass: Boot.VolumeType;
  debuggerDevice: UNSPECIFIED;
  offset: PageOffset;
  state: ControlDefs.StateVector;

pCursorXY: LONG POINTER TO MACHINE DEPENDENT RECORD [x, y: CARDINAL] = LOOPHOLE[LONG[426B]];
pCurmap: LONG POINTER TO ARRAY [0..16) OF UNSPECIFIED = LOOPHOLE[LONG[431B]];
pCursorXY↑ ← [512-8, 404-8];
pCurmap[0] ← 177777B; Inline.LongCOPY[from: @pCurmap[0], to: @pCurmap[1], nwords: 15];

  FrameOps.Free[FrameOps.GetReturnFrame[]];

  -- Get key switch settings passed by RunPilot if present else from keyboard
  BEGIN OPEN SDDefs--SD, sFirstFree--;
  IF Boot.LP[highbits: BootSwap.mdsiGerm, lowbits: @SD[sFirstFree]]↑~ = 0 THEN
    switches ← LOOPHOLE[Boot.LP[highbits: BootSwap.mdsiGerm, lowbits: @SD[sFirstFree]], LONG POINTER TO
Keys.KeyBits]↑
  ELSE WHILE pKeys.Space = up DO switches ← pKeys↑ ENDLOOP
  END;
  IF (debuggerVolume ← switches.D = down) THEN switches.A ← down; -- temporary
```

```
SetMP[PilotMP.cRuntime];

-- Get start traps working first
RuntimePrograms.Start[LOOPHOLE[RuntimePrograms.Traps]];

BootSwap.InitializeMDS[]; -- set pMon
h ← BootSwap.GetPStartListHeader[]; tableBase ← h.table;
BootSwap.GetSelf[@fcSelf.fID]; fcSelf.permissions ← LAST[File.Permissions];
countLoaded ← h.lastVMPage + 1;

IF h.version~ = StartList.VersionID THEN -- is it the version of StartList.mesa that we think it is?
  Punt[PilotMP.cBadBootFile];

EnumerateStartList[DescribePass0]; -- until StartPilot is changed

-- Initialize ResidentMemory to none available
FOR offset IN [0..256) DO SetBit[busy, @map, BD[offset]] ENDLOOP;

-- Initialize Mesa runtime
START RuntimePrograms.Frames;
START RuntimePrograms.Instructions;
FrameOps.SetReturnLink[ControlDefs.TrapLink]; -- so it can be set to Processes.End
START RuntimePrograms.Processes[pagePDA: h.pdaBase, countPDA: h.pdaPages];
START RuntimePrograms.Signals;
START RuntimePrograms.PilotNub[
  debuggerFile: FIDFromIndex[h.debuggerFile],
  coreFile: FIDFromIndex[h.coreFile],
  debuggerDA: h.debuggerDA,
  coreDA: h.coreDA,
  pageLoadState: tableBase[h.loadState].vmpage, countLoadState: tableBase[h.loadState].pages,
  pVMMapLog: @mapLogDescriptor];
--pageGerm ← PageFromPointer[Inline.LowHalf[RuntimeInternal.Codebase[LOOPHOLE[RuntimePrograms.Germ]]]];
--EnumerateStartList[InitializeGerm];
BootSwap.Initialize[mdsiOther: BootSwap.mdsiGerm];
InitializeDebuggerSwap[]; -- read in germ for debugger's world; initialize saved map entries
START RuntimePrograms.InterruptKey;

-- SnapshotImpl is started below after the virtual memory is running
START PerformancePrograms.PilotCounter;
START PerformancePrograms.PilotPerfMonitor;

-- Mesa runtime and debugger now fully operational

-- Set countBuffer to size of real memory (can be overriddent at key stop 0)
BEGIN page: PageNumber;
countBuffer ← 0;
FOR page IN [FIRST[PageNumber]..FIRST[PageNumber] + countVM) DO OPEN PageMap;
  IF Inline.BITAND[SetF[page, valueClean], valueVacant]~ = valueVacant THEN
    countBuffer ← countBuffer + 1
  ENDLOOP
END;

IF switches.Zero = down THEN Runtime.CallDebugger["Key Stop 0"];

-- Start OIS stuff (faces)
START FacePrograms.FaceControl;

-- Initialize ResidentMemory implementation (delay allows key stop 0 intervention)
EnumerateStartList[FindResidentMemory]; -- set indexResidentMemory
IF indexResidentMemory = StartList.NullSpaceIndex THEN Runtime.CallDebugger["MDS Full"L];
BEGIN OPEN tableBase[indexResidentMemory]; -- steal a piece of the space
pageResidentMemory ← vmpage;
vmpage ← vmpage + countResidentMemory; pages ← pages-countResidentMemory;
END;
StealRealMemory[page: pageResidentMemory, count: countResidentMemory];
FOR offset IN [0..countResidentMemory) DO
  SetBit[free, @map, BD[offset]]
  ENDLOOP;

-- Initialize SystemImpl
SetMP[PilotMP.cDrivers];
```

```
START ControlPrograms.SystemImpl;

-- Initialize Streams, Utilities, and Zone
START MiscPrograms.PPDataImpl;
START MiscPrograms.ZoneImpl;
START MiscPrograms.ResidentHeapImpl;
START MiscPrograms.UtilitiesImpl;
START MiscPrograms.StreamImpl;

-- Initialize Storage drivers
START DriverPrograms.DriverControl;

-- Initialize Storage

-- Initialize lower Storage

BEGIN OPEN StoragePrograms;

SetMP[PilotMP.cStorage];
START FilerControl;

IF countBuffer>countVM-countLoaded THEN RuntimeInternal.WorryCallDebugger["VM Full"L];
pageBuffer ← FIRST[PageNumber] + countLoaded; countLoaded ← countLoaded + countBuffer;
START SwapperControl[pageBuffer, countBuffer, @mapLogDescriptor];

-- Free extra real memory (so can no longer call StealRealMemory)
-- NOTE: We are careful not to free real memory behind pageSavedMap (if any)
DescribeSpace[free, pageFirstReal, --countVM--pageSavedMap-pageFirstReal, Space.defaultWindow];

-- Describe pageSavedMap
IF switches.A = down THEN DescribeSpace[options[TRUE][space][resident], pageSavedMap, 1,
  Space.WindowOrigin[File.Capability[File.nullID, File.read], 0]];

-- Describe buffer, resident, and empty spaces (latter provide room for simple spaces)
DescribeSpace[specialOptions, pageBuffer, countBuffer, Space.defaultWindow];
DescribeSpace[options[TRUE][space][resident], pageResidentMemory, countResidentMemory,
  Space.WindowOrigin[File.Capability[File.nullID, File.write], 0]];
countSwap ← 0;
EnumerateStartList[DescribePass1]; -- set countSwap
DescribeSpace[empty, FIRST[PageNumber] + countLoaded, --countVM--pageSavedMap-countLoaded,
Space.defaultWindow];

-- Now that free real memory exists, start virtual memory replacement process
Process.Detach[FORK ReplacementProcess[countThreshold]];

-- Initialize upper Storage
--bootFile ← ???;
bootFile.device ← IF debuggerVolume THEN 1 ELSE --clientVolume-- 0; -- TEMPORARY KLUDGE
[debugClass, debuggerDevice] ← START FileImpl[@bootFile, @lvBootFiles]; -- find file ids from booting info
IF ~switches.A = down THEN
  IF debuggerDevice = StoragePrograms.nullDevice OR lvBootFiles[debugger].da = Boot.nullDiskFileID.da THEN
    switches.M ← down -- disable map logging
  ELSE
    BEGIN
    parameters.locDebugger ← [device: debuggerDevice, vp: disk[lvBootFiles[debugger]]];
    parameters.locDebuggee ← [device: debuggerDevice, vp: disk[lvBootFiles[debuggee]]];
    canSwap ← TRUE -- now can get to debugger
    END;
InitVMMgr[countVM, @mapLogDescriptor];

SetMP[PilotMP.cMap];

IF ~switches.U = down THEN -- (we only want a code file if this is not UtilityPilot)
```

```
    BEGIN
    fcSwap ← KernelFile.GetRootFile[PilotFileTypes.tCodeFile, Volume.SystemID[]];
    countSwap ← (countSwap + 63)/64*64;
    IF fcSwap.fID = File.nullID THEN
        File.MakePermanent[fcSwap ← File.Create[Volume.SystemID[], countSwap, PilotFileTypes.tCodeFile]]
    ELSE
        File.SetSize[fcSwap, countSwap];
    KernelFile.Pin[fcSwap];
    END;
  offsetSwap ← 0;
  EnumerateStartList[DescribePass2];

  -- Now everything allocated by StartPilot has been placed in the map log
  IF switches.One = down THEN Runtime.CallDebugger["Key Stop 1"];

  START RuntimePrograms.SnapshotImpl; -- not initially resident

  IF ~switches.U = down THEN
      Space.Delete[HandleFromPage[PageFromPointer[tableBase]]]; --Space.Delete not currently implemented for UtilityPilot

  END; -- of OPEN StoragePrograms

  SetMP[PilotMP.cDeleteTemps];
  IF ~switches.U = down THEN KernelFile.DeleteTemps[Volume.SystemID[]];

  DriverPrograms.StartIODrivers[];

  SetMP[PilotMP.cCommunication];
  IF switches.C = up THEN START CommunicationPrograms.CommunicationControl;

  -- Ready to start the client
  SetMP[PilotMP.cClient];
  IF ~switches.U = down THEN  -- no instrumentation for UtilityPilot
      BEGIN
      PPData.Initialize[];  -- initialize instrumentation tables
      PPData.TurnOn[];  -- turn on instrumentation (blow in its ear)
      END;
  IF switches.Two = down THEN Runtime.CallDebugger["Key Stop 2"];
  state.instbyte ← state.stkptr ← 0; state.dest ← PilotClient.Run; state.source ← FrameOps.GetReturnFrame[];
  RETURN WITH state
  END;

-- Modify start list (until StartPilot is changed):
--   Set each space entry bootLoaded IFF some subSpace bootLoaded
--   Set entry for start list itself to bootLoaded readOnly resident other
--   Set pageFirstReal = last page boot loaded + 1
DescribePass0: StartListProc =
  BEGIN
  WITH tableBase[index] SELECT FROM
    space =>
      BEGIN
      IF bootLoaded AND class~ = code THEN pageFirstReal ← MAX[pageFirstReal, vmpage + pages];
      IF PageFromPointer[h] IN [vmpage..vmpage + pages) THEN
        BEGIN
        type ← resident; class ← other; readOnly ← bootLoaded ← TRUE; backingStore ← StartList.SelfFileIndex
        END
      END;
    subSpace => IF bootLoaded THEN
      BEGIN
      pageFirstReal ← MAX[pageFirstReal, tableBase[parent].vmpage + base + pages];
      tableBase[parent].bootLoaded ← TRUE
      END;
    ENDCASE
  END;
```

```
-- Describe initially resident spaces to Swapper, and set countSwap
DescribePass1: StartListProc =
  BEGIN
  WITH tableBase[index] SELECT FROM
    space =>
      BEGIN
      IF vmpage + pages>h.lastVMPage + 1 THEN ERROR; --Error[cBadBootFile]
      IF type = swappable OR type = residentDescriptor THEN countSwap ← countSwap + pages;
      StoragePrograms.DescribeSpace[
        options[bootLoaded][IF class = code THEN parentSpace ELSE space][type],
        vmpage, pages,
        Space.WindowOrigin[File.Capability[FIDFromIndex[backingStore], IF readOnly THEN File.read ELSE File.read + File.write],
          backingPage]]
      END;
    subSpace => StoragePrograms.DescribeSpace[
      options[bootLoaded][subSpace][tableBase[parent].type],
      tableBase[parent].vmpage + base, pages,
      Space.defaultWindow];
    ENDCASE
  END;

-- Describe non-initially resident spaces to VMMgr and unpin initially resident swappable ones
DescribePass2: StartListProc =
  BEGIN OPEN Space;
  indexOuter: StartList.SpaceIndex = LOOPHOLE[index]; -- rename to reach inner scope
  sh, shSuper: Handle;
  pageSuper: PageNumber;
  CreateSubspace: StartListProc = -- create a child of this parent space
    BEGIN
    WITH tableBase[index] SELECT FROM
      subSpace => IF parent = indexOuter AND ~bootLoaded THEN [] ← Create[pages, sh, base];
      ENDCASE
    END;
  WITH tableBase[index] SELECT FROM
    space =>
      BEGIN
      IF type = residentDescriptor OR (type = swappable AND bootLoaded) THEN
        sh ← StoragePrograms.HandleFromPage[vmpage];
      IF type = swappable THEN
        BEGIN
        IF bootLoaded THEN BEGIN IF class~ = code THEN SpecialSpace.MakeSwappable[sh] END
        ELSE
          BEGIN [shSuper, pageSuper] ← StoragePrograms.SuperFromPage[vmpage]; sh ← Create[pages, shSuper, vmpage-
pageSuper] END;
        IF class = code THEN EnumerateStartList[CreateSubspace]; -- create children before mapping
        IF ~bootLoaded THEN Map[sh, [[FIDFromIndex[backingStore], IF readOnly THEN File.read ELSE File.read + File.write],
backingPage]]
        END;
      IF (type = swappable OR type = residentDescriptor) AND ~switches.U = down THEN
        BEGIN
        Remap[sh, [fcSwap, offsetSwap]];
        IF readOnly THEN MakeReadOnly[sh];
        offsetSwap ← offsetSwap + pages
        END
      END;
    ENDCASE
  END;

StartListProc: TYPE = PROCEDURE [index: StartList.Index];

EnumerateStartList: PROCEDURE [Proc: StartListProc] =
  BEGIN OPEN StartList;
  index: Index;
  size: CARDINAL;
  FOR index ← StartIndex, index + size UNTIL tableBase[index].option = stop DO
    SELECT tableBase[index].option FROM
      space, subSpace => Proc[index];
      ENDCASE;
    size ← (WITH tableBase[index] SELECT FROM
      space => SIZE[space Entry],
      subSpace => SIZE[subSpace Entry],
```

```
  INLINE BEGIN
  pageSubSpace: PageNumber;
  WITH tableBase[index] SELECT FROM
    subSpace --base, pages, parent-- =>
      BEGIN
      pageSubSpace ← tableBase[parent].vmpage + base;
      IF pageGerm IN [pageSubSpace..pageSubSpace + pages) THEN
        BEGIN
        -- Add or remove real memory so as to have pages + 1 mapped pages starting with
mdsiGerm*maxPagesInMDS + countSkip
        Inline.LongCOPY[
          from: PointerFromPage[pageSubSpace],
          nwords: pages*wordsPerPage,
          to: Boot.LP[highbits: BootSwap.mdsiGerm, lowbits: PointerFromPage[BootSwap.countSkip]]]
        END
      END;
    ENDCASE
  END;

-- Save away an Alto's worth of map entries which can be set up on way to (old) debugger
-- Note: This code uses pageSavedMap = countVM-1, see calls to DescribeSpace above
InitializeSavedMap: PROCEDURE =
  BEGIN OPEN PageMap;
  page: PageNumber ← pageFirstReal-1;
  offset: PageOffset;
  value: Value;
  StealRealMemory[page: pageSavedMap, count: 1];
  parameters.pSavedMap ← LOOPHOLE[LONG[pageSavedMap]*wordsPerPage];
  FOR offset IN [0..maxPagesInMDS) DO
    parameters.pSavedMap[offset] ←
      IF Inline.BITAND[value←SetF[FIRST[PageNumber] + offset, valueClean], valueVacant] ~ = valueVacant THEN value
      ELSE SetF[page←page + 1, valueClean]
    ENDLOOP
  END;

PageFromPointer: PROCEDURE [pointer: POINTER] RETURNS [PageNumber] =
  BEGIN
  RETURN[LOOPHOLE[pointer, CARDINAL]/wordsPerPage]
  END;

PointerFromPage: PROCEDURE [page: PageNumber] RETURNS [POINTER] =
  BEGIN
  RETURN[LOOPHOLE[page*wordsPerPage]]
  END;

-- When all else fails . . .
Punt: PROCEDURE [c: PilotMP.Code] = INLINE BEGIN SetMP[c]; DO ENDLOOP END;

-- Set Diagnostic Notification Panel code
SetMP: PROCEDURE [PilotMP.Code] = MACHINE CODE BEGIN Mopcodes.zMISC, 10B END;

-- Steal real memory (after that loaded by germ) (can only be called early in ProcessStartList)
StealRealMemory: PROCEDURE [page: PageNumber, count: PageCount] =
  BEGIN OPEN PageMap; -- Assoc, SetF, valueClean, valueVacant
  pageFirstUnmapped: PageNumber;
  offset: PageOffset;
  -- Find last real memory
  FOR pageFirstUnmapped ← pageFirstReal, pageFirstUnmapped + 1 DO
    IF Inline.BITAND[SetF[pageFirstUnmapped, valueClean], valueVacant] = valueVacant THEN EXIT
    ENDLOOP;
  -- Steal last count pages
  IF pageFirstUnmapped-pageFirstReal<count THEN RuntimeInternal.WorryCallDebugger["Real Memory Full"L];
  FOR offset DECREASING IN [0..count) DO
    Assoc[page + offset, SetF[pageFirstUnmapped ← pageFirstUnmapped-1, valueVacant]]
    ENDLOOP
  END;

-- ResidentMemory implementation

map: ARRAY [0..16) OF CARDINAL;
PageState: TYPE = {free, busy};
GetBit: PROCEDURE [POINTER, CARDINAL] RETURNS [PageState] =
```

```
  MACHINE CODE BEGIN Mopcodes.zRFS END;
SetBit: PROCEDURE [PageState, POINTER, CARDINAL] =
  MACHINE CODE BEGIN Mopcodes.zWFS END;
BD: PROCEDURE [CARDINAL] RETURNS [CARDINAL] =
  MACHINE CODE BEGIN Mopcodes.zLI4; Mopcodes.zSHIFT END;

Allocate: PUBLIC PROCEDURE [where: ResidentMemory.Location, pages: CARDINAL]
  RETURNS [LONG POINTER] =
  BEGIN
  RETURN[AllocateMDSPages[pages]] -- ignores 'where', putting everything in mds
  END;

AllocateMDSPages: PUBLIC PROCEDURE [pages: CARDINAL] RETURNS [POINTER] =
  BEGIN
  start, page: CARDINAL;
  FOR start ← 0, page + 1 UNTIL start + pages>countResidentMemory DO
    ProcessInternal.DisableInterrupts[];
    FOR page IN [start..start + pages) DO
      IF GetBit[@map, BD[page]] = busy THEN EXIT --value of page must survive loop exit
      REPEAT FINISHED =>
        BEGIN
        FOR page IN [start..start + pages) DO
          SetBit[busy, @map, BD[page]]
          ENDLOOP;
        ProcessInternal.EnableInterrupts[];
        RETURN[PointerFromPage[pageResidentMemory + start]]
        END
      ENDLOOP;
    ProcessInternal.EnableInterrupts[]
    ENDLOOP;
  DO RuntimeInternal.WorryCallDebugger["Resident Memory Exhausted"L] ENDLOOP
  END;

FreeMDSPages: PUBLIC PROCEDURE [base: POINTER, pages: CARDINAL] =
  BEGIN
  start: CARDINAL = PageFromPointer[base]-pageResidentMemory;
  page: CARDINAL;
  FOR page IN [start..start + pages) DO
    SetBit[free, @map, BD[page]]
    ENDLOOP
  END;
```

Time: April 17, 1979 3:22 PM	*By: McJones*	*Action: Add microcode swapping; fix Launch to observe 64K boundaries*
Time: May 1, 1979 11:35 AM	*By: Lauer*	*Action: Start MiscPrograms.**
Time: July 12, 1979 1:57 PM	*By: McJones*	*Action: Germ, new StartList*
Time: August 1, 1979 11:49 AM	*By: McJones*	*Action: Add boot file id's to FileImpl*
Time: August 6, 1979 12:29 PM	*By: McJones*	*Action: DebuggerSwap*
Time: August 16, 1979 9:35 AM	*By: McJones*	*Action: SpecialFile = >KernelFile; VMMode = >PilotSwitches; make T Key unconditional; InitializeSavedMap used countLoaded instead of pageFirstReal*
Time: August 27, 1979 5:05 PM	*By: McJones*	*Action: Move saved map entries into memory, remapp more*
Time: August 30, 1979 1:13 PM	*By: Gobbel*	*Action: Faces*
Time: August 30, 1979 6:35 PM	*By: McJones*	*Action: Make swap file big enough for additional remapping*
Time: August 30, 1979 6:35 PM	*By: Knutsen*	*Action: Change "START VMMControl[...]" to "InitVMMgr[...]".*
Time: September 4, 1979 9:52 AM	*By: Forrest*	*Action: Change to use PilotFileTypes vs FileTypes*
Time: September 6, 1979 3:16 PM	*By: Ladner*	*Action: Instrumentation*
Time: September 11, 1979 5:20 PM	*By: Knutsen*	*Action: Key control of file creation; optionally get keys from Runpilot*
Time: September 18, 1979 5:08 PM	*By: McJones*	*Action: Move some types from BootChannel and StoragePrograms to Boot*
Time: October 5, 1979 9:26 AM	*By: McJones*	*Action: Set countBuffer to size of real memory*
Time: October 10, 1979 11:34 PM	*By: McJones*	*Action: Start PerformancePrograms.*, SnapshotImpl*