Pilot Definitions

Date: October 1980
Version: 5.0

XEROX
Office Products Division
System Development Department
Palo Alto, California

```
-- file LongBases.Mesa
-- last modified by Satterthwaite, July 30, 1980  10:32 AM
-- Copyright  Xerox Corporation 1980

Bases: DEFINITIONS =
  BEGIN

  Base: TYPE = LONG ORDERED BASE POINTER;
  Finger: TYPE = POINTER TO Base;
  Limit: CARDINAL = 40000B;
  Index: TYPE = Base RELATIVE POINTER [0..Limit);
  OrderedIndex: TYPE = Base RELATIVE ORDERED POINTER [0..Limit);

  IPointer: TYPE = LONG POINTER;

  BaseDescriptor: TYPE = LONG DESCRIPTOR FOR ARRAY OF Base;
  SizeDescriptor: TYPE = LONG DESCRIPTOR FOR ARRAY OF CARDINAL;

  END.
```

```
-- BcdDefs.Mesa  Edited by Satterthwaite on August 26, 1980  9:15 AM
-- Copyright  Xerox Corporation 1979, 1980

DIRECTORY
  PrincOps: TYPE USING [EPIndex, GFTIndex, GFTNull, MaxFrameSize, MaxNGfi],
  Table: TYPE USING [Base, Limit, Selector],
  TimeStamp: TYPE USING [Null, Stamp];

BcdDefs: DEFINITIONS =
  BEGIN

  Base: TYPE = Table.Base;

  -- allocation codes for the binder

  BinderNTables: CARDINAL = 17;

  treetype: Table.Selector = 0; -- trees
  httype: Table.Selector = 1; -- hash table
  sstype: Table.Selector = 2; -- (packed) string table
  cttype: Table.Selector = 3; -- config table
  mttype: Table.Selector = 4; -- module table
  imptype: Table.Selector = 5; -- import table
  exptype: Table.Selector = 6; -- export table
  sgtype: Table.Selector = 7; -- segment table
  fttype: Table.Selector = 8; -- file table
  sttype: Table.Selector = 9; -- semantic table
  cxtype: Table.Selector = 10; -- context table
  nttype: Table.Selector = 11; -- name table
  evtype: Table.Selector = 12; -- external variable table
  sptype: Table.Selector = 13; -- space table
  fptype: Table.Selector = 14; -- frame pack table
  typtype: Table.Selector = 15; -- type table
  tmtype: Table.Selector = 16; -- type table

  -- version identification

  VersionStamp: TYPE = TimeStamp.Stamp;
  NullVersion: TimeStamp.Stamp = TimeStamp.Null;

  -- BCD Header

  VersionID: CARDINAL = 01280;

  BCD: TYPE = RECORD [
    versionIdent: CARDINAL,
    version: VersionStamp,
    creator: VersionStamp,
    sourceVersion: VersionStamp,
    source: NameRecord,
    spare1, spare2: BOOLEAN,
    nPages: [0..377B],
    nConfigs, nModules: CARDINAL,
    nImports, nExports: CARDINAL,
    definitions, repackaged, typeExported, tableCompiled: BOOLEAN,
    versions, extended: BOOLEAN,
    firstdummy: GFTIndex,
    nDummies: CARDINAL,
    ssOffset: CARDINAL, -- string table
    ssLimit: CARDINAL,
    ctOffset: CARDINAL, -- config table
    ctLimit: CTIndex,
    mtOffset: CARDINAL, -- module table
    mtLimit: MTIndex,
    impOffset: CARDINAL, -- import table
    impLimit: IMPIndex,
    expOffset: CARDINAL, -- export table
    expLimit: EXPIndex,
    evOffset: CARDINAL, -- external variable table
    evLimit: EVIndex,
    sgOffset: CARDINAL, -- segment table
    sgLimit: SGIndex,
    ftOffset: CARDINAL, -- file table
    ftLimit: FTIndex,
    spOffset: CARDINAL, -- space table
    spLimit: SPIndex,
    ntOffset: CARDINAL, -- name table
    ntLimit: NTIndex,
```

```
    typOffset: CARDINAL, -- type table
    typLimit: TYPIndex,
    tmOffset: CARDINAL, -- type map table
    tmLimit: TMIndex,
    fpOffset: CARDINAL, -- frame pack table
    fpLimit: FPIndex];

  -- Portable Type

  Portable: TYPE = {module, interface};

  -- Name Table

  PackedString: TYPE = MACHINE DEPENDENT RECORD [
    SELECT OVERLAID * FROM
      string => [string: StringBody],
      size => [size: PACKED ARRAY [-3..-3) OF [0..377B]]
      ENDCASE];

  NameRecord: TYPE = RECORD [CARDINAL];
  NullName: NameRecord = [1];

  NTRecord: TYPE = RECORD [name: NameRecord, item: Namee];

  Namee: TYPE = RECORD [
    SELECT type: * FROM
      config => [cti: CTIndex],
      module => [mti: MTIndex],
      import => [impi: IMPIndex],
      export => [expi: EXPIndex]
      ENDCASE];

  NTIndex: TYPE = Table.Base RELATIVE POINTER [0..Table.Limit) TO NTRecord;
  NTNull: NTIndex = LAST[NTIndex];

  -- Configuration Table

  CTRecord: TYPE = RECORD [
    name: NameRecord,
    namedInstance: BOOLEAN,
    file: FTIndex,
    config: CTIndex,
    nControls: CARDINAL,
    controls: ARRAY [0..0) OF MTIndex];

  CTIndex: TYPE = Table.Base RELATIVE POINTER [0..Table.Limit) TO CTRecord;
  CTNull: CTIndex = LAST[CTIndex];

  -- Module Table

  LinkLocation: TYPE = {frame, code};

  MTRecord: TYPE = RECORD [
    name: NameRecord,
    namedInstance: BOOLEAN,
    initial: BOOLEAN,
    file: FTIndex,
    links: LinkLocation,
    config: CTIndex,
    code: CodeDesc,
    sseg: SGIndex,
    long, tableCompiled, boundsChecks, nilChecks: BOOLEAN,
    acMap: SGIndex,
    framesize: [0..PrincOps.MaxFrameSize),
    altoCode, residentFrame, crossJumped, packageable: BOOLEAN,
    gfi: GFTIndex,
    variables: EVIndex,
    ngfi: [1..PrincOps.MaxNGfi],
    frame: FrameFrag];

  CodeDesc: TYPE = RECORD [
    sgi: SGIndex, packed: BOOLEAN, linkspace: BOOLEAN, offset, length: CARDINAL];

  FrameFrag: TYPE = RECORD [length: CARDINAL, frag: ARRAY [0..0) OF Link];

  MTIndex: TYPE = Table.Base RELATIVE POINTER [0..Table.Limit) TO MTRecord;
  MTNull: MTIndex = LAST[MTIndex];
```

```
  -- Import Table

  IMPRecord: TYPE = RECORD [
    name: NameRecord,
    port: Portable,
    namedInstance: BOOLEAN,
    file: FTIndex,
    gfi: GFTIndex,
    ngfi: [1..PrincOps.MaxNGfi]];

  IMPIndex: TYPE = Table.Base RELATIVE POINTER [0..Table.Limit) TO IMPRecord;
  IMPNull: IMPIndex = LAST[IMPIndex];

  -- Export Table

  EXPRecord: TYPE = RECORD [
    name: NameRecord,
    size: [0..377B],
    port: Portable,
    namedInstance, typeExported: BOOLEAN,
    file: FTIndex,
    links: ARRAY [0..0) OF Link];

  EXPIndex: TYPE = Table.Base RELATIVE POINTER [0..Table.Limit) TO EXPRecord;
  EXPNull: EXPIndex = LAST[EXPIndex];

  -- External Variable Table

  EVRecord: TYPE = RECORD [length: CARDINAL, offsets: ARRAY [1..1) OF CARDINAL];

  EVIndex: TYPE = Table.Base RELATIVE POINTER [0..Table.Limit) TO EVRecord;
  EVNull: EVIndex = LAST[EVIndex];

  -- Segment Table

  SegClass: TYPE = {code, symbols, acMap, other};

  SGRecord: TYPE = RECORD [
    class: SegClass, file: FTIndex, base: CARDINAL, pages, extraPages: [0..256)];

  SGIndex: TYPE = Table.Base RELATIVE POINTER [0..Table.Limit) TO SGRecord;
  SGNull: SGIndex = LAST[SGIndex];

  -- File Table

  FTRecord: TYPE = RECORD [name: NameRecord, version: VersionStamp];

  FTIndex: TYPE = Table.Base RELATIVE POINTER [0..Table.Limit) TO FTRecord;
  FTNull: FTIndex = LAST[FTIndex];
  FTSelf: FTIndex = LAST[FTIndex] - 1;

  -- Space Table

  SPRecord: TYPE = RECORD [
    seg: SGIndex, length: CARDINAL, spaces: ARRAY [0..0) OF SpaceID];

  SpaceID: TYPE = RECORD [
    name: NameRecord, resident: BOOLEAN, offset: [0..256), pages: [1..128]];

  SPIndex: TYPE = Table.Base RELATIVE POINTER [0..Table.Limit) TO SPRecord;
  SPNull: SPIndex = LAST[SPIndex];

  -- Frame Pack Table

  FPRecord: TYPE = RECORD [
    name: NameRecord, length: CARDINAL, modules: ARRAY [0..0) OF MTIndex];

  FPIndex: TYPE = Table.Base RELATIVE POINTER [0..Table.Limit) TO FPRecord;
  FPNull: FPIndex = LAST[FPIndex];

  -- Type Table

  TYPRecord: TYPE = RECORD [version: VersionStamp, id: RECORD [UNSPECIFIED]];

  TYPIndex: TYPE = Table.Base RELATIVE POINTER [0..Table.Limit) TO TYPRecord;
  TYPNull: TYPIndex = LAST[TYPIndex];

  -- Type Map Table
```

```
  TMRecord: TYPE = RECORD [
    version: VersionStamp, offset: CARDINAL, map: TYPIndex];

  TMIndex: TYPE = Table.Base RELATIVE POINTER [0..Table.Limit) TO TMRecord;
  TMNull: TMIndex = LAST[TMIndex];

  -- Links

  GFTIndex: TYPE = PrincOps.GFTIndex;
  GFTNull: GFTIndex = PrincOps.GFTNull;

  EPIndex: TYPE = PrincOps.EPIndex;
  EPLimit: CARDINAL = LAST[EPIndex] + 1;

  VarIndex: TYPE = [0..17B];
  VarLimit: CARDINAL = LAST[VarIndex] + 1;

  NullLink: Link = [procedure[0, 0, FALSE]];
  UnboundLink: Link = [procedure[0, 0, TRUE]];

  LinkTag: TYPE = {variable, procedure, type};
  VarTag: TYPE = {var, proc0, type, proc1};

  Link: TYPE = MACHINE DEPENDENT RECORD [
    SELECT OVERLAID LinkTag FROM
      variable => [vgfi: GFTIndex, var: VarIndex, vtag: VarTag],
      procedure => [gfi: GFTIndex, ep: EPIndex, tag: BOOLEAN],
      type => [typeID: TYPIndex, type: BOOLEAN, proc: BOOLEAN]
      ENDCASE];

  END.
```

```
-- LongBcdOps.Mesa  Edited by Sandman on August 12, 1980  4:14 PM
-- Copyright  Xerox Corporation 1979, 1980

DIRECTORY
  BcdDefs USING [
    BCD, CTIndex, CTRecord, EVIndex, EVRecord, EXPIndex, EXPRecord, FPRecord,
    FTIndex, FTRecord, IMPIndex, IMPRecord, MTIndex, MTRecord, Namee, NameRecord,
    NTIndex, NTRecord, PackedString, SGIndex, SGRecord, SPIndex, SPRecord,
    TMRecord, TYPRecord, VersionStamp];

BcdOps: DEFINITIONS =
  BEGIN OPEN BcdDefs;

  BcdBase: TYPE = LONG POINTER TO BcdDefs.BCD;

  NTHandle: TYPE = LONG POINTER TO BcdDefs.NTRecord;
  CTHandle: TYPE = LONG POINTER TO BcdDefs.CTRecord;
  MTHandle: TYPE = LONG POINTER TO BcdDefs.MTRecord;
  IMPHandle: TYPE = LONG POINTER TO BcdDefs.IMPRecord;
  EXPHandle: TYPE = LONG POINTER TO BcdDefs.EXPRecord;
  EVHandle: TYPE = LONG POINTER TO BcdDefs.EVRecord;
  SGHandle: TYPE = LONG POINTER TO BcdDefs.SGRecord;
  FTHandle: TYPE = LONG POINTER TO BcdDefs.FTRecord;
  SPHandle: TYPE = LONG POINTER TO BcdDefs.SPRecord;
  FPHandle: TYPE = LONG POINTER TO BcdDefs.FPRecord;
  TYPHandle: TYPE = LONG POINTER TO BcdDefs.TYPRecord;
  TMHandle: TYPE = LONG POINTER TO BcdDefs.TMRecord;

  NameString: TYPE = LONG POINTER TO PackedString;

  ProcessConfigs: PROCEDURE [
    bcd: BcdBase, proc: PROCEDURE [CTHandle, CTIndex] RETURNS [BOOLEAN]]
    RETURNS [cth: CTHandle, cti: CTIndex];

  ProcessExternals: PROCEDURE [
    bcd: BcdBase, proc: PROCEDURE [EVHandle, EVIndex] RETURNS [BOOLEAN]]
    RETURNS [evh: EVHandle, evi: EVIndex];

  ProcessExports: PROCEDURE [
    bcd: BcdBase, proc: PROCEDURE [EXPHandle, EXPIndex] RETURNS [BOOLEAN]]
    RETURNS [eth: EXPHandle, eti: EXPIndex];

  ProcessFiles: PROCEDURE [
    bcd: BcdBase, proc: PROCEDURE [FTHandle, FTIndex] RETURNS [BOOLEAN]]
    RETURNS [fth: FTHandle, fti: FTIndex];

  ProcessImports: PROCEDURE [
    bcd: BcdBase, proc: PROCEDURE [IMPHandle, IMPIndex] RETURNS [BOOLEAN]]
    RETURNS [ith: IMPHandle, iti: IMPIndex];

  ProcessModules: PROCEDURE [
    bcd: BcdBase, proc: PROCEDURE [MTHandle, MTIndex] RETURNS [BOOLEAN]]
    RETURNS [mth: MTHandle, mti: MTIndex];

  ProcessNames: PROCEDURE [
    bcd: BcdBase, proc: PROCEDURE [NTHandle, NTIndex] RETURNS [BOOLEAN]]
    RETURNS [nth: NTHandle, nti: NTIndex];

  ProcessSegs: PROCEDURE [
    bcd: BcdBase, proc: PROCEDURE [SGHandle, SGIndex] RETURNS [BOOLEAN]]
    RETURNS [sgh: SGHandle, sgi: SGIndex];

  ProcessSpaces: PROCEDURE [
    bcd: BcdBase, proc: PROCEDURE [SPHandle, SPIndex] RETURNS [BOOLEAN]]
    RETURNS [sph: SPHandle, spi: SPIndex];

  FindName: PROCEDURE [bcd: BcdBase, owner: Namee] RETURNS [name: NameRecord];

  ModuleVersion: PROCEDURE [bcd: BcdBase, mti: MTIndex]
    RETURNS [version: VersionStamp];

  END.
```

```
DIRECTORY
    Environment: FROM "Environment" USING [BitAddress],
    Mopcodes: FROM "Mopcodes" USING [zBITBLT];

BitBlt: DEFINITIONS =
BEGIN

-- Types used to make up a BitBltTable

BitAddress: TYPE = Environment.BitAddress;

BitBltFlags: TYPE = MACHINE DEPENDENT RECORD[ -- determines the BitBlt function
    direction: Direction ← forward,
    disjoint: BOOLEAN ← FALSE,
    disjointItems: BOOLEAN ← FALSE,
    gray: BOOLEAN ← FALSE,
    srcFunc: SrcFunc ← null,
    dstFunc: DstFunc ← null,
    reserved: [0..511] ← 0];

Direction: TYPE = {forward, backward};

DstFunc: TYPE = {null, and, or, xor};

GrayParm: TYPE = MACHINE DEPENDENT RECORD[ -- used with Src to describe gray brick
    reserved: [0..15] ← 0,
    yOffset: [0..15],
    widthMinusOne: [0..15], -- restricted to 0 for initial microcode implementations
    heightMinusOne: [0..15]];

SrcDesc: TYPE = MACHINE DEPENDENT RECORD[SELECT OVERLAID * FROM
    gray => [gray: GrayParm],
    srcBpl => [srcBpl: INTEGER],
    ENDCASE];

SrcFunc: TYPE = {null, complement};

-- BitBlt and BBTable

BBptr, BitBltTablePtr: TYPE = POINTER TO BBTable;

BBTable, BitBltTable: TYPE = MACHINE DEPENDENT RECORD [
    dst: BitAddress,
    dstBpl: INTEGER,
    src: BitAddress,
    srcDesc: SrcDesc,
    width: CARDINAL,
    height: CARDINAL,
    flags: BitBltFlags,
    reserved: UNSPECIFIED ← 0];

BITBLT: PROCEDURE [ptr: BBptr] = MACHINE CODE BEGIN Mopcodes.zBITBLT END;

-- alignment issues

BBTableAlignment: CARDINAL = 16;
BBTableSpace: TYPE = ARRAY [1..SIZE[BBTable] + BBTableAlignment) OF UNSPECIFIED;
AlignedBBTable: PROCEDURE [ip: POINTER TO BBTableSpace] RETURNS [b: BBptr] =
    INLINE BEGIN
    align: TYPE = MACHINE DEPENDENT RECORD[
        s: [0..LAST[WORD]/BBTableAlignment),
        z: [0..BBTableAlignment)];
    b ← LOOPHOLE[ip + BBTableAlignment - 1];
    LOOPHOLE[b, align].z ← 0;
    END;

END.
```

LOG
Time: April 11, 1980 11:09 AM By: Forrest Action: Trimmed log to Amargosa; converted to PrincOps BitBlt.

Time: April 16, 1980 1:26 PM By: Forrest Action: formatting/spelling corections. Changed BBTableSpace to [1.. from [0..
Time: April 17, 1980 12:07 AM By: Forrest Action: Move BitAddress to Environment
Time: July 17, 1980 4:03 PM By: Karlton Action: Add defaults to Flags record
Time: timeStamp By: yourName Action: shortDescription

```
DIRECTORY
    Device USING [Type],
    File USING [ID, nullID, PageNumber],
    Mopcodes USING [zEXCH, zLI1, zLP, zPOP],
    Volume USING [Type];

Boot: DEFINITIONS =

BEGIN

    -- Basic request to germ
Request: TYPE = MACHINE DEPENDENT RECORD [
    action: Action, location: Location];
pRequest: POINTER TO Request = LOOPHOLE[1360B]; -- relative to germ's MDS

-- NOTE: pRequest + SIZE[Request] must not be greater than 1400B, the start of the GFT.

Action: TYPE = RECORD [CARDINAL];
    inLoad: Action = [0]; -- restore volatile processor state from BootFile-format snapshot
    outLoad: Action = [1]; -- save volatile processor state in BootFile-format snapshot
    bootPhysicalVolume: Action = [2]; -- do inLoad using Location specified indirectly in pilot entry of PVBootFiles array of physical volume root
    page of disk specified by accompanying Location
    teledebug: Action = [3];
    noOp: Action = [4]; -- simply Enter and Exit germ

pInitialLink: POINTER--TO ControlDefs.ControlLink-- = LOOPHOLE[1376B]; -- itself an indirect control link within germ MDS

    -- Description of boot file location

Location: TYPE = MACHINE DEPENDENT RECORD [
    deviceType: Device.Type, -- e.g. diablo31, sa4000, ethernet
    deviceOrdinal: CARDINAL, -- position of device within all those of same type
    vp: SELECT OVERLAID * FROM
        disk => [diskFileID: DiskFileID],
        ethernet => [bootFileNumber: CARDINAL, net, host: CARDINAL ← 0],
        any => [a, b, c, d, e, f, g, h: UNSPECIFIED],
        ENDCASE];

-- NOTE: The notion of a device ordinal is intended to be a function purely of a hardware configuration, and not of the microcode or
heads running on that configuration. This eases the problem of microcode swapping.

DiskFileID: TYPE = MACHINE DEPENDENT RECORD [
    fID: File.ID,
    firstPage: File.PageNumber,
    da: DiskAddress];

DiskAddress: TYPE = PRIVATE RECORD [UNSPECIFIED, UNSPECIFIED];

-- Convention: a DiskFileID is null if its fID is null

nullDiskFileID: DiskFileID = [File.nullID, 0, [0, 0]]; -- delete this

    -- Description of boot files pointed to from root pages of physical and logical volumes
    -- The following cannot be changed without invalidating all Pilot volumes
BootFileType: TYPE = {hardMicrocode, softMicrocode, germ, pilot, debugger, debuggee};

PVBootFiles: TYPE = ARRAY BootFileType[hardMicrocode..pilot] OF DiskFileID;

LVBootFiles: TYPE = ARRAY BootFileType OF DiskFileID;

VolumeType: TYPE = Volume.Type;

    -- Miscellaneous useful definitions

LP: PROCEDURE [lowbits, highbits: UNSPECIFIED] RETURNS [LONG POINTER] =
    MACHINE CODE BEGIN END;

MDSIndex: TYPE = [0..256);

ReadMDS: PROCEDURE RETURNS [MDSIndex] = -- HighHalf[LONG[LOOPHOLE[1, POINTER]]] triggers compiler bug
    MACHINE CODE BEGIN Mopcodes.zLI1; Mopcodes.zLP; Mopcodes.zEXCH; Mopcodes.zPOP END;

END.
```

LOG

Time: September 13, 1979 6:03 PM By: McJones Action: Create file
Time: January 25, 1980 4:22 PM By: McJones Action: Add bootPhysicalVolume and noOp Request's and ethernet Location
Time: January 25, 1980 6:37 PM By: McJones Action: Replace Location.device with Location.deviceType and .deviceOrdinal
Time: April 17, 1980 12:41 AM By: Forrest Action: Added teledebug
Time: April 17, 1980 10:39 AM By: Luniewski Action: Added net and host to Location.ethernet
Time: July 15, 1980 10:09 PM By: Forrest Action: Add NoOp; refer to Volume.Type

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

BootChannel: DEFINITIONS =

BEGIN OPEN Environment;

Operation: TYPE = {read,   -- read
                   write,  -- write
                   rawRead};          -- raw read: read (single) boot page with no label verity

Create: PROCEDURE [pLocation: POINTER TO Boot.Location, operation: Operation,
                   dFirst64KStorage: LONG DESCRIPTOR FOR ARRAY OF WORD]
          RETURNS [handle: Handle];
    -- Create boot channel from specification of device.
    -- dFirst64KStorage should be used for allocating IOCB's, etc. in the first 64K.  It is guaranteed to be 16-word aligned.
    -- (When germ resides in first 64K, dHardwareState and dCSBStorage need not be LONG).

Handle: TYPE = PROCEDURE [page: PageNumber, count: PageCount];

Transfer: PROCEDURE [handle: Handle, page: PageNumber, count: PageCount] =
    INLINE BEGIN handle[page, count] END;
    -- Perform transfer of the next sequential count pages to/from the boot file starting at the specified (virtual memory) page.
    -- When all transfers have been completed, Transfer is called with count = 0.  At this time, the BootChannel should perform any
       necessary cleanup.

END.
```

LOG

Time: December 18, 1978 9:34 AM	*By: McJones*	*Action: Create file*
Time: July 30, 1979 4:32 PM	*By: McJones*	*Action: Added substructure and disk variant to Location*
Time: September 19, 1979 6:39 PM	*By: McJones*	*Action: Move Location, DiskFileID, DiskAddress to Boot*
Time: October 5, 1979 6:09 PM	*By: McJones*	*Action: Move dHardwareState from Open to Create*
Time: October 13, 1979 4:30 PM	*By: McJones*	*Action: Combine TransferPage, Wait into Apply*
Time: November 16, 1979 10:25 AM	*By: McJones*	*Action: Runs: drop Open, Wait; add count to Transfer*
Time: January 17, 1980 2:46 PM	*By: Gobbel*	*Action: SA4000 booting: added rawRead to Operation type*
Time: February 4, 1980 8:51 AM	*By: Knutsen*	*Action: Added F64K allocator, cleanup call of* Transfer.
Time: February 4, 1980 8:51 AM	*By: McJones*	*Action: Drop hardware state*

```
DIRECTORY
 Environment USING [PageCount, PageNumber, wordsPerPage],
 PageMap USING [Value];

BootFile: DEFINITIONS =
BEGIN

currentVersion: CARDINAL = 101; -- for boot files written using these definitions (see Header below)

-- A boot file is intended to capture the state of real memory and the relevant processor registers, most notably the map. It consists of a
Header page followed by one or more data pages, followed by zero or more groups each consisting of a Trailer page followed by one or
more data pages. Each Header or Trailer page assigns virtual and real addresses and flag bits to the accompanying data pages. The
Header page contains some additional global state.

-- Assertion: if Entry[p, v] precedes Entry[p', v'] in a boot file, then p is less than (and not equal to) p'.
-- If a boot file does NOT contain an entry for a given page, that page is understood to be vacant.

-- There are two variations of boot files, corresponding to whether the captured state is a snapshot of a running program or has been
fabricated with "bit tweezers". In the former case (resumptive Continuation), some process is waiting on the
BootSwap.pMon.condResponse condition variable. In the latter case (initial Continuation), a distinguished PSB is made ready to xfer to a
given control link and the BootSwapGerm waits.

-- There is an additional distinction depending on whether the inLoadMode of a boot file is load or resume.
-- An inLoadMode = load signifies that the program captured in the boot file does not care into which real pages it is loaded. In this case
InLoad expects all real memory to be mapped to an initial prefix of virtual memory; it leaves all excess real memory mapped immediately
following the last virtual page it loads. (Thus those virtual pages between ones which are loaded are set vacant.)
-- An inLoadMode = restore signifies that the program captured in the boot file expects to be reloaded into the real pages specified in
the boot file.
-- With either inLoadMode, the flags (W, D, R) of the loaded pages are set to the values specified in the boot file.

-- To be distributed via the current ethernet boot servers, one of these boot files would have to be "encapsulated" by preceding it with a
dummy page containing an appropriate timestamp.

Header: TYPE = MACHINE DEPENDENT RECORD [ -- first page of boot file
 version --(0)--: CARDINAL ← currentVersion,
 creationDate --(1)--: LONG UNSPECIFIED, -- System.GreenwichMeanTime
 pStartListHeader --(3)--: POINTER, -- when continuation kind = initial (and relative to that mds)
 fill: [0..100000B) ← 0,
 inLoadMode --(4:0..15)--: InLoadMode,
  -- description of computation after InLoad
 continuation --(5)--: Continuation,
  -- processor CSB (how much applies only in resumptive case?)
 -- currentPSB, readyList, currentSV,firstPSB,lastPSB,firstSV,wakeupsWaiting,active,timeoutCounter:
  -- page map data
 countData --(7)--: CARDINAL, -- number of nonvacant pages (not counting germ)
 entries --(10B)--: ARRAY [0..0) OF Entry];

Trailer: TYPE = MACHINE DEPENDENT RECORD [
 version --(0)--: CARDINAL ← currentVersion,
 entries --(1)--: ARRAY [0..0) OF Entry];

InLoadMode: TYPE = {load, restore};

Continuation: TYPE = MACHINE DEPENDENT RECORD [
 fill: [0..177B] ← 0,
 vp: SELECT kind --(0:0..7)--: ContinuationKind FROM
  initial => [
   mdsi --(0:8..15)--: MDSIndex,
   destination --(1)--: UNSPECIFIED -- control link --],
  resumptive => [
   mdsi --(0:8..15)--: MDSIndex -- for WriteMDS hack --,
   resumee --(1)--: UNSPECIFIED -- handle of waiting process --],
  ENDCASE];
ContinuationKind: TYPE = {initial, resumptive};

Entry: TYPE = MACHINE DEPENDENT RECORD [
 page (0): Environment.PageNumber,
 value (1): PageMap.Value];

MDSIndex: TYPE = RECORD [[0..256)]; -- high order bits of long point to MDS (low 16 must be zero)

MemorySizeToFileSize: PROCEDURE [countReal: Environment.PageCount] RETURNS [Environment.PageCount] =
```

```
 -- Calculate upper bound on boot file size as function of real memory size.
 INLINE BEGIN RETURN[
  countReal -- total data pages
   + 1 -- header page
   + (MAX[countReal, maxEntriesPerHeader]-maxEntriesPerHeader + maxEntriesPerTrailer-1)
    /maxEntriesPerTrailer -- trailer pages
  ] END;

maxEntriesPerHeader: CARDINAL = (Environment.wordsPerPage-SIZE[Header])/SIZE[Entry];
maxEntriesPerTrailer: CARDINAL = (Environment.wordsPerPage-SIZE[Trailer])/SIZE[Entry];

END.
```

LOG

Time: December 15, 1978 5:32 PM By: McJones Action: Create file
Time: January 26, 1980 11:29 PM By: Forrest Action: Move MDSIndex here from Boot
Time: July 11, 1980 9:09 AM By: McJones Action: Delete self field of Header

-- The location of switches should be defined in SDDefs!!

```
DIRECTORY
 Boot USING [
  Action, inLoad, Location, LP, MDSIndex, outLoad, pInitialLink, pRequest,
  ReadMDS, Request, teledebug],
 BootFile USING [Continuation, ContinuationKind, InLoadMode],
 --DeviceCleanup: FROM "DeviceCleanup" USING [Perform],
 Environment USING [PageCount, wordsPerPage],
 Frame USING [MyLocalFrame],
 Inline USING [LongCOPY],
 Mopcodes USING [zMISC],
 PrincOps USING [BytePC, FrameHandle],
 SDDefs USING [SD --, SFirstFree--],
 TemporaryBooting USING [defaultSwitches, Switches];

BootSwap: DEFINITIONS IMPORTS Boot--, DeviceCleanup--, Frame, Inline =
BEGIN
sPilotSwitches: CARDINAL = 142B; -- SDDefs.SD[sPilotSwitches] is the home for switches, used to be called
SDDefs.sFirstFree.

Initialize: PROC [mdsiOther: Boot.MDSIndex];
 -- Per-system initialization (does InitializeMDS also)
InitializeMDS: PROC;
 -- Per-MDS initialization

GetPStartListHeader: PROC RETURNS [POINTER] =
 -- Valid after start of initial boot file up until first call to OutLoad
 INLINE { RETURN[pMon.pStartListHeader] };

OutLoad: PROC [pLocation: POINTER TO Boot.Location, inLoadMode: BootFile.InLoadMode]
  RETURNS [ResponseKind[outLoaded..resumed]] =
 -- Assume interrupts disabled, all devices turned off
 INLINE BEGIN
 -- Set up request in cross-MDS monitor shared with germ
 --  (until MDS field is added to PSB, simulate process switch with WriteMDS)
 RC: TYPE = MACHINE DEPENDENT RECORD [ -- to get around compiler bug
  fill: [0..177B], kind: BootFile.ContinuationKind, mdsi: Boot.MDSIndex , resumee: UNSPECIFIED];
 -- pMon.continuation ← [fill: 0, vp: resumptive[mdsi: Boot.ReadMDS[], resumee: Frame.MyLocalFrame[]]];
 pMon.continuation ← LOOPHOLE[RC[
  fill: 0, kind: resumptive, mdsi: Boot.ReadMDS[], resumee: Frame.MyLocalFrame[]]];
 pMon.inLoadMode ← inLoadMode;
 SetGermArguments[Boot.outLoad, pLocation];
 -- pMon.state ← request;
 -- NOTIFY pMon.condRequest;
 -- WHILE pMon.state~ = response DO WAIT pMon.condResponse ENDLOOP;
 CallGerm[];
 RETURN[pMon.responseKind]
 END;

ResponseKind: TYPE = {initiated, outLoaded, resumed};

InLoad: PROC [
  pMicrocodeCopy,
  pGermCopy: LONG POINTER, nGerm: CARDINAL,
  pLocation: POINTER TO Boot.Location,
  switches: TemporaryBooting.Switches ← TemporaryBooting.defaultSwitches] =
 -- Load boot file, performing microcode and germ swap if necessary
 INLINE BEGIN
 -- IF pMicrocodeCopy~ = NIL THEN DeviceCleanup.Perform[kill]; + + causes FATAL ERROR IN PASS 4
 IF pGermCopy~ = NIL THEN
  Inline.LongCOPY[from: pGermCopy, nwords: nGerm, to: Boot.LP[highbits: mdsiGerm, lowbits: countSkip*Environment.wordsPerPage]];
 SetGermArguments[Boot.inLoad, pLocation];
 LOOPHOLE[Boot.LP[highbits: mdsiGerm, lowbits: @SDDefs.SD[sPilotSwitches]], LONG POINTER TO TemporaryBooting.Switches]↑ ←
  switches;
 IF pMicrocodeCopy~ = NIL THEN LoadRam[pMicrocode: pMicrocodeCopy, andJump: TRUE]; -- never returns
 CallGerm[]
 END;

-- The germ is free to ingore pLocation
Teledebug: PROC [pLocation: POINTER TO Boot.Location] =
```

```
 -- Assume interrupts disabled, all devices turned off
 INLINE BEGIN
 -- Set up request in cross-MDS monitor shared with germ
 --  (until MDS field is added to PSB, simulate process switch with WriteMDS)
 SetGermArguments[Boot.teledebug, pLocation];
 CallGerm[];
 END;

SetGermArguments: PRIVATE PROC [action: Boot.Action, pLocation: POINTER TO Boot.Location] =
 INLINE BEGIN
 pRequest: LONG POINTER TO Boot.Request = LOOPHOLE[Boot.LP[highbits: mdsiGerm, lowbits: Boot.pRequest]];
 pRequest↑ ← [action, pLocation↑];
 END;

CallGerm: PRIVATE PROC =
 INLINE BEGIN
 pMon.CrossMDSCall[mdsiOther: mdsiGerm, Dest: LOOPHOLE[Boot.pInitialLink]];
 LOOPHOLE[pMon.CrossMDSCall, POINTER TO PrincOps.FrameHandle].pc ← LOOPHOLE[pMon.pcCross];
 END;

LoadRam: PRIVATE PROC [pMicrocode: LONG POINTER, andJump: BOOLEAN] = MACHINE CODE BEGIN Mopcodes.zMISC, 3 END;

 -- Extension to basic request for cross-MDS communication
 -- WHEN THIS IS CHANGED, both Pilot and Germ will be incompatible untill both are updated!! (at the very least add
things to the front)
Mon: PRIVATE TYPE = --MONITORED-- RECORD [
   -- constants for WriteMDS hack:
 CrossMDSCall: PROC [mdsiOther: Boot.MDSIndex, Dest: PROC],
 pcCross: PrincOps.BytePC,
   -- additional fields for outLoad request:
 continuation: resumptive BootFile.Continuation, -- for WriteMDS hack
 inLoadMode: BootFile.InLoadMode,
   -- response fields:
 responseKind: ResponseKind,
 fill: [0..8192) ← 0,
 pStartListHeader: POINTER, -- only if responseKind = booted
 selfFill: RECORD [a, b, c, d: UNSPECIFIED] -- obsolete, remove someday
 -- state: {request, response},
 -- condRequest: CONDITION,
 -- condResponse: CONDITION, + + only if responseKind~ = booted
 ];

pMon: PRIVATE LONG POINTER TO Mon;

 -- Attributes of germ
mdsiGerm: Boot.MDSIndex = 76B; -- until Pilot is moved from first 64K
countSkip: Environment.PageCount = 2; -- initial vacant pages within germ; will become 0
pCountGerm: PRIVATE POINTER TO Environment.PageCount = LOOPHOLE[1377B]; -- excluding pageBuffer

END.
```

LOG

Time: April 17, 1980 12:39 AM By: Forrest Action: Move teledebug to Boot
Time: April 24, 1980 5:07 PM By: Forrest Action: FrameOps = > Frame, ControlDefs = > PrincOps
Time: July 19, 1980 7:16 PM By: Forrest Action: Started programming around SDDefs changing. Took self out of PMon

```
-- File: BufferDefs.Mesa,  Last Edit: Garlick  October 12, 1980  4:49 PM

DIRECTORY
  OISCPTypes USING [
    maxDataWordsPerSpp,
    OisAddress, OisConnectionID, OisHostID, OisNetID, OisPacketType, OisSocketID],
  PupTypes USING [
    maxDataWordsPerGatewayPup,
    Pair, PupAddress, PupErrorCode, PupHostID, PupNetID, PupSocketID, PupType,
    RppAddress],
  DriverTypes USING [DeviceType, Encapsulation, Seal];


BufferDefs: DEFINITIONS = BEGIN

-- Basic Data TYPEs
Byte: TYPE = [0..377B];

OisAddress: TYPE = OISCPTypes.OisAddress;
OisHostID: TYPE = OISCPTypes.OisHostID;
OisNetID: TYPE = OISCPTypes.OisNetID;
OisSocketID: TYPE = OISCPTypes.OisSocketID;
OisConnectionID: TYPE = OISCPTypes.OisConnectionID;

PupAddress: TYPE = PupTypes.PupAddress;
PupHostID: TYPE = PupTypes.PupHostID;
PupNetID: TYPE = PupTypes.PupNetID;
PupSocketID: TYPE = PupTypes.PupSocketID;

RppAddress: TYPE = PupTypes.RppAddress;

-- QUEUE
Queue: TYPE = LONG POINTER TO QueueObject;
QueueObject: TYPE = PRIVATE RECORD [
  length: PUBLIC CARDINAL,
  first, last: Buffer,
  seal: DriverTypes.Seal ];

-- BufferPool
BufferPool: TYPE = LONG POINTER TO BufferPoolObject;
BufferPoolObject: TYPE = PRIVATE MONITORED RECORD [
  freeQueue: QueueObject,
  seal: DriverTypes.Seal,
  freeQueueNotEmpty: CONDITION,
  sendBufferAvailable: CONDITION,
  receiveBufferAvailable: CONDITION,
  total, reserve, send, receive: CARDINAL,
  sendInUse, receiveInUse: CARDINAL,
  dataWordsPerBuffer: CARDINAL,
  firstBuffer: Buffer,
  bufferSize: BufferSize,
  wordsPerBuffer: CARDINAL ];


Enqueue: PROCEDURE [Queue, Buffer];
Dequeue: PROCEDURE [Queue] RETURNS [Buffer];
ExtractFromQueue: PUBLIC PROCEDURE [Queue, Buffer] RETURNS [Buffer];
QueueLength: PROCEDURE [q: Queue] RETURNS [CARDINAL] = INLINE
  BEGIN RETURN[q.length]; END;
QueueEmpty: PROCEDURE [q: Queue] RETURNS [BOOLEAN] = INLINE
  BEGIN RETURN[q.length=0]; END;
QueueInitialize: PROCEDURE [Queue];
QueueCleanup: PROCEDURE [Queue];


PupBufferObject: TYPE = pup BufferObject;
PupBuffer: TYPE = LONG POINTER TO PupBufferObject;
RppBufferObject: TYPE = rpp BufferObject;
RppBuffer: TYPE = LONG POINTER TO RppBufferObject;
OisBufferObject: TYPE = ois BufferObject;
OisBuffer: TYPE = LONG POINTER TO OisBufferObject;
SppBuffer: TYPE = LONG POINTER TO spp OisBufferObject;


-- From specified BufferPool
BufferPoolMake: PROCEDURE [BufferPool];
BufferPoolDestroy: PROCEDURE [BufferPool];
GetBufferFromPool: PROCEDURE [BufferPool] RETURNS [Buffer];
```

```
GetFreeSendBufferFromPool: PROCEDURE [BufferPool] RETURNS [Buffer];
GetFreeReceiveBufferFromPool: PROCEDURE [BufferPool] RETURNS [Buffer];
MaybeGetBufferFromPool: PROCEDURE [BufferPool] RETURNS [Buffer];
MaybeGetFreeSendBufferFromPool: PROCEDURE [BufferPool] RETURNS [Buffer];
MaybeGetFreeReceiveBufferFromPool: PROCEDURE [BufferPool] RETURNS [Buffer];
ReturnBufferToPool: PROCEDURE [BufferPool, Buffer];
ReturnSendBufferToPool: PROCEDURE [BufferPool, Buffer];
ReturnReceiveBufferToPool: PROCEDURE [BufferPool, Buffer];
ReturnBufferToCorrectPool: PROCEDURE [Buffer];
BuffersLeftInPool: PROCEDURE [BufferPool] RETURNS [CARDINAL];
SendBuffersLeftInPool: PROCEDURE [BufferPool] RETURNS [CARDINAL];
ReceiveBuffersLeftInPool: PROCEDURE [BufferPool] RETURNS [CARDINAL];
EnumerateBuffersInPool: PROCEDURE [BufferPool, PROCEDURE[Buffer]];

-- From SystemPool
GetFreeBuffer: PROCEDURE RETURNS [Buffer];
ReturnFreeBuffer: PROCEDURE [Buffer];
BuffersLeft: PROCEDURE RETURNS [CARDINAL];
EnumerateBuffers: PROCEDURE [PROCEDURE[Buffer]];
AdjustBufferParms: PROCEDURE [bufferPoolSize, bufferSize: CARDINAL];
GetBufferParms: PROCEDURE RETURNS [bufferPoolSize, bufferSize: CARDINAL];

DataWordsPerRawBuffer: PROCEDURE RETURNS [CARDINAL];
DataWordsPerPupBuffer: PROCEDURE RETURNS [CARDINAL];
DataWordsPerOisBuffer: PROCEDURE RETURNS [CARDINAL];
DataWordsPerSppBuffer: PROCEDURE RETURNS [CARDINAL];

defaultSystemBufferPoolSize: CARDINAL = 11;
defaultSystemBuffersToReserve: CARDINAL = 2;
defaultDataWordsPerSystemBuffer: CARDINAL = MAX[
    PupTypes.maxDataWordsPerGatewayPup,
    OISCPTypes.maxDataWordsPerSpp];
```

```
-- BUFFER
BufferType: TYPE = {raw, pup, rpp, ois, rejected, processed};
BufferSize: TYPE = {big, small, user};

Buffer: TYPE = LONG POINTER TO BufferObject;
BufferObject: TYPE = MACHINE DEPENDENT RECORD [
  next: PRIVATE Buffer,
  queue: PRIVATE Queue,
  pool: PRIVATE BufferPool,
  requeueProcedure: PROCEDURE [Buffer],
  seal: PRIVATE DriverTypes.Seal,
  length: PRIVATE CARDINAL,
  network: PRIVATE LONG POINTER,  -- DriverDefs.Network
  iocbChain: PRIVATE LONG POINTER,
  userPtr: PRIVATE LONG POINTER,
  userData: PRIVATE LONG POINTER,
  userDataLength: PRIVATE CARDINAL,
  status: PRIVATE WORD,
  time: PRIVATE LONG CARDINAL,
  tries: PRIVATE CARDINAL,
  allNets: PRIVATE BOOLEAN,
  bypassZeroNet: PRIVATE BOOLEAN,
  device: PRIVATE DriverTypes.DeviceType,
  type: BufferType,
  size: BufferSize,
  sendInProgress: BOOLEAN,
  spare: [0..17B],
  bufferNumber: CARDINAL,
  debug: PRIVATE UNSPECIFIED,
  sendCompleted: CONDITION,
  encapsulation: PRIVATE DriverTypes.Encapsulation,


  bufferBody: SELECT OVERLAID BufferType FROM
    raw => [ rawWords: ARRAY [0..0) OF WORD ],

    pup => [
      pupLength: CARDINAL,  -- in bytes, includes header
      pupTransportControl: Byte,  -- 4 bits of hopcount, 4 spares
      pupType: PupTypes.PupType,
      pupID: PupTypes.Pair,
      dest, source: PupAddress,
      pupBody: SELECT OVERLAID * FROM
        pupWords => [ pupWords: ARRAY [0..0) OF WORD ],
        pupBytes => [ pupBytes: PACKED ARRAY [0..0) OF Byte ],
        pupChars => [ pupChars: PACKED ARRAY [0..0) OF CHARACTER ],
        pupString => [ pupString: StringBody ],
        rfc => [ address: PupAddress ],
        ack => [
          maximumBytesPerPup: CARDINAL,
          numberOfPupsAhead: CARDINAL,
          numberOfBytesAhead: CARDINAL ],
        abort => [
          abortCode: WORD,
          abortText: PACKED ARRAY [0..0) OF CHARACTER ],
        error => [  -- see [MAXC]<PUP>Error.ears
          errorHeader: ARRAY [0..9] OF WORD,
          errorCode: PupTypes.PupErrorCode,
          errorOptions: WORD,
          errorText: PACKED ARRAY [0..0) OF CHARACTER ],
        ENDCASE ],
      -- software checksum (1 word) comes after all the data

    rpp => [  -- variant of Pup  NOTE: header is one word longer
      rppPupLength: CARDINAL,  -- in bytes, includes header
      rppTransportControl: Byte,
      rppPupType: [0..37B],  -- 4
      system, sendAck, notify: BOOLEAN,
      seqNumber, ackNumber: CARDINAL,
      rppDest, rppSource: RppAddress,
      allocation, subType: Byte,
      rppBody: SELECT OVERLAID * FROM
        rppWords => [ rppWords: ARRAY [0..0) OF WORD ],
        rppBytes => [ rppBytes: PACKED ARRAY [0..0) OF Byte ],
        rppChars => [ rppChars: PACKED ARRAY [0..0) OF CHARACTER ],
        rppString => [ rppString: StringBody ],
        ENDCASE ],
```

```
        -- software checksum (1 word) comes after all the data
```

```
    ois => [
      checksum: CARDINAL,
      oisPktLength: CARDINAL,  -- in bytes, includes header
      oisTransportControl: Byte,
      packetType: OISCPTypes.OisPacketType,
      destination, source: OisAddress,
      oisBody: SELECT OVERLAID * FROM
        oisWords => [ oisWords: ARRAY [0..0) OF WORD ],
        oisBytes => [ oisBytes: PACKED ARRAY [0..0) OF Byte ],
        oisChars => [ oisChars: PACKED ARRAY [0..0) OF CHARACTER ],
        error => [
          errorType: CARDINAL,
          errorParameter: CARDINAL,
          errorBody: ARRAY [0..0) OF WORD],
        spp => [ -- first word of level 2 Sequenced Protocol
          systemPacket: BOOLEAN,
          sendAck: BOOLEAN,
          attention: BOOLEAN,
          unusedType: [0B..37B],
          subtype: Byte,
          sourceConnectionID: OisConnectionID,
          destinationConnectionID: OisConnectionID,
          sequenceNumber: CARDINAL,
          acknowledgeNumber: CARDINAL,
          allocationNumber: CARDINAL,
          sppBody: SELECT OVERLAID * FROM
            sppWords => [ sppWords: ARRAY [0..0) OF WORD ],
            sppBytes => [ sppBytes: PACKED ARRAY [0..0) OF Byte ],
            sppChars => [ sppChars: PACKED ARRAY [0..0) OF CHARACTER ],
            sppString => [ sppString: StringBody ],
            ENDCASE ],
        ENDCASE ],
    ENDCASE ];

END.

LOG

Time: October 12, 1980  4:48 PM By: Garlick  Action: Changed defaultSystemBufferPoolSize from 15 to 11.
```

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

ByteBlt: DEFINITIONS =

BEGIN

ByteBlt: PROCEDURE [to, from: Environment.Block] RETURNS [nBytes: CARDINAL];
    -- Moves as much as it can, and tells you how much that was.  0 length in either to or from is ok, it will just return 0.  If you
        startIndex > stopIndexPlusOne in either to or from, it will complain since that doesn't make sense.  Does a BLT for the easy
        case, otherwise is uses BitBlt to do the ripple

StartIndexGreaterThanStopIndexPlusOne: ERROR;

END.
```

LOG

Time: July 9, 1979 12:03 PM By: Gobbel Action: Created file
Time: January 25, 1980 4:04 PM By: McJones Action: Add StartIndexGreaterThanStopIndexPlusOne

-- This interface defines operations on regions and swap units and higher-level access to region descriptors. The reader is assumed to be familiar with Virtual Memory Management as described in the Pilot Functional Specification.

-- Definitions:

-- Swap units are the units of storage which are swapped in or out as a unit. They are of uniform size within a region, except that the last one may be short.

-- If a region hasSwapUnits, *a swap unit is the subregion containing* pageMember *which starts on an even* sizeSwapUnit *boundary; if it does not, a swap unit is the whole region.*

-- Invariants:

-- If any swap unit of a region is in, its descriptor is in the cache.

-- If a region is inPinned, *at least one of its swap units are in.*

-- If a swap unit is in, and its region is inPinned, *the swap unit will be continuously present at its assigned address.*

-- If a region is outDead, *all of its swap units are* outDead. *Currently, if a region* hasSwapUnits, *we do not ever mark it* outDead.

-- If a region is unMapped, *all of its swap units are out.*

-- If a region hasSwapUnits *and its descriptor is absent from the cache, its* state = unmapped.

-- If a region has ~writeProtected AND ~needsLogging, *at least one of its swap units has the pageFlags set to allow writing.*

-- If a region has protection = readOnly, *at least one of its swap units has the pageFlags set to disallow writing.*

-- If a region is beingFlushed, *at least one of its swap units are out.*

-- Caveats:

-- The proper operation of remap *and* unmap *is dependent on there being only one client operating on a region at a time (exclusive of page faults).*

-- For clean[andWriteProtect:TRUE], clean[andNeedsLogging:TRUE], flush, remap, *and* unmap *the caller must* Apply *the particular operation (successfully) once for each swap unit in the region. In addition, in the case of* flush, flush *must be repetitively called in ascending order of swap unit.*

-- The current implementation demands that there be only one space being remapped at any given time!

-- Notes:

-- If a space is being remapped, there are two possible windows in which each swap unit in the space may reside.

-- A swap unit which is In *may nevertheless have zero real pages assigned, if it is past the end of the mapping file.*

```
DIRECTORY
    CachedSpace USING [Desc, Level],
    Utilities USING [BitIndex],
    VM USING [Interval, PageCount, PageNumber];

CachedRegion: DEFINITIONS =

BEGIN

Desc: TYPE = RECORD [                          -- A Region Descriptor.
    interval: VM.Interval,
    level: CachedSpace.Level,                  -- level of most-deeply-nested containing space.
    levelMapped: CachedSpace.Level,            -- level of mapping space (assuming state IN Mapped).
    hasSwapUnits: BOOLEAN,                     -- is region divided into uniform swap units?
    dTemperature: DTemperature,                -- for cache replacement algorithm state (private to impl.)
    dPinned: BOOLEAN,                          -- descriptor ineligible for replacement?
    dDirty: BOOLEAN,                           -- descriptor modified since cached? (actually, have any of the parts of the descriptor
                                               --   which are also in a projection STLeaf.Desc been modified since cached?)
    state: State,
    writeProtected: BOOLEAN,                   -- Does the client have permission to write into the region?
    needsLogging: BOOLEAN,                     -- Some swap units of the region are pageMap-writeProtected because the region is part of
                                               --   a transaction and should be writable, but not all swap units have been saved in the
                                               --   TransactionLog yet.
    beingFlushed: BOOLEAN ];                   -- "although the Desc is in the cache, it should be considered missing."

DTemperature: TYPE = [0..4);                   -- should have 2^n values (assuming state IN Mapped).
```

```
State: TYPE = {
    missing,                                    -- descriptor is missing from cache (and therefore, the region from memory)
    checkedOut,                                 -- descriptor is in cache, but checked-out
    -- Descriptor available:
        unmapped,
        -- Mapped:
            outDead,                            -- contents of backing window may be ignored.
                -- Alive:
                    -- AliveSwappable:
                    outAlive,
                        -- In:
                        inSwappableColdest, inSwappable2, inSwappable3, inSwappableWarmest,
                inPinned};                      -- (only if space or ancestor is pinned)
```

-- *If a region* hasSwapUnits *and* region.state *IN* AliveSwappable, *the specific value* of state *is meaningless. In this case, the* State *is determined from the page flags.*

-- *If a region* hasSwapUnits *and* region.state = inPinned, outDead, or unMapped, region.state *shows the target state for all of the swap units.*

```
DescAvailable:    TYPE = State[unmapped..inPinned];

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

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

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

Alive:            TYPE = Mapped[outAlive..inPinned];

AliveSwappable:   TYPE = Swappable[outAlive..inSwappableWarmest];

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

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

-- *The following type is not used in a Desc at present, but may be useful elsewhere:*
-- *(It reflects that fact that that the state* writeProtected AND needsLogging *does not occur.)*

```
Protection: TYPE = {                            -- write-protected state of swap units of region.
    writable,                                   -- the client has permission to write into the region.
    readOnly,                                   -- the client does not have permission to write into the region.
    needsLogging };                             -- some swap units of the region are pageMap-writeProtected because the region is part of
                                                   a transaction and should be writable, but not all swap units have been saved in the
                                                   TransactionLogFile yet.
```

```
Insert: PROCEDURE [desc: Desc] RETURNS [descVictim: Desc];
    -- inserts a region descriptor into the region cache.
    -- desc.dPinned and desc.dDirty are significant; desc.dTemperature is ignored.
    -- If descVictim.dDirty, then descVictim must be written through to the higher level database.
```

```
Apply: PROCEDURE [pageMember: VM.PageNumber, operation: Operation]
                RETURNS [outcome: Outcome, pageNext: VM.PageNumber];
    -- Sets pageNext to the start of the next swap unit (or next region if this is the last swap unit of the region) if a descriptor for the
       region containing pageMember is in the region descriptor cache; else sets pageNext to start of the next region in the cache
       (possibly having state = missing); if no next region, sets pageNext to pageTop.
    -- operation acts on the swap unit containing pageMember, except that createSwapUnits, deleteSwapUnits, get, map, and unpin
       act on the entire region (and return pageNext = the start of the next region).

pageTop: VM.PageNumber;

Outcome: TYPE = RECORD [SELECT kind: * FROM
    ok => NULL,
    notePinned => [levelMax: CachedSpace.Level], -- the region was pinned on entry. Only from unpin. (The requested action was done).
    regionDMissing => NULL, -- region desc not in region cache. (The requested action was not done.)
    regionDDirty => NULL, -- only from flush. (The requested action was not done.)
    retry => NULL, -- Please try again! Only from copyIn and copyOut. (The requested action was not done.)
    spaceDMissing => [level: CachedSpace.Level], -- space desc not in space cache. (The requested action was not done.)
```

```
    error => [state: CachedRegion.State], -- (The requested action was not done.)
    ENDCASE];

ActivatePagefault: TYPE = {activate, pagefault};

BackFileType: TYPE = {file, data, none};

ReportSkip: TYPE = {report, skip};

ReturnWait: TYPE = {return, wait};

maxRemapSpaceSize: CARDINAL = LAST[Utilities.BitIndex]-FIRST[Utilities.BitIndex] + 1;
PageLocationInSpace: TYPE = LONG DESCRIPTOR FOR ARRAY [0..0) OF WORD; -- (one bit per page of space being remapped. Currenly limited to
        maxRemapSpaceSize bits maximum.)
pageLocationInSpace: PageLocationInSpace; -- supplied to Apply by caller of remap for the duration of the Space.Remap operation.
oldSpaceCountMapped: VM.PageCount; -- supplied to Apply by caller of remap for the duration of the Space.Remap operation.


Operation: TYPE = RECORD [
    ifMissing: ReportSkip, -- "if descriptor is missing from region desc cache"
    ifCheckedOut: ReturnWait, -- "if someone else is operating on region"
    afterForking: ReturnWait, -- wait if any i/o started?
    vp: SELECT action: * FROM
        activate => [why: ActivatePagefault], -- get swap unit in. activate[why:activate] is a hint and is legal on unmapped regions.
        age => NULL, -- make swap unit older. If old enough, swap it out.
        clean => [andWriteProtect: BOOLEAN, andNeedsLogging: BOOLEAN], -- make sure contents of window reflect current content of
            region. A no-op on pinned regions. The caller must call clean once successfully (outcome = [ok[]]) for each swap unit
            of the region!
        copyIn => [from: POINTER TO CachedSpace.Desc], -- input from other than the normal backing file. Sets needsLogging to FALSE.
        copyOut => [to: POINTER TO CachedSpace.Desc], -- output to other than the normal backing file.
        createSwapUnits => NULL, -- mark the (whole) region "has swap units".
        deactivate => NULL, -- write this swap unit out. This is a hint.
        flush => NULL, -- flush all swap units of the region from memory and the region descriptor from the cache. If outcome = [regionDDirty]]. no
            action was taken. The caller must, in ascending order, call flush once successfully (outcome = [ok[]]) for each
            swap unit of the region!
        get => [andResetDDirty: BOOLEAN, pDescResult: POINTER TO Desc], -- "get (whole) region descriptor"
        kill => [andDeallocate: BOOLEAN], -- mark swap unit contents "of no value". This is a hint. If andDeallocate, deallocate any memory now
            behind the swap unit. This part is not a hint: it is always done.
        makeWritable => NULL, -- turn off write protect. The caller should have write access to the backing file. Sets needsLogging to FALSE.
        map => [level: CachedSpace.Level, backFileType: BackFileType, andWriteProtect: BOOLEAN, andNeedsLogging: BOOLEAN], --
            map the (whole) region. If backFileType = none, allocates real memory and pins it.
        noOp => NULL,
        pin => NULL, -- get the swap unit in, then pin into memory all swap units of the region which are in. OK to pin a swap unit of an already pinned
            region.
        remapA => [firstClean: BOOLEAN], -- (implicit parameter: pageLocationInSpace.) Mark the location of the swap unit to be "in old window".
            The caller must call remapA once successfully (outcome = [ok[]]) for each swap unit of the region! The current
            implementation demands that there be only one space being remapped at any given time!
        remapB => [from: POINTER TO CachedSpace.Desc], -- (implicit parameter: pageLocationInSpace.) Make sure that the swap unit is in the
            new window, or is In and dirty. Sets needsLogging to FALSE. The caller must call remapB once successfully
            (outcome = [ok[]]) for each swap unit of the region!
        unmap => NULL, -- writes out the swap unit. If it is the last swap unit of the region, also marks the region unmapped. If region was pinned on
            entry, does the work and returns outcome = [notePinned[...]]. The caller must call unmap once successfully
            (outcome = [ok[]] or [notePinned[...]]) for each swap unit of the region!
        unpin => NULL, -- unpin the whole region. OK to unpin an unpinned region.
        ENDCASE];
```

```
-- Standard region operations
-- name:          mumble Operation          = [ifMissing,  ifCkdOut,  afterForking,  action[other parameters]]
activate:         activate Operation        = [report,     return,    return,        activate[why: activate]];
age:              age Operation             = [skip,       return,    return,        age[]];
createSwapUnits:  createSwapUnits Operation = [report,     wait,      wait,          createSwapUnits[]];
deactivate:       deactivate Operation      = [skip,       return,    return,        deactivate[]];
flush:            flush Operation           = [skip,       wait,      wait,          flush[]];
forceOut:         clean Operation           = [skip,       wait,      wait,          clean[andWriteProtect: FALSE,
                                                                                          andNeedsLogging: FALSE]];
invalidate:       kill Operation            = [report,     wait,      return,        kill[andDeallocate: TRUE]];
kill:             kill Operation            = [report,     wait,      return,        kill[andDeallocate: FALSE]];
needsLogging:     clean Operation           = [report,     wait,      wait,          clean[andWriteProtect: TRUE,
                                                                                          andNeedsLogging: TRUE]];
makeWritable:     makeWritable Operation    = [report,     wait,      wait,          makeWritable[]];
pin:              pin Operation             = [report,     wait,      wait,          pin[]];

pagefault:        activate Operation        = [report,     return,    return,        activate[why: pagefault]];
-- Note that pagefault.ifCheckedOut = return. In this case, the faulting process is left sleeping by the pageFault handling process. This
   requires that, whenever any Desc is checked in, processes waiting on pageFaults on that region must be restarted.

startForceOut:    clean Operation           = [skip,       wait,      return,        clean[andWriteProtect: FALSE,
                                                                                          andNeedsLogging: FALSE]];
startUnmap:       unmap Operation           = [report,     wait,      return,        unmap[]];
unmap:            unmap Operation           = [report,     wait,      wait,          unmap[]];
unpin:            unpin Operation           = [report,     wait,      return,        unpin[]; -- actually, ifMissing = > error
writeProtect:     clean Operation           = [report,     wait,      wait,          clean[andWriteProtect: TRUE,
                                                                                          andNeedsLogging: FALSE]];
wait:             noOp Operation            = [skip,       wait,      wait,          noOp[]];

END.
```

Notes

The level and levelMapped fields can be more compactly encoded by observing that if state IN Mapped, levelMapped is IN (1..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*
Time: August 30, 1979 1:46 PM	*By: Knutsen*	*Added backFileType to map operation. Added writeProtect to Desc (not utilized yet). Added noOp, startOut, startUnmap, and wait Operations.*
Time: October 15, 1979 4:05 PM	*By: Knutsen*	*Action: Added hasSwapUnits and beingFlushed to Desc. Added createSwapUnits and deleteSwapUnits. Modified parameters of remapA. Added pageLocationInSpace and remapOldSpaceD. Documented the interface.*
Time: January 21, 1980 10:13 AM	*By: Knutsen*	*Action: Replaced remapOldSpaceD with oldSpaceCountMapped. Improved documentation.*
Time: March 28, 1980 3:11 PM	*By: Knutsen*	*Action: Deleted deleteSwapUnits. Added outputSpecial.*
Time: May 20, 1980 4:50 PM	*By: Gobbel*	*Action: Added needsToBeLogged field to Desc,* retry *to Outcome,* from *field to outputSpecial variant of Operation.*
Time: July 31, 1980 10:50 AM	*By: Knutsen*	*Action: Added* needsLogging *parameter to Desc and various operations. Added* andDeallocate *to* kill *and* invalidate *operation. Added* Protection.

```
-- This interface defines some operations on the region cache.  They should be used only by modules in the Swapper.

-- Some CachedRegionInternal procedures are involved in recovering from the frame heap becoming exhausted.  Because of this,
   invoking these procedures must not cause any frame allocations.  This means that these procedures (and any that they call) must be
   INLINEs or coroutines.  Since there is no such thing as an ENTRY coroutine, it must be simulated by an ENTRY INLINE procedure (which
   acquires the monitor lock) which itself calls the coroutine.  To have an ENTRY INLINE PROCEDURE, the monitor lock must be available in
   the DEFINITIONS module.  To allow a coroutine to be called as a public procedure, the procedure descriptor must be bundled into a
   record to force it to be a procedure variable.

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

CachedRegionInternal: DEFINITIONS
    LOCKS regionCacheLock =

BEGIN

checkIn: CONDITION;  -- broadcast whenever a region descriptor is checked in.

AwaitNotCheckedOut: PROCEDURE [pageMember: VM.PageNumber];

AllocateMStoreRuthlessly: ENTRY PROCEDURE [interval: VM.Interval] = INLINE
    -- Allocates and maps real memory to all of the specified interval.
    -- Guaranteed not to do an ALLOC from the frame heap.
    BEGIN allocateMStoreRuthlesslyInternal[interval] END;

CheckIn: PROCEDURE [desc: CachedRegion.Desc];
    -- terminates caller's exclusive access to region.
    -- If desc.state = missing, the descriptor is deleted from the cache.

CheckOut: PROCEDURE [pageMember: VM.PageNumber, ifCheckedOut: CachedRegion.ReturnWait]
                RETURNS [desc: CachedRegion.Desc];
    -- gives exclusive access to region containing pageMember.
    -- If returned desc.state = missing, interval.page + interval.count gives start of the next region in the cache (which may have
        state = missing).

LongMoveUp: PROCEDURE [pSource: LONG POINTER, size: CARDINAL, pSink: LONG POINTER] = INLINE
    -- Move the contents of size words starting at pSource to the size words starting at pSink.  pSink must be > = pSource to avoid
            clobbering source field
    -- Similar to version in UtilitiesImpl, but INLINE.
    BEGIN
    i: CARDINAL;
    FOR i DECREASING IN [0..size) DO (pSink + i)↑ ← (pSource + i)↑ ENDLOOP
    END;

regionCacheLock: PRIVATE MONITORLOCK;

AllocateMStoreRuthlesslyInternal: PRIVATE TYPE = RECORD[
    proc: PROCEDURE [interval: VM.Interval]];
allocateMStoreRuthlesslyInternal: PRIVATE AllocateMStoreRuthlesslyInternal;

END.
```

LOG

Time: September 26, 1979 11:32 AM By: Knutsen Action: Created file.
Time: March 18, 1980 2:49 PM By: Knutsen Action: Added AllocateMStoreRuthlessly, LongMove.

```
DIRECTORY
    Space USING [WindowOrigin],
    Transaction USING [Handle],
    VM USING [Interval, PageCount, PageNumber];

CachedSpace: DEFINITIONS
    IMPORTS Transaction =

BEGIN

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

SizeSwapUnit: TYPE = [VM.PageCount[0]..VM.PageCount[64]);   -- (note: 0 is an illegal value.)

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.
    hasSwapUnits: BOOLEAN,
    sizeSwapUnit: SizeSwapUnit,                         -- assuming hasSwapUnits.
    dataOrFile: DataOrFile,
    pageRover: VM.PageNumber,                           -- for Space.Create with default base.
    vp: SELECT OVERLAID * FROM
        short => NULL,
        long => [
            window: WindowOrigin,
            countMapped: VM.PageCount,
            transaction: Transaction.Handle],           -- Transaction.nullHandle here means no transaction
        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];
    -- Inserts a Desc into the space cache.
    -- 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];
    -- "Get descriptor." If not found, returns Desc[..., state: missing, ...].

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 1, 1978 2:25 PM *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*
Time: August 27, 1979 10:10 AM *By: Knutsen Action: Improved documentation.*
Time: September 14, 1979 5:33 PM *By: Knutsen Action: Added hasSwapUnits, sizeSwapUnit.*
Time: April 11, 1980 12:30 PM *By: Knutsen Action: Allow SizeSwapUnit[0].*
Time: May 20, 1980 4:44 PM *By: Gobbel Action: Added transaction handle to Desc.*

```
DIRECTORY
 MiscAlpha USING [aCHKSUM],
 Mopcodes USING [zADD, zDADD, zDUP, zEXCH, zINC, zLI0, zMISC];

Checksum: DEFINITIONS =
BEGIN

-- Produce checksum for nWords starting at p.  Start with initial value cs (useful if forming a single checksum for discontinuous
areas of memory).
ComputeChecksum: PROC [
 cs: CARDINAL ← 0, nWords: CARDINAL, p: LONG POINTER]
 RETURNS [checksum: CARDINAL] =
 MACHINE CODE BEGIN Mopcodes.zMISC, MiscAlpha.aCHKSUM; END;

-- ComputeChecksumSoftware is provided as documentation.
ComputeChecksumSoftware: PRIVATE PROC [
 cs: CARDINAL ← 0, nWords: CARDINAL, p: LONG POINTER]
 RETURNS [checksum: CARDINAL] =
 INLINE BEGIN
 THROUGH [0..nWords) DO
  t: CARDINAL;
  -- cs ← ((cs ONESADD p↑) LEFTROTATE 1)
  IF cs > (t ← cs + p↑) THEN cs ← t + 1 ELSE cs ← t;
  IF cs >= 100000B THEN cs ← cs*2 + 1 ELSE cs ← cs*2;
  p ← p + 1;
  ENDLOOP;
 IF cs = 177777B THEN cs ← 0;
 RETURN[cs];
 END;

-- ComputeChecksumProc is provided as an aid to processors required to compute checksum as part of their unimplemented
instruction trap.
ComputeChecksumProc: PRIVATE PROC [
 cs: CARDINAL ← 0, nWords: CARDINAL, p: LONG POINTER]
 RETURNS [checksum: CARDINAL] =
 INLINE BEGIN
 Push: PROC [CARDINAL] = MACHINE CODE BEGIN END;
 Pop: PROC RETURNS [CARDINAL] = MACHINE CODE BEGIN END;
 TOS: PROC RETURNS [CARDINAL] = MACHINE CODE BEGIN Mopcodes.zDUP; END;
 IncTOS: PROC = MACHINE CODE BEGIN Mopcodes.zINC; END;
 OnesAddAndLeftRotate: PROC [
  --cs: CARDINAL,-- zero: CARDINAL ← 0, wd: CARDINAL, zero1: CARDINAL ← 0]
  --RETURNS [cs: CARDINAL]-- =
  MACHINE CODE BEGIN
  Mopcodes.zDADD; Mopcodes.zADD; -- cs ← (cs ONESADD wd)
  Mopcodes.zDUP; Mopcodes.zLI0; Mopcodes.zEXCH; Mopcodes.zLI0; -- Long[cs], Long[cs]
  Mopcodes.zDADD; Mopcodes.zADD;  -- Rotate via long add to self, merging carry
  END;
 --TOS←-- Push[cs];  -- leave cs on the stack throughout the procedure
 THROUGH [0..nWords) DO
  --TOS←-- OnesAddAndLeftRotate[--cs: TOS,-- wd: p↑];
  p ← p + 1
  ENDLOOP;
 IF TOS[] = 177777B THEN IncTOS[];
 RETURN[Pop[--TOS--]];
 END;


 END....
```

```
DIRECTORY
 BufferDefs USING [OisBuffer],
 MiscAlpha USING [aCHKSUM],
 Mopcodes USING [zMISC];

Checksums: DEFINITIONS =
 BEGIN

 -- This procedure sets the checksum field of the Ois Packet.
 SetChecksum: PUBLIC PROCEDURE [b: BufferDefs.OisBuffer];

 -- This procedure checks the checksum field of the Ois Packet,
 -- and returns TRUE or FALSE.  The buffer is always a system buffer
 TestChecksum: PUBLIC PROCEDURE [b: BufferDefs.OisBuffer] RETURNS [BOOLEAN];

 -- This procedure increments the oisTransportControl field of the buffer b and
 -- updates the checksum field to reflect the update.
 -- The buffer is always a system buffer.
 IncrOisTransportControlAndUpdateChecksum: PUBLIC PROCEDURE [b: BufferDefs.OisBuffer];

 -- This procedure computes the checksum.
 ComputeChecksum: PROCEDURE [cs: CARDINAL, nWords: CARDINAL, p: LONG POINTER]
  RETURNS [CARDINAL] =
 MACHINE CODE BEGIN
 Mopcodes.zMISC, MiscAlpha.aCHKSUM;
 END;

 END.
```

-- **CommunicationInternal**.mesa *(last edited by: BLyon on: August 20. 1980 10:37 AM)*

```
CommunicationInternal: DEFINITIONS =
BEGIN
-- the interface to Pilot.
PilotCommUtil: PROGRAM;
-- the main OIS Router and Socket modules.
ChecksumsImpl: PROGRAM;
RouterImpl: PROGRAM;
RoutingTableImpl: PROGRAM;
SocketImpl: PROGRAM;
EchoServerImpl: PROGRAM;
-- the OIS Packet and Network Stream managers.
PacketStreamMgr: PROGRAM;
NetworkStreamMgr: PROGRAM;
END.
```

LOG
Time: July 7, 1979 1:44 PM By: Dalal Action: Created file.
Time: January 4, 1980 2:51 PM By: Dalal Action: Changed SocketMgr to SocketImpl.
Time: January 4, 1980 2:51 PM By: Dalal Action: Changed OISCPRouter to RouterImpl and RoutingTableImpl.
Time: January 27, 1980 10:22 AM By: Dalal Action: added PacketStreamMgr.
Time: August 20, 1980 10:37 AM By: BLyon Action: added ChecksumsImpl.

```
CommunicationPrograms: DEFINITIONS =

BEGIN

InitializeCommunication: PROCEDURE; -- EXPORTed by CommunicationControl.

END.
```

LOG

Time: August 18, 1978 11:37 AM By: Redell
Action: Created file

Time: August 18, 1978 11:37 AM By: Knutsen
Action: CommunicationControl: PROGRAM changed to InitializeCommunication[].

```
-- File: CommUtilDefs.Mesa,  Last Edit: BLyon  July 29, 1980  1:17 PM

DIRECTORY
  Mopcodes USING [zLLB],
  PrincOps USING [FrameHandle, returnOffset];

CommUtilDefs: DEFINITIONS =
BEGIN

-- for finding out where we really are
thisIsAnAlto: BOOLEAN = FALSE;

-- Buffer/IOCB Allocation
AllocateBuffers: PROC [nwords: CARDINAL] RETURNS [base: LONG POINTER];
FreeBuffers: PROC [base: LONG POINTER];
LockBuffers: PROC [LONG POINTER];  -- noop in Alto world
UnlockBuffers: PROC [LONG POINTER];  -- noop in Alto world
AllocateIocbs: PROC [CARDINAL] RETURNS [LONG POINTER];
FreeIocbs: PROC [LONG POINTER];

-- Lock Interrupt routines
LockCode: PROC [UNSPECIFIED] = INLINE { };  -- useful in Alto world
UnlockCode: PROC [UNSPECIFIED] = INLINE { };  -- useful in Alto world


-- This is needed because of Process/ProcessDefs LONG/Short complications
-- Shortens in the Alto world, noop in Pilot world
MaybeShorten: PROC [p: LONG POINTER] RETURNS [LONG POINTER] =
  INLINE { RETURN[p] };

GetEthernetHostNumber: PROC RETURNS [CARDINAL];

-- Debugging things
FrameHandle: TYPE = PrincOps.FrameHandle;
GetReturnFrame: PROC RETURNS [FrameHandle] = MACHINE CODE
  BEGIN Mopcodes.zLLB, PrincOps.returnOffset END;

-- Other goodies
Zero: PROC [POINTER, CARDINAL];
CopyLong: PROC [from: LONG POINTER, nwords: CARDINAL, to: LONG POINTER];

-- This is a hack for debugging
SetDebuggingPointer: PROC [LONG POINTER] = INLINE { };  -- useful in Alto world

END.....
July 19, 1980  3:33 PM  By: Forrest     Action: Removed references to Interupt levels.  Remove pointer into
middle of page 1.  Change ControlDefs to Runtime.
```

```
ControlPrograms: DEFINITIONS =

BEGIN

SystemImpl: PROGRAM;

InitializeGMT: PROCEDURE;
    -- Temporary, until clock chip arrives.  This procedure request time from server on Ethernet.  Should only be used
            when Communication is known to be inactive, i.e. before it is started or after inLoad with interrupts and
            devices turned off.

END.
```

LOG

Time: February 23, 1979 11:48 AM By: Horsley Action: Created file
Time: August 30, 1979 6:05 PM By: McJones Action: Added InitializeGMT

```
-- File: CountPrivate.mesa
-- Edited by Sandman on September 19, 1980  7:54 AM

DIRECTORY
  PerfStructures USING [FrameHandle, PsbHandle];

CountPrivate: DEFINITIONS =
  BEGIN

  VersionID: CARDINAL = 09190;

  Mode: TYPE = {plain, matrix};

  Table: TYPE = RECORD [
    SELECT OVERLAID Mode FROM
      plain => [plain: ARRAY [0..0) OF LONG CARDINAL],
      matrix => [matrix: ARRAY GroupIndex OF ARRAY GroupIndex OF LONG CARDINAL],
      ENDCASE];
  TableHandle: TYPE = POINTER TO Table;

  MaxGroup: CARDINAL = 15;
  GroupIndex: TYPE = CARDINAL [0..MaxGroup];
  GroupTable: TYPE = ARRAY [0..0) OF GroupIndex;
  GroupHandle: TYPE = POINTER TO GroupTable;

  ControlRecord: TYPE = MACHINE DEPENDENT RECORD [
    gfi(0), prevGfi(1): CARDINAL,
    newMeasurement(2): BOOLEAN,
    trace(3): BOOLEAN,
    mode(4): Mode,
    counts(5), times(6): TableHandle,
    groups(7): GroupHandle,
    length(8): CARDINAL,
    process(9): PerfStructures.PsbHandle,
    version(10): CARDINAL,
    saveBreakHandler(11): PerfStructures.FrameHandle,
    self(12): PerfStructures.FrameHandle,
    newSession(13): BOOLEAN,
    pulseConversion(14): LONG CARDINAL ← NULL];

  PCR: TYPE = POINTER TO ControlRecord;

  END..
```

```
-- File: CPSwapDefs.mesa
-- Last edited by Sandman; August 13, 1980  3:16 PM

DIRECTORY
  PrincOps USING [GlobalFrameHandle, StateVector, SVPointer, BytePC];

CPSwapDefs: DEFINITIONS =

  BEGIN

  SwapInfo: POINTER TO POINTER TO ExternalStateVector = LOOPHOLE[456B];

  SVPointer: TYPE = PrincOps.SVPointer;

  ExternalStateVector: TYPE = MACHINE DEPENDENT RECORD [
    state: SVPointer,
    reason: SwapReason, level: [0..3777B],
    parameter: POINTER TO DebugParameter,
    versionident: CARDINAL,
    loadstatepage: CARDINAL,
    lspages: CARDINAL,
    mapLog: LONG POINTER,
    mds: CARDINAL,
    pda: CARDINAL ← 0,
    fill: ARRAY [10..19) OF WORD];

  VersionID: CARDINAL = 08130;

  SwapReason: TYPE = {
    -- handled by user's nub
    proceed,    -- THIS MUST BE FIRST !!
    start,
    call,
    resume,
    quit,
    showscreen,
    kill,

    -- handled by debugger
    install,
    breakpoint,
    worrybreak,
    worrycall,
    uncaughtsignal,
    explicitcall,
    return,
    punt,
    interrupt,
    cleanmaplog,
    addressfault,
    writeprotect,
    spare1,
    spare2,
    spare3
    };

  DebugParameter: TYPE = MACHINE DEPENDENT RECORD [
    string: STRING,
    body: SELECT OVERLAID SwapReason FROM
      uncaughtsignal => [
        msg: UNSPECIFIED,
        signal: UNSPECIFIED],
      return => [
        value: UNSPECIFIED],
      start => [
        frame: PrincOps.GlobalFrameHandle],
      call => [
        sv: PrincOps.StateVector],
      addressfault, writeprotect => [
        page, psbIndex: CARDINAL],
      ENDCASE];

  uncaughtsignalDP: TYPE = POINTER TO uncaughtsignal DebugParameter;
  returnDP: TYPE = POINTER TO return DebugParameter;
  startDP: TYPE = POINTER TO start DebugParameter;
  callDP: TYPE = POINTER TO call DebugParameter;
  addressfaultDP: TYPE = POINTER TO addressfault DebugParameter;
  writeprotectDP: TYPE = POINTER TO writeprotect DebugParameter;
```

```
  GetLevel: PROCEDURE RETURNS [INTEGER];
  SetLevel: PROCEDURE [l: INTEGER];
  CoreSwap: PROCEDURE [why: SwapReason, sp: SVPointer];
  CantSwap: SIGNAL;
  CAbort: SIGNAL;

  -- Conditional Breakpoint Stuff

  BBArray: TYPE = RECORD [
    length: CARDINAL,
    blocks: ARRAY [0..0) OF UserBreakBlock];

  BBHandle: TYPE = POINTER TO BBArray;

  UserBreakBlock: TYPE = RECORD [
    frame: PrincOps.GlobalFrameHandle,
    pc: PrincOps.BytePC,
    ptrL: POINTER,
    ptrR: POINTER,
    posnL: [0..16),
    posnR: [0..16),
    sizeL: [1..16],
    sizeR: [1..16],
    inst: [0..377B],
    relation: Relations,
    immediateR: BOOLEAN,
    counterL: BOOLEAN,
    localL: BOOLEAN,
    localR: BOOLEAN,
    stackRelative: BOOLEAN];

  UBBPointer: TYPE = POINTER TO UserBreakBlock;

  Relations: TYPE = {eq, ne, lt, ge, gt, le};

  END ...
```

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

DOInputOutput: DEFINITIONS =

BEGIN

-- Support for D0 heads (device face implementations)

ControllerNumber: TYPE = [0..16); -- controller number = task number = CSB number
nullControllerNumber: ControllerNumber = 0; -- not a valid I/O controller number

ControllerType: TYPE = [0..377B];                   -- Hardware-defined controller type code
    iutfp: ControllerType = 001B;                   -- Interim User Terminal, Full Page
    utvfc: ControllerType = 002B;                   -- User Terminal Variable Format Controller
    rdc: ControllerType = 003B;                     -- Rigid Disk Controller
    irdc: ControllerType = 004B;                    -- Interim Rigid Disk Controller
    rdcWithServiceLate: ControllerType = 005B;      -- RDC has status bits in ID register!
    ethernet1Out: ControllerType = 006B;            -- Ethernet-1 output
    ethernet1In: ControllerType = 007B;             -- Ethernet-1 input
    eim: ControllerType = 010B;                     -- Electronic Input Module (RIS)
    idc: ControllerType = 011B;                     -- Imaginal Disk Controller (CDC disk)
    uibScb: ControllerType = 012B;                  -- User Interface Board / System Control Board
    fdc: ControllerType = 013B;                     -- Floppy Disk Controller
    ethernetOut: ControllerType = 014B;             -- Ethernet-2 output
    ethernetIn: ControllerType = 015B;              -- Ethernet-2 input
    eomHigh: ControllerType = 016B;                 -- Electronic Output Module (ROS) - higher priority section
    eomLow: ControllerType = 017B;                  -- Electronic Output Module (ROS) - lower priority section
    mioc: ControllerType = 020B;                    -- Miscellaneous I/O Controller
    cmtc: ControllerType = 021B;                    -- Computer Magnetic Tape Controller
    null: ControllerType = 377B;                    -- no controller present
-- Compatibility (delete someday):
    etherOut: ControllerType = ethernet1Out;
    etherIn: ControllerType = ethernet1In;
    xWireOut: ControllerType = ethernetOut;
    xWireIn: ControllerType = ethernetIn;

GetNextController: PROCEDURE [type: ControllerType, prev: ControllerNumber ← nullControllerNumber]
        RETURNS [next: ControllerNumber];
    -- Stateless enumeration of controllers of given type; enumeration begins and ends with nullControllerNumber.
    -- NOTE: When multiple controllers of a given type are present, they are enumerated in order of DECREASING priority (highest
       controller number first).

CSB: TYPE = ARRAY [0..16) OF UNSPECIFIED;
CSBArray: TYPE = ARRAY[0..16) OF CSB;

IOPage: READONLY LONG POINTER TO CSBArray;

IOAddress: TYPE = MACHINE DEPENDENT RECORD [
zero: [0..377B] ← 0,
controller: ControllerNumber,
register: [0..17B]];

Input: PROCEDURE [IOAddress] RETURNS [UNSPECIFIED] = MACHINE CODE BEGIN Mopcodes.zMISC, 5; END;
    -- Read one word from specified controller register.

Output: PROCEDURE [datum: UNSPECIFIED, register: IOAddress] = MACHINE CODE BEGIN Mopcodes.zMISC, 6; END;
    -- Write one word to specified controller register.

END.
```

LOG

Time: August 7, 1979 12:05 PM By: Redell Action: Create file
Time: February 4, 1980 7:10 PM By: McJones Action: Remove RegisterDevice, DeviceHandle, DeviceNumber; change ControllerNumber from 8 to 4 bits; add nullControllerNumber
Time: May 15, 1980 10:54 AM By: McJones Action: Add Input, Output, IOAddress; rename Ethernet controller types

```
DIRECTORY
  Boot: FROM "Boot" USING [Location];

DebuggerSwap: DEFINITIONS =

BEGIN

-- Information controlling the world swap to the debugger.

Parameters: TYPE = RECORD [
    -- used each swap to debugger:
  locDebuggee: Boot.Location, -- where to outLoad
  pMicrocodeCopy, pGermCopy: LONG POINTER, -- swap if non nil
  nGerm: CARDINAL, -- number of words
  locDebugger: Boot.Location, -- where to inLoad
    -- used during initialization:
  pLocMicrocode, pLocGerm: POINTER TO Boot.Location,
  nMicrocode: CARDINAL]; -- number of words

canSwap: BOOLEAN; -- TRUE if parameters has been initialized

parameters: Parameters;

END.
```

LOG

Time: August 6, 1979 12:13 PM *By: McJones Action: Create file*
Time: August 27, 1979 5:40 PM *By: McJones Action: Add pSavedMap, dropped altoDebugger*
Time: September 18, 1979 5:44 PM *By: McJones Action: Move Location from BootChannel to Boot*
Time: April 17, 1980 10:43 AM *By: McJones Action: Delete definitions for CoPilot 0*

```
Device: DEFINITIONS =
BEGIN

-- Definitions common to all devices supporting Pilot boot loaders or a DiskChannel implementation

    -- Device type
    -- Extendible enumeration implemented in conjuction with DeviceTypes, a definitions module containing constants of type
            Device.Type and which should not be included by any other definitions module, thus enabling free recompilation
Type: TYPE = PRIVATE RECORD [UNSPECIFIED];
nullType: Type = [177777B];
END.
```

LOG

Time: January 21, 1980 11:55 AM By: McJones Action: Create file

```
DeviceCleanup: DEFINITIONS =

BEGIN

Reason: TYPE = MACHINE DEPENDENT {
    turnOff(0) , -- stop fetches to memory other than to controller status block
    turnOn, -- inverse of turnOff
    disconnect, -- release resources such as special real memory in preparation for booting different system
    kill, -- prepare for demise of controller microcode
    (255)};

Await: PROCEDURE [pItem: POINTER TO Item] RETURNS [Reason] = INLINE BEGIN RETURN[linkage.Await[pItem]] END;
    -- Wait until next execution of Perform

-- InitializeSampleCleanup: PROCEDURE [<parameters>] =
    -- BEGIN
    -- item: DeviceCleanup.Item;
    -- reason: DeviceCleanup.Reason;
    -- <private state>
    -- <initialization code - may call other procedures>
    -- Following code loops forever - must not call other procedures except INLINE, fixed frame
    -- DO
        -- reason ← DeviceCleanup.Await[@item];
        -- SELECT reason FROM
            -- turnOff => <code to turn device off>;
            -- turnOn => <code to turn device on>;
            -- disconnect => <code to release resources such as special real memory>;
            -- kill => <code to prepare for demise of controller microcode>;
            -- ENDCASE
        -- Any select arms irrelevant for a given device may be omitted.
        -- ENDLOOP
    -- END;

Item: TYPE [2];

Perform: PROCEDURE [reason: Reason] = INLINE BEGIN linkage.Perform[reason] END;
    -- Execute each waiting cleanup procedure, passing given reason
    -- Interrupts should have been previously disabled

Linkage: PRIVATE TYPE = RECORD [
    Await: PROCEDURE [POINTER TO Item] RETURNS [Reason],
    Perform: PROCEDURE [Reason]];
linkage: PRIVATE Linkage;

END.
```

LOG

Time: August 23, 1979 12:07 PM By: McJones Action: Created file
Time: July 8, 1980 6:08 PM By: McJones Action: Add disconnect reason

```
DIRECTORY
    Device: FROM "Device" USING [nullType, Type];

DeviceTypes: DEFINITIONS SHARES Device =

BEGIN

-- Enumeration of Device.Type's for devices supporting Pilot boot loaders or a DiskChannel implementation

-- NOTE: This module contains only constants, and should not be included by any definitions module.  Thus it will always be possible to
recompile it without recompiling all the implementation modules including it.

null:       Device.Type = Device.nullType;
diablo31:   Device.Type = [0];
sa800:      Device.Type = [1]; -- Shugart Associates 8xy
sa1000:     Device.Type = [2];
sa4000:     Device.Type = [3]; -- Shugart Associates 400x
cdc9730:    Device.Type = [4];
ethernet:   Device.Type = [5];

END.
```

LOG

Time: January 21, 1980 4:11 PM By: McJones Action: Create file

```
DIRECTORY
    Environment: FROM "Environment";

DiagnosticPilotClient: DEFINITIONS =

BEGIN

Run: PROCEDURE [firstFree: Environment.PageNumber, countFree: Environment.PageCount];
    -- all free real memory in the system is mapped to virtual pages in the interval [firstFree..firstFree + countFree)

END.
```

LOG

Time: April 4, 1979 4:06 PM By: McJones Action: Created file

```
Dialup: DEFINITIONS =
BEGIN
AbortCall: PROCEDURE [dialerNumber: CARDINAL];
Dial: PROCEDURE [dialerNumber: CARDINAL, number: STRING, retries: RetryCount] RETURNS [Outcome];
RetryCount: TYPE = [0..7];
Outcome: TYPE = { success, failure, aborted, formatError, transmissionError, dataLineOccupied, dialerNotPresent, dialingTimeout,
        transferTimeout };
END.
```

LOG

Time: October 18, 1978 8:35 AM By: Schwartz Action: Created file
Time: March 9, 1979 11:03 AM By: Schwartz Action: Added ***AbortCall***
Time: May 9, 1979 10:07 AM By: Schwartz Action: Added outcome of dialerNotpresent
Time: September 21, 1979 9:49 AM By: Schwartz Action: changes associated with definition of RS366Face.mesa.
Time: November 30, 1979 4:19 PM By: Danielson Action: added timeouts.
Time: January 21, 1980 2:51 PM By: Schwartz Action: added a type for RetryCount.

```
DIRECTORY
    Environment USING [PageNumber],
    Device USING [Type],
    PilotDisk USING [Handle, Label];

DiskChannel: DEFINITIONS =

BEGIN

    -- Drives
DiskPageCount: TYPE = LONG CARDINAL;
DiskPageNumber: TYPE = LONG CARDINAL;
Drive: TYPE = LONG POINTER TO DriveObject;
DriveObject: TYPE;
PVHandle: TYPE = RECORD [drive: Drive, changeCount: CARDINAL]; -- denotes a drive instance - the combination of a drive and the physical
        volume, whether Pilot or non-Pilot, if any, on that drive.
DriveState: TYPE = {inactive, channel, pilot};
DriveStatus: TYPE = {alreadyAsserted, badDisk, hardwareError, invalidDrive, notReady, ok, writeProtected, wrongFormat};
nullDrive: Drive = NIL;

AwaitStateChange: PROCEDURE [count: CARDINAL, type: Device.Type, index: CARDINAL] RETURNS [currentChangeCount: CARDINAL];
GetNextDrive: PROCEDURE [prev: Drive] RETURNS [Drive];
    -- Stateless enumeration of drives, beginning and ending with nullDrive. Drives of given type are grouped together, in increasing
            GetDriveAttrubites[].deviceOrdinal order.
    -- NOTE: a new drive may appear after an enumeration has been started
GetDriveAttributes: PROCEDURE [drive: Drive]
        RETURNS [deviceType: Device.Type, deviceHandle: PilotDisk.Handle, deviceOrdinal: CARDINAL, nPages: DiskPageCount, ready:
        BOOLEAN, state: DriveState, changeCount: CARDINAL];
    -- See DeviceTypes.mesa for possible values of deviceType.
    -- deviceOrdinal gives the position of this drive within all devices of same type.
GetDriveTag: PROCEDURE [drive: Drive] RETURNS [tag: CARDINAL];
SetDriveState: PROCEDURE [drive: Drive, changeCount: CARDINAL, state: DriveState] RETURNS [s: DriveStatus];
    -- A tag may be associated with each drive as a convenience to the client.
SetDriveTag: PROCEDURE [drive: Drive, tag: CARDINAL];

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

    -- Channels
Handle: TYPE = PRIVATE RECORD [UNSPECIFIED];
nullHandle: Handle = [0];
Create: PROCEDURE [drive: Drive, completion: CompletionHandle] RETURNS [Handle];
    -- It is permissable to create more than one channel per drive.
Delete: PROCEDURE [channel: Handle];
GetAttributes: PROCEDURE [channel: Handle] RETURNS [drive: Drive];

Suspend: PROCEDURE [channel: Handle];
    -- Cause any further I/O transactions to wait until the channel is restarted.
Idle: PROCEDURE [channel: Handle];
    -- Suspend channel and wait until I/O transactions already in progress have completed.
Restart: PROCEDURE [channel: Handle];
    -- Allow I/O transactions to proceed.

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

    -- fields defining the IO Request
    command: Command,
    channel: Handle,
    diskPage: DiskPageNumber,
    memoryPage: Environment.PageNumber,
    dontIncrement: BOOLEAN ← FALSE, -- TRUE means use same memory page for all disk pages
    count: DiskPageCount,
    countDone: DiskPageCount,
    label: PLabel,
    tag: CARDINAL,

    -- 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 [
    cylinder: Cylinder,
    head: Head,
    sector: Sector];
Command: TYPE = {vvr, vvw, vrr, vww};
    -- Specifies actions for header, label, data (read/verify/write). Label verification is defined by PilotDisk.MatchLabels. For
            IORequest's with count > 1, the label is automatically updated after each sector using PilotDisk.NextLabel.
Cylinder: TYPE = CARDINAL;
Head: TYPE = [0..256);
Sector: TYPE = [0..256);
DriveNumber: TYPE = CARDINAL;
PLabel: TYPE = LONG POINTER TO Label;
    Label: TYPE = PilotDisk.Label; -- labels should be quadword aligned
Direction: TYPE = {put, get};
CompletionStatus: TYPE = {goodCompletion, noSuchPage, labelDoesNotMatch, seekFailed, checkError, checksumError,
        hardwareError, notReady, notPilotVolume, invalidChannel};

InitiateIO: PROCEDURE [req: IORequestHandle];
    -- Initiate a new I/O transaction, waiting if necessary until the channel is not suspended.
WaitAny: PROCEDURE [completion: CompletionHandle] RETURNS [IORequestHandle];
    -- Wait for the next I/O transaction to complete.

GetPageAddress: PROCEDURE [channel: Handle, page: DiskPageNumber] RETURNS [Address];
    -- Perform seek address calculation.
GetPageNumber: PROCEDURE [drive: Drive, page: Address] RETURNS [DiskPageNumber];
    -- Perform inverse of seek address calculation.

END.
```

LOG

Time: January 8, 1979 7:17 PM By: TH Action: Create file to replace RigidDisk
Time: July 23, 1979 3:06 PM By: Gobbel Action: Replace "reservedForDriver" field of IORequest by "address"
Time: August 2, 1979 2:39 PM By: Redell Action: Simplify interface; eliminate DriveHandle/DriveID distinction, reformatted Addresses, add GetPageAddress, remove all ERRORs, etc. etc. Prepare for run-of-pages by adding count field to IORequest
Time: November 26, 1979 4:08 PM By: Gobbel Action: Replace direction and verifyLabel fields of request by command field, also add tag (for use by FileTaskImpl). Add countDone field to request (more preparation for runs of pages)
Time: December 6, 1979 12:40 AM By: Gobbel Action: Add values to Command type to allow label operations to no-op on data field
Time: December 7, 1979 6:52 PM By: Gobbel Action: Add dontIncrement field to IOrequest
Time: January 25, 1980 6:39 PM By: McJones Action: Eliminate dependence on OISProcessorFace
Time: June 10, 1980 6:44 PM By: Luniewski Action: Add PVHandle, DriveObject (and make Drive be a LONG POINTER to a DriveObject), DriveError, DriveState, ErrorType, SetDriveState; add the ready, state and changeCount results to GetDriveAttributes; add notPilotVolume to CompletionStatus
Time: June 17, 1980 9:30 AM By: McJones Action: OISDisk = > PilotDisk
Time: August 4, 1980 5:41 PM By: Luniewski Action: SetDriveState returns status instead of raising an ERROR. Deleted labelLength.
Time: September 2, 1980 5:15 PM By: Jose Action: Added to DriveStatus.

```
-- THINGS TO DO:
-- 1) Get procedure descriptors out of DriveObject to accommodate multiple MDS's

DIRECTORY
    Device USING [Type],
    DiskChannel USING [Address, DiskPageCount, DiskPageNumber, DriveState, Handle, IORequestHandle],
    PilotDisk USING [Handle];

DiskChannelBackend: DEFINITIONS =

BEGIN

DriveHandle: TYPE = LONG POINTER TO DriveObject;

DriveID: TYPE = RECORD [ -- representation of DiskChannel.Handle
    type: Device.Type,
    handle: PilotDisk.Handle];

DriveObject: TYPE = RECORD [
    driveID: DriveID,
    cylinders, movingHeads, fixedHeads, sectorsPerTrack: CARDINAL,
    -- next field used only in Pilot, as opposed to channel, world
    nPages: DiskChannel.DiskPageCount,
    tag: CARDINAL, -- for exclusive use of the DiskChannel client
    driverStorage: LONG POINTER, -- for exclusive use of the driver
    stateChangeLocked: BOOLEAN, -- locks ability to change state field
    state: DiskChannel.DriveState,
    changeCount: CARDINAL,
    getStatus: GetStatusProc,
    requestIO: RequestIOProc,
    getPageAddress: GetPAProc,
    getPageNumber: GetPNProc,
    changeState: ChangeStateProc,
    -- next two fields for the exclusive use of the DiskChannel implementation:
    driveOrdinal: CARDINAL ← NULL,
    next: PRIVATE DriveHandle ← NULL];
RequestIOProc: TYPE = PROCEDURE [req: DiskChannel.IORequestHandle];
GetPAProc: TYPE = PROCEDURE [dH: DriveHandle, page: DiskChannel.DiskPageNumber] RETURNS [DiskChannel.Address];
GetPNProc: TYPE = PROCEDURE [dH: DriveHandle, page: DiskChannel.Address] RETURNS [DiskChannel.DiskPageNumber];
GetStatusProc: TYPE = PROCEDURE [dH: DriveHandle] RETURNS [ready: BOOLEAN, changeCount: CARDINAL];
        -- Note that the GetStatusProc may cause changes to the DriveObject, in particular, changeCount
ChangeStateProc: TYPE = PROCEDURE [dH: DriveHandle, state: DiskChannel.DriveState];

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

END.
```

LOG

Time: January 15, 1979 2:51 PM By: Horsley Action: Create file
Time: August 15, 1979 5:12 PM By: Redell Action: Change DriveID to Drive
Time: August 16, 1979 10:57 PM By: Redell Action: Add tag field to DriveObject
Time: January 31, 1980 5:41 PM By: McJones Action: Move Device types from OISProcesorFace to Device and Device handles from OISProcessorFace to OISDisk; add driveOrdinal
Time: June 10, 1980 10:30 AM By: Luniewski Action: Add changeCount, state and getStatus to DriveObject; add GetStatusProc
Time: June 23, 1980 4:41 PM By: McJones Action: OISDisk = > PilotDisk
Time: July 19, 1980 2:48 PM By: Jose Action: Add changeState to DriveObject, type ChangeStateProc
Time: August 7, 1980 8:54 AM By: Luniewski Action: Add stateChangeLock to DriveObject.
Time: September 16, 1980 11:52 PM By: Jose Action: Change stateChangeLock to stateChangeLocked, add caveats.

```
DIRECTORY
    DiskChannel: FROM "DiskChannel" USING [Cylinder, IORequestHandle, Sector],
    DiskChannelBackend: FROM "DiskChannelBackend" USING [DriveObject];

DiskDriverShared: DEFINITIONS =

BEGIN

ScheduleHandle: TYPE = LONG POINTER TO ScheduleObject;
ScheduleObject: TYPE = RECORD [
    drive: DiskChannelBackend.DriveObject,
    currentSector: DiskChannel.Sector,
    currentCylinder: DiskChannel.Cylinder,
    first: DiskChannel.IORequestHandle,
    previousRequest: DiskChannel.IORequestHandle];

GetNextPendingRequest: PROCEDURE [schedule: ScheduleHandle] RETURNS [DiskChannel.IORequestHandle];

PeekNextPendingRequest: PROCEDURE [schedule: ScheduleHandle] RETURNS [DiskChannel.IORequestHandle];

InsertRequest: PROCEDURE [req: DiskChannel.IORequestHandle];

END.
```

LOG

Time: February 2, 1979 4:31 PM By: Horsley Action: Create file
Time: July 24, 1979 3:32 PM By: Gobbel Action: Delete AddressEtc, get Sector, Track, etc., from DiskChannel
Time: August 16, 1979 11:16 PM By: Redell Action: Minor cleanup, incl DriveObject not machine dependent
Time: February 1, 1980 9:06 AM By: McJones Action: Drive = > Schedule, Track = > Cylinder

-- 1) Is SetScanLineWakeup really necessary? Is it implementable on all controllers?
-- 2) Is 16 word alignment of globalState area necessary on D0?

```
DIRECTORY
    BitBlt USING [BBTable],
    Environment USING [Base, PageCount, PageNumber];

DisplayFace: DEFINITIONS =

BEGIN
```

-- Processor-independent interface to bitmap display.

-- There is no provision for multiple displays on a single machine.

-- The display has two state variables: mode and background. The mode has three values: on, off, and disconnected; the background has two values: white and black. When the system is booted, state = disconnected, and background = white.

```
hasBuffer: READONLY BOOLEAN;
```
-- TRUE = > display implementation provides real memory for bitmap;
-- FALSE = > client provides real memory for bitmap.

```
pagesForBitmap: READONLY Environment.PageCount;
```
-- size of virtual memory required to hold full size bitmap; may be larger than implied by size of visible image

```
Connect: PROCEDURE [bitmap: Environment.PageNumber];
```
-- Display must be in 'disconnected' mode; connects display to specified bitmap and leaves it in 'off' mode. Bitmap is taken by reference (first page in virtual address space); subsequent changes to bitmap in memory will affect screen image. Bitmap is always full size (i.e. pagesForBitmap pages). The bitmap does not appear on the screen until the display is turned on (see below).

```
Disconnect: PROCEDURE;
```
-- Display must be in 'off' mode; places display in 'disconnected' mode. Turns screen black, and minimizes consumption of resources (e.g. microprocessor time, memory accesses, etc.). The virtual memory area used for the bitmap may be reused for other purposes while the display is disconnected. The real memory may be reclaimed by the party providing it (i.e. if 'buffered' is FALSE, the face client may reclaim the real memory; if 'buffered' is TRUE, the face implementation may reclaim the real memory.)

```
TurnOn: PROCEDURE;
```
-- Display must be in 'off' mode; places display in 'on' mode, causing bitmap to be displayed on screen. Bitmap must have real memory mapped to it before this operation is invoked.

```
TurnOff: PROCEDURE;
```
-- Display must be in 'on' mode; places display in 'off' mode, causing background color to be displayed on screen.

```
GetBitBltTable: PROCEDURE RETURNS [BitBlt.BBTable];

SetBackground: PROCEDURE [background: Background];
Background: TYPE = {white, black};
```
-- white = > normal (black on white) view of bitmap (0 = >white, 1 = >black);
-- black = > inverted (white on black) view of bitmap (0 = >black, 1 = >white);

```
width, height: READONLY CARDINAL [0..32767];
```
-- Dimensions of screen in pixels
```
pixelsPerInch: READONLY CARDINAL;
```
-- Size of a pixel
```
refreshRate: READONLY CARDINAL;
```
-- Number of times per second that entire screen is refreshed.
```
interlaced: READONLY BOOLEAN;
```
-- Is the refresh two-way interlaced?

```
SetScanLineWakeup: PROCEDURE [line: CARDINAL, mask: WORD];
```
-- Causes naked notify using specified interrupt mask each time the specified line is refreshed on the screen.

```
GetScanLine: PROCEDURE RETURNS [line: CARDINAL];
```
-- Returns scan line currently being refreshed on the screen.

```
hasBorder: READONLY BOOLEAN;
```
-- Is border pattern implemented?

```
SetBorderPattern: PROCEDURE [oddPairs, evenPairs: [0..377B]];
```
-- Defines bit pattern shown in visible screen area (if any) outside bitmap. The bit pattern for an individual scan line is defined by displaying a single byte repeatedly along the entire scan line. The same pattern is shown on alternating pairs of lines. (Byte for even pairs shown on lines -4, -3, 0, 1, 4, 5, etc. Byte for odd pairs shown on lines -2, -1, 2, 3, 6, 7, etc.)

```
SetCursorPattern: PROCEDURE [cusorPtr: CursorPtr];
```
-- Sets the cursor bitmap to the indicated pattern by value; subsequent changes to the indicated cursor in memory will not affect screen image, hence SetCursorPattern must be called each time the cursor pattern is changed.
```
CursorPtr: TYPE = LONG POINTER TO Cursor;
Cursor: TYPE = ARRAY [0..16) OF WORD;

cursorPosition: READONLY LONG POINTER TO Point;

Point: TYPE = RECORD [x, y: INTEGER];
```
-- x and y, measured from upper-left corner of screen.

-- General

```
globalStateSize: READONLY CARDINAL;

GlobalStatePtr: TYPE = Environment.Base RELATIVE POINTER;

InitializeCleanup: PROCEDURE;
```

Initialize: PROCEDURE [globalState: GlobalStatePtr, wakeVF: WORD]; -- *This procedure initializes the implementation: 'globalState' is a client supplied block of* ***globalStateSize*** *words, 16-word aligned in first 64K of address space. Lifetime: permanently allocated and passed in on call to* ***Initialize****; 'wakeVF' is the wakeup mask for vertical-field interrupts (i.e. refresh of scan lines 0 and 1).*

END.

LOG

Time: February 6, 1980 2:22 PM By: Redell Action: Created file
Time: February 7, 1980 6:30 PM By: Gobbel Action: Made ***width*** *and* ***height*** *be subranges to avoid signed/unsigned problems*
Time: July 19, 1980 2:11 PM By: McJones Action: Added cursorPosition and (temporarily) mousePosition
Time: July 29, 1980 10:00 AM By: McJones Action: Added hasBorder; removed SetCursorPosition; split off MouseFace
Time: July 30, 1980 5:43 PM By: McJones Action: Added pagesForBitmap, GetBitBltTable; buffered = > hasBuffer

```
-- File: DriverDefs.Mesa,
--  Last Edit: HGM  September 14, 1980  1:31 AM
--  Last Edit: BLyon  August 1, 1980  3:37 PM
--  Last Edit: Garlick  October 10, 1980  4:51 PM

DIRECTORY
  StatsDefs USING [StatCounterIndex],
  BufferDefs USING [
    BufferType, Buffer, BufferPool, OisBuffer, PupBuffer, RppBuffer, Queue],
  PupTypes USING [PupHostID, PupErrorCode],
  DriverTypes USING [DeviceType],
  SpecialSystem USING [HostNumber, NetworkNumber];

DriverDefs: DEFINITIONS =
BEGIN OPEN BufferDefs;

-- Compile Time Switches
doStats: BOOLEAN = TRUE;  -- TRUE for FatPup, FALSE for TinyPup
doErrors: BOOLEAN = TRUE;
doDebug: BOOLEAN = doStats;
doSee: BOOLEAN = TRUE;
doCheck: BOOLEAN = TRUE;
doStorms: BOOLEAN = doStats;
doShow: BOOLEAN = doStats;

-- Interface from device drivers to Dispatcher and QueuePackage
PutOnGlobalInputQueue: PROCEDURE [Buffer];
PutOnGlobalDoneQueue: PROCEDURE [Buffer];
GetInputBuffer: PROCEDURE RETURNS [Buffer];  -- b.length (words) includes encap
ReturnFreeBuffer: PROCEDURE [Buffer];

-- These will return the last buffer.  They never wait.  Be careful.  NIL if its empty.
MaybeGetFreePupBuffer: PROCEDURE RETURNS [PupBuffer];
MaybeGetFreeRppBuffer: PROCEDURE RETURNS [RppBuffer];
MaybeGetFreeOisBuffer: PROCEDURE RETURNS [OisBuffer];
MaybeGetFreeBuffer: PROCEDURE RETURNS [Buffer];


-- Interface to Routers
Router: TYPE = LONG POINTER TO RouterObject;
RouterObject: TYPE = RECORD [
  input: PROCEDURE [Buffer],
  broadcast: PROCEDURE [Buffer],
  addNetwork: PROCEDURE [Network],
  removeNetwork: PROCEDURE [Network],
  stateChanged: PROCEDURE [Network] ];


-- Interface to Device Drivers
Network: TYPE = LONG POINTER TO NetworkObject;
NetworkObject: TYPE = RECORD [
  next: Network,
  decapsulateBuffer: PROCEDURE [Buffer] RETURNS [BufferDefs.BufferType],
  encapsulatePup: PROCEDURE [PupBuffer, PupTypes.PupHostID],
  encapsulateRpp: PROCEDURE [RppBuffer, PupTypes.PupHostID],
  encapsulateOis: PROCEDURE [OisBuffer, SpecialSystem.HostNumber],
  sendBuffer: PROCEDURE [Buffer],
  forwardBuffer: PROCEDURE [Buffer] RETURNS [PupTypes.PupErrorCode],
  activateDriver: PROCEDURE,
  deactivateDriver: PROCEDURE,
  deleteDriver: PROCEDURE,
  interrupt: PROCEDURE,  -- to lock things in memory
  index: [0..400B),  -- assigned by AddDeviceToChain (starts at 1)
  device: DriverTypes.DeviceType,  -- 4 bits
  alive: BOOLEAN,
  speed: CARDINAL,  -- KiloBits/Sec
  buffers: [0..256),
  spare: [0..256),
  netNumber: SpecialSystem.NetworkNumber,
  hostNumber: CARDINAL,  -- This is the Pup host number
  pupStats: PROCEDURE [PupBuffer, Network] RETURNS [BOOLEAN],
  stats: LONG POINTER ];  -- and debugging

-- ON/OFF
CommPackageGo: PROCEDURE;
CommPackageOff: PROCEDURE;
GetUseCount: PROCEDURE RETURNS [CARDINAL];
SetPupRouter, SetOisRouter: PROCEDURE [Router];  -- NIL to Deactivate
```

```
GetPupRouter, GetOisRouter: PROCEDURE RETURNS [Router];
StateChanged: PROCEDURE [Network];

-- Device Things
AddDeviceToChain: PROCEDURE [network: Network, iocbSize: CARDINAL];
RemoveDeviceFromChain: PROCEDURE [Network];
GetDeviceChain: PROCEDURE RETURNS [Network];
SmashDeviceChain: PROCEDURE; -- only used by LoopBackPlug (to forget Ethernet)

-- Setting up individual device drivers
CreateDefaultEthernetOneDrivers: PROCEDURE RETURNS [BOOLEAN];
CreateDefaultEthernetDrivers: PROCEDURE RETURNS [BOOLEAN];
CreateLoopBackDriver: PROCEDURE RETURNS [BOOLEAN];

CreateEthernetDriver: PROCEDURE [
  netNumber: CARDINAL, deviceNumber: [0..3)] RETURNS [BOOLEAN];
CreateChainedEthernetDriver: PROCEDURE [
  netNumber: CARDINAL, deviceNumber: [0..3)] RETURNS [BOOLEAN];
CreateEIADriver: PROCEDURE [host, net, lines: CARDINAL] RETURNS [BOOLEAN];
CreateCommProcDriver: PROCEDURE [host, net, lines: CARDINAL] RETURNS [BOOLEAN];
AssignPassword: PROCEDURE [line: CARDINAL, password: STRING];
CreatePacketRadioDriver: PROCEDURE [host, net: CARDINAL] RETURNS [BOOLEAN];
AdjustLengthOfEthernetInputQueue: PROCEDURE [CARDINAL];
AdjustLengthOfEthernetIIInputQueue: PROCEDURE [CARDINAL];

-- Stats linkage
EthernetStats: PROCEDURE [PupBuffer, Network] RETURNS [BOOLEAN];
SlaStats: PROCEDURE [PupBuffer, Network] RETURNS [BOOLEAN];
PacketRadioStats: PROCEDURE [PupBuffer, Network] RETURNS [BOOLEAN];

-- Internal routines
FreeQueueMake: PROCEDURE [extra: CARDINAL];
FreeQueueDestroy: PROCEDURE;
DispatcherOn: PROCEDURE;
DispatcherOff: PROCEDURE;


-- Larry and Yogen
DriverXmitStatus: TYPE = {
   pending, goodCompletion, aborted, noRouteToNetwork, hardwareProblem,
   invalidDestAddr, crateTooSmall,
  -- the following apply to circuit-like media only and mostly to the first packet sent
    noAnswerOrBusy,  -- auto-dial case only
    noTranslationForDestination,  -- no phone number for this destination
    circuitInUse, -- being used to talk to another destination
    circuitNotReady,  -- dial the phone or connect modems (non-auto-dial case)
    noDialingHardware,
    dialerHardwareProblem
};

-- General catchall for things that shouldn't happen
Glitch: PROCEDURE [ERROR];


-- interface to SystemBufferPool
GetWordsPerIocb: PROCEDURE RETURNS [CARDINAL];
SetWordsPerIocb: PROCEDURE [CARDINAL];
GetSystemBufferPool: PROCEDURE RETURNS[BufferDefs.BufferPool];


-- For getting things STARTed
Boss, DispatcherImpl, QueueImpl, SystemBufferPoolImpl, BufferPoolImpl: PROGRAM;

-- Beware MakeImage does not preserve debugPointer.

GetGiantVector: PROCEDURE RETURNS [POINTER TO GiantVector];
GiantVector: TYPE = RECORD [
  firstNetwork: Network,
  firstBuffer: Buffer,
  bufferPoolSize: CARDINAL,
  wordsPerBuffer: CARDINAL,
  pupRoutingTable: DESCRIPTOR FOR ARRAY OF UNSPECIFIED,
  firstPupSocket: LONG POINTER,
  freeQueue, globalInputQueue, globalOutputQueue: Queue,
  -- hacks until we get more drivers
  ethernetOutputQueue: Queue,
  currentInputBuffer, nextInputBuffer, currentOutputBuffer: POINTER TO Buffer,
  statCounters: LONG POINTER TO ARRAY StatsDefs.StatCounterIndex OF LONG CARDINAL,
```

```
  statStrings: LONG POINTER TO ARRAY StatsDefs.StatCounterIndex OF STRING,
  slaThings: LONG POINTER,
  prThings: LONG POINTER,
  spare: LONG POINTER ];

END.
```

LOG
August 1, 1980 3:38 PM by BLyon, Action: modified netNumber in NetworkObject.
August 6, 1980 10:17 AM By: Garlick Action: Added several DriverXmitStatus's for circuit-oriented network errors.
August 1, 1980 3:38 PM by HGM, Action: added buffers, stateChanged, and such.
Time: October 10, 1980 4:50 PM By: Garlick Action: Added DriverXmitStatus noAnswerOrBusy.

```
DriverStartChain: DEFINITIONS =

BEGIN

-- NOTE: When AR 3536 is fixed, the modules HeadStartChain, DriverStartChain, and StoreDriverStartChain may be
replace with single module StartChain.

-- Mechanism used to implement starting extensible set of modules (e.g. heads, drivers, test programs)

-- We refer to a member of the set of modules as a "link"; we refer to the module wishing to start the set as the "master link".
-- Each link should export Start with a body consisting simply of a call on Start imported through a second instance of StartChain (e.g.
     IMPORTS RemainingHeads: StartChain).
-- The master link should export Start with a body which just returns and should also contain a call on Start imported through a second
     instance of StartChain named EntireStartChain.
-- The machinery is completed with a configuration which sets up a chain of StartChain imports-exports from EntireStartChain, through
     each of the links, and back to the master link.

Start: PROCEDURE;
     -- Start the (rest of the) chain

END.
```

LOG

Time: January 31, 1980 11:09 AM By: McJones Action: Create file

```
-- File: DriverTypes.Mesa,  Last Edit: HGM  August 19, 1980  10:01 AM

DIRECTORY
  SpecialSystem USING [HostNumber, ProcessorID];

DriverTypes: DEFINITIONS =
BEGIN

Byte: TYPE = [0..377B];

DeviceType: TYPE = {
  unknown, local, xwire, ethernet, sla, arpanet, packetradio, phonenet, spare};

HostNumber: TYPE = SpecialSystem.HostNumber;


-- ENCAPSULATION
-- This lives here because it is common to Pup and Ois, and needs to be compiled before BufferDefs

-- An instance of this record is the last thing in a Buffer before the real data block.  The hardware
transfers bits to/from it and then keeps going into the body of the packet.  Note that the spare words are
at the beginning of the record, not the end.  That way the driver can easily skip over them by just
starting at the right place.


-- BEWARE OF QUAD WORD ALIGNMENT HICKUPS    REALLY!!!!!!
Encapsulation: TYPE = MACHINE DEPENDENT RECORD [
  SELECT OVERLAID DeviceType FROM
    local => [
      localSpare1, localSpare2, localSpare3, localSpare4, localSpare5: WORD,
      localHost: CARDINAL,
      localType: LocalPacketType ],
    xwire => [
        -- bits on the wire start here
        xwireDest: HostNumber,
        xwireSource: SpecialSystem.ProcessorID,
        xwireType: XWirePacketType ],
    ethernet => [
      etherSpare1, etherSpare2, etherSpare3, etherSpare4: WORD,
      etherSpare5: [0..77777B],
      translationWorked: BOOLEAN,
      -- Bits on the ether start here
      etherDest, etherSource: Byte,
      ethernetType: EthernetPacketType ],
    sla => [
      slaSpare1: WORD,
      slaSpare2: WORD,
      slaSpare3: [0..37777B],
      slaHi: BOOLEAN,
      slaBroadcast: BOOLEAN,
      slaTimeQueued: CARDINAL,
      slaSourceLine, slaDestHost: WORD,  -- BEWARE: these don't get sent
      -- Bits on the line start here
      slaType: SlaPacketType ],
    arpanet => [
      arpaSpare1, arpaSpare2, arpaSpare3, arpaSpare4, arpaSpare5: WORD,
      arpanetControl: Byte,  -- 0 for "Regular message"
      arpanetHost: Byte,  -- source if sending, dest if receiving
      arpanetLink: ArpanetLink,
      apranetZero: Byte ],
    packetradio => [  -- Larry Stewart
      prSpare1: WORD,
      prLength: WORD,
      timer: CARDINAL,
      prSequence: WORD,
      prBroadcast: BOOLEAN,
      numFrags, numFragsTrans, numFragsRcvd: [0..3],
      first, second, third: BOOLEAN,
      transCount: [0..77B],
      prAddress: WORD,
      prType: PRPacketType,
      prSpare: Byte ],
    phonenet => [  -- Larry Garlick
      framing0, framing1, framing2, framing3, framing4, framing5: Byte,
      recognition: Byte,  -- 0 for auto recognition of OISCP vs SDLC/HDLC
      pnType: PhonePacketType,
      pnSrcID: SpecialSystem.ProcessorID ],
    spare => [
```

```
      spare1, spare2, spare3, spare4, spare5, spare6, spare7: WORD ],
    ENDCASE ];

-- Many places will fall apart if SIZE[Encapsulation]#6.
encapsulationTrap: [7..7] = SIZE[Encapsulation];
```

```
    ethernetBroadcastHost: Byte = 0;
    ethernetPeekHost: Byte = 376B;  -- DMT error info
    ethernetBootLoaderHost: Byte = 377B;

    -- offsets are in words
    localEncapsulationOffset: CARDINAL = 5;
    localEncapsulationBytes: CARDINAL = 4;
    ethernetEncapsulationOffset: CARDINAL = 5;
    ethernetEncapsulationBytes: CARDINAL = 4;
    slaEncapsulationOffset: CARDINAL = 6;
    slaEncapsulationBytes: CARDINAL = 2;
    prEncapsulationOffset: CARDINAL = 6;
    prEncapsulationBytes: CARDINAL = 2;
    prEncapsulationWords: CARDINAL = prEncapsulationBytes/2;
    phoneEncapsulationOffset: CARDINAL = 3;
    phoneEncapsulationBytes: CARDINAL = 8;

    LocalPacketType: TYPE = MACHINE DEPENDENT {
      pupLocalPacket(1000B),
      oisLocalPacket(3000B),
      last(LAST[WORD])};

    XWirePacketType: TYPE = MACHINE DEPENDENT {
      echoMe(700B),
      echoed(701B),
      pup(1000B),
      ois(3000B),
      last(LAST[WORD]) };

    EthernetPacketType: TYPE = MACHINE DEPENDENT {
      peekData(402B),
      breathOfLife(602B),
      echoMeEthernetPacket(700B),
      echoedEthernetPacket(701B),
      pupEthernetPacket(1000B),
      oisEthernetPacket(3000B),
      translationPacket(3001B),
      last(LAST[WORD]) };

    SlaPacketType: TYPE = MACHINE DEPENDENT {
      pupSlaPacket(1000B),
      routingSlaPacket(1001B),
      oisSlaPacket(3000B),
      last(LAST[WORD]) };

    ArpanetLink: TYPE = MACHINE DEPENDENT {
      pupArpanetLink(152),
      last(LAST[Byte]) };

    PRPacketType: TYPE = MACHINE DEPENDENT {
      pupPRPacketType(1),
      oisPRPacketType(2),
      imAlivePRPacketType(3),
      bcastPupPRPacketType(4),
      last(LAST[Byte]) };

    PhonePacketType: TYPE = MACHINE DEPENDENT {
      pupPhonePacket(100B),
      oisPhonePacket(300B),
      turnAroundPhonePacket(301B),
      turnAroundMTSPhonePacket(302B),
      last(LAST[Byte]) };


    -- DEBUGGING

    Seal: TYPE = RECORD [WORD];
      unsealed: Seal = [0];
      queueSeal: Seal = [123001B];
      bufferSeal: Seal = [123002B];
      bufferPoolSeal: Seal = [123003B];

    END.
```

```
-- Function: The definitions module for OISCP Echo Protocol.

Echo: DEFINITIONS =
BEGIN

-- definitions

-- format of echo packets
echoRequest: CARDINAL = 1;
echoResponse: CARDINAL = 2;


-- interface

CreateServer: PROCEDURE;
DeleteServer: PROCEDURE;

END. -- Echo
```

LOG

Time: January 31, 1980 12:28 PM By: Dalal Action: created file.

```
Environment: DEFINITIONS =

BEGIN

    -- Fundamental properties of the OIS processor
bitsPerWord: CARDINAL = 16;
logBitsPerWord: CARDINAL = 4; -- logarithm of bitsPerWord

bitsPerByte, bitsPerCharacter: CARDINAL = 8;
logBitsPerByte, logBitsPerChar: CARDINAL = 3; -- logarithm of bitsPerByte

bytesPerWord, charsPerWord: CARDINAL = bitsPerWord/bitsPerByte;
logBytesPerWord, logCharsPerWord: CARDINAL = logBitsPerWord-logBitsPerByte; -- logarithm of bytesPerWord

wordsPerPage: CARDINAL = 256;
bytesPerPage, charsPerPage: CARDINAL = wordsPerPage*bytesPerWord;
logWordsPerPage: CARDINAL = 8; -- logarithm of wordsPerPage
logBytesPerPage, logCharsPerPage: CARDINAL = logWordsPerPage + logBytesPerWord; -- logarithm of bytesPerPage

    -- The following is the base pointer to the first 64K of virtual memory
first64K: Base = LOOPHOLE[LONG [0]];

maxINTEGER: INTEGER = LAST[INTEGER]; -- 32767
minINTEGER: INTEGER = FIRST[INTEGER]; -- -32768
maxCARDINAL: CARDINAL = LAST[CARDINAL]; -- 177777B
maxLONGINTEGER: LONG INTEGER = LAST[LONG INTEGER]; -- 2147483647
minLONGINTEGER: LONG INTEGER = FIRST[LONG INTEGER]; -- -2147483648
maxLONGCARDINAL: LONG CARDINAL = LAST[LONG CARDINAL]; -- 4294967295

Byte: TYPE = [0..255];
Word: TYPE = [0..65535];
Long, LongNumber: TYPE = MACHINE DEPENDENT RECORD [SELECT OVERLAID * FROM
        lc => [lc: LONG CARDINAL],
        li => [li: LONG INTEGER],
        lp => [lp: LONG POINTER],
        lu => [lu: LONG UNSPECIFIED],
        num => [lowbits, highbits: CARDINAL],
        any => [low, high: UNSPECIFIED],
        ENDCASE];

    -- Common types used throughout Pilot
Base: TYPE = LONG BASE POINTER;

BitAddress: TYPE = MACHINE DEPENDENT RECORD[
    word: LONG POINTER,
    reserved: [0..LAST[WORD]/bitsPerWord) ← 0,
    bit: [0..bitsPerWord)];

Block: TYPE = RECORD [ -- descriptor for arbitrary sequence of bytes
        blockPointer: LONG POINTER,
        startIndex, stopIndexPlusOne: CARDINAL];
nullBlock: Block = [NIL, 0, 0];

maxPagesInVM: CARDINAL = 65535; -- Note that this is one less than the number of VM pages provided by the hardware; the highest numbered VM
        page is pre-empted for system purposes
maxPagesInMDS: CARDINAL = 256;
PageNumber: TYPE = [0..maxPagesInVM];
PageOffset: TYPE = [0..maxPagesInVM];
PageCount: TYPE = [0..maxPagesInVM];

END.
```

LOG

Time: April 26, 1978 4:34 PM By: McJones Action: Created file
Time: May 3, 1978 1:46 PM By: Lauer Action: Merged contents of AltoDefs into file; changed name of file from "Processor".
Time: June 21, 1978 10:03 AM By: Lauer Action: Added maxINTEGER, minINTEGER, maxCARDINAL, maxLONGINTEGER, minLONGINTEGER
Time: July 21, 1978 11:31 AM By: Lauer Action: Moved type 'Block' from Stream to Environment
Time: August 14, 1978 10:26 AM By: Horsley Action: Used FIRST and LAST for number bounds
Time: March 7, 1979 10:31 AM By: McJones Action: Added Long; increased upper bound for PageNumber, PageOffset

Time: July 16, 1979 6:26 PM By: Knutsen Action: Added first64K, Base.
Time: January 25, 1980 5:46 PM By: Forrest AR1607: nullBlock added, AR1740: logBitsPerByte, logBitsPerWord added; change any branch of LONG NUMBER to use UNSPECIFIEDs.
Time: April 17, 1980 12:08 AM By: Forrest Action: added BitAddress.

```
DIRECTORY
    Environment USING [Base],
    SpecialSystem USING [ProcessorID];

EthernetFace: DEFINITIONS =

BEGIN
```

-- Note: This in an Ethernet 2 face. See EthernetOneFace for the interface to the old Ethernet. We may also have to do something about multicasting one of these days.

-- PROCEDURES

GetNextDevice: PROCEDURE [DeviceHandle] RETURNS [DeviceHandle];
-- Return handle of next device in physical order (starts and ends with EthernetFace.nullDeviceHandle*).*

AddCleanup: PROCEDURE [DeviceHandle];
RemoveCleanup: PROCEDURE [DeviceHandle];
-- Add/delete the device cleanup procedure which is used to turn off an Ethernet controller around world swaps (see DeviceCleanup.mesa). For now, RemoveCleanup *doesn't do anything, but the (leftover) cleanup proc is careful to avoid doing anything nasty. (Extra calls to* AddCleanup *are currently ignored.)*

TurnOn: PROCEDURE [device: DeviceHandle, host: SpecialSystem.ProcessorID, inInterrupt, outInterrupt: WORD, globalState: GlobalStatePtr];
TurnOff: PROCEDURE [device: DeviceHandle];
-- Turn on/off the device associated with this DeviceHandle. Calling TurnOn *when the device is already on may forget any operations currently in progress. Extra calls to* TurnOff *are harmless.*

QueueOutput: PROCEDURE [device: DeviceHandle, buffer: LONG POINTER, length: CARDINAL, cb: ControlBlock];
QueueInput: PROCEDURE [device: DeviceHandle, buffer: LONG POINTER, length: CARDINAL, cb: ControlBlock];
-- Queue the buffer for transmit or receive. A (different) control block must be passed in with each call for the private use of the head. cb *should be a pointer to a block of* EthernetFace.controlBlockSize *words.* ***It must be allocated within the first 64K and must be quad word aligned. The buffer must have 11 in the low 2 bits (ie it starts one word before a quad word aligned buffer) and less than 32K words long.*** *When* GetStatus[cb] *returns anything other than* pending, *both the control block and the data buffer are available for reuse.*

GetStatus: PROCEDURE [cb: ControlBlock] RETURNS [status: Status];
-- Returns the status of the packet associated with this control block. See the comments on Status *below.*

GetRetries: PROCEDURE [cb: ControlBlock] RETURNS [CARDINAL];
-- Returns the number of times the packet was retransmitted because of collisions.

GetPacketLength: PROCEDURE [cb: ControlBlock] RETURNS [CARDINAL];
-- Returns the length in words of a recently arrived packet.

GetPacketsMissed: PROCEDURE [DeviceHandle] RETURNS [CARDINAL];
-- Returns the number of input packets missed because a buffer was not ready.

-- TYPES

```
DeviceHandle: TYPE[1];

GlobalStatePtr: TYPE = Environment.Base RELATIVE POINTER;

Status: TYPE = {pending, ok,
 overrun, -- Input data arrived when FIFO was full
 underrun, -- Output FIFO went empty before end of packet was set
 packetTooLong, -- Input packet didn't fit into buffer
 tooManyCollisions, -- More than 16 attempts to transmit the same packet
 crc,
 crcAndBadAlignment,
 badAlignmentButOkCrc,
 otherError};
```

-- Note: The error conditions are an open ended set to allow a smart driver to collect statistics. Thus new error conditions may be added to allow more sophisticated data collection, so a driver should not get upset if it encounters a strange error status that it does not understand. Dumb drivers should only check for pending *or* ok *and consider everything else as an error.*

```
ControlBlock: TYPE = LONG POINTER TO ControlBlockRecord;
ControlBlockRecord: TYPE;
```

-- EXPORTED VARIABLES

globalStateSize: READONLY CARDINAL;
controlBlockSize: READONLY CARDINAL;
nullDeviceHandle: READONLY DeviceHandle;
hearSelf: READONLY BOOLEAN;

-- CONSTANTS

END.

LOG
Time: September 5, 1980 12:16 AM By: HGM Action: Create from EthernetOneFace

```
DIRECTORY
    Environment USING [Base, Byte];

EthernetOneFace: DEFINITIONS =

BEGIN

-- PROCEDURES

GetNextDevice: PROCEDURE [DeviceHandle] RETURNS [DeviceHandle];
    -- Return handle of next device in physical order (starts and ends with EtherNetFace.nullDeviceHandle).

GetEthernet1Address: PROCEDURE [DeviceHandle] RETURNS [net, host: Environment.Byte];
    -- Returns the net + host numbers from the switches (or Prom, or ...).

AddCleanup: PROCEDURE [DeviceHandle];
RemoveCleanup: PROCEDURE [DeviceHandle];
    -- Add/delete the device cleanup procedure which is used to turn off an Ethernet controller around world swaps (see
        DeviceCleanup.mesa). For now, RemoveCleanup doesn't do anything, but the (leftover) cleanup proc is careful to avoid doing
        anything nasty. (Extra calls to AddCleanup are currently ignored.)

TurnOn: PROCEDURE [device: DeviceHandle, host: Environment.Byte, inInterrupt, outInterrupt: WORD, globalState: GlobalStatePtr];
TurnOff: PROCEDURE [device: DeviceHandle];
    -- Turn on/off the device associated with this DeviceHandle. Calling TurnOn when the device is already on may forget any
        operations currently in progress. Extra calls to TurnOff are harmless.

QueueOutput: PROCEDURE [device: DeviceHandle, buffer: LONG POINTER, length: CARDINAL, cb: ControlBlock];
QueueInput: PROCEDURE [device: DeviceHandle, buffer: LONG POINTER, length: CARDINAL, cb: ControlBlock];
    -- Queue the buffer for transmit or receive. A (different) control block must be passed in with each call for the private use of the
        head. cb should be a pointer to a block of EtherNetFace.controlBlockSize words. It must be allocated within the first
        64K and must be quad word aligned. The data buffer must be quad word aligned and less than 32K words long.
        When GetStatus[cb] returns anything other than pending, both the control block and the data buffer are available for reuse.

GetStatus: PROCEDURE [cb: ControlBlock] RETURNS [status: Status];
    -- Returns the status of the packet associated with this control block. See the comments on Status below.

GetRetries: PROCEDURE [cb: ControlBlock] RETURNS [CARDINAL];
    -- Returns the number of times the packet was retransmitted because of collisions.

GetPacketLength: PROCEDURE [cb: ControlBlock] RETURNS [CARDINAL];
    -- Returns the length in words of a recently arrived packet.

GetPacketsMissed: PROCEDURE [DeviceHandle] RETURNS [CARDINAL];
    -- Returns the number of input packets missed because a buffer was not ready.

-- TYPES

DeviceHandle: TYPE[1];

GlobalStatePtr: TYPE = Environment.Base RELATIVE POINTER;

Status: TYPE = {pending, ok,
 overrun, -- Input data arrived when FIFO was full
 underrun, -- Output FIFO went empty before end of packet was set
 packetTooLong, -- Input packet didn't fit into buffer
 tooManyCollisions, -- More than 16 attempts to transmit the same packet
 crc,
 crcAndBadAlignment,
 badAlignmentButOkCrc,
 otherError};
    -- Note: The error conditions are an open ended set to allow a smart driver to collect statistics. Thus new error conditions may be
        added to allow more sophisticated data collection, so a driver should not get upset if it encounters a strange error status that it
        does not understand. Dumb drivers should only check for pending or ok and consider everything else as an error.

ControlBlock: TYPE = LONG POINTER TO ControlBlockRecord;
ControlBlockRecord: TYPE;

-- EXPORTED VARIABLES

globalStateSize: READONLY CARDINAL;
controlBlockSize: READONLY CARDINAL;
nullDeviceHandle: READONLY DeviceHandle;
```

hearSelf: READONLY BOOLEAN;

-- CONSTANTS

END.

LOG

Time: February 4, 1980 5:48 PM By: McJones Action: Create file
Time: February 6, 1980 12:54 PM By: Dawson Action: Add interface items
Time: April 18, 1980 5:51 PM By: Murray Action: Giant Cleanup
Time: August 20, 1980 2:05 PM By: BLyon Action: renames EthernetFace to EthernetOneFace and added hearSelf
Time: September 5, 1980 12:11 AM By: HGM Action: fix typos, add ControlBlockRecord

```
DIRECTORY
    Inline USING [BITAND],
    System USING [FileID, nullID, VolumeID],
    Transaction USING [Handle, nullHandle];

File: DEFINITIONS IMPORTS Inline, Transaction =
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); -- simulate 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
    RETURN[Capability[originalFC.fID, LOOPHOLE[Inline.BITAND[originalFC.permissions, maxPermissions]]]]
    END;

Type: TYPE = RECORD [CARDINAL];

    -- Addressing within a file
maxPagesPerFile: LONG CARDINAL = 8388607; -- = 2^23 - 1
PageNumber: TYPE = LONG CARDINAL; -- simulate TYPE = [0..maxPagesPerFile);
    firstPageNumber: PageNumber = 0;
    lastPageNumber: PageNumber = maxPagesPerFile-1;
PageCount: TYPE = LONG CARDINAL; -- simulate TYPE = [0..maxPagesPerFile];
    firstPageCount: PageCount = 0;
    lastPageCount: PageCount = maxPagesPerFile;

    -- Creating, deleting, and moving files
Create: PROCEDURE [volume: System.VolumeID, initialSize: PageCount, type: Type, transaction: Transaction.Handle ←
        Transaction.nullHandle] RETURNS [file: Capability];
Delete: PROCEDURE [file: Capability, transaction: Transaction.Handle ← Transaction.nullHandle];
Move: PROCEDURE [file: Capability, volume: System.VolumeID, transaction: Transaction.Handle ← Transaction.nullHandle];

    -- Immutable files
MakeImmutable: PROCEDURE [file: Capability, transaction: Transaction.Handle ← Transaction.nullHandle];
ReplicateImmutable: PROCEDURE [file: Capability, volume: System.VolumeID, transaction: Transaction.Handle ← Transaction.nullHandle];
DeleteImmutable: PROCEDURE [file: Capability, volume: System.VolumeID, transaction: Transaction.Handle ← Transaction.nullHandle];

    -- Attributes of files
GetSize: PROCEDURE [file: Capability, transaction: Transaction.Handle ← Transaction.nullHandle] RETURNS [size: PageCount];
SetSize: PROCEDURE [file: Capability, size: PageCount, transaction: Transaction.Handle ← Transaction.nullHandle];
GetAttributes: PROCEDURE [file: Capability, transaction: Transaction.Handle ← Transaction.nullHandle] RETURNS [type: Type, immutable,
        temporary: BOOLEAN, volume: System.VolumeID];
MakePermanent: PROCEDURE [file: Capability, transaction: Transaction.Handle ← Transaction.nullHandle];

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

    -- Signals and errors
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
Time: March 31, 1980 4:26 PM By: Gobbel Action: Added transaction handles
Time: April 15, 1980 3:59 PM By: Gobbel Action: Added nullTransactionHandle.
Time: June 16, 1980 2:17 PM By: McJones Action: Cleanup

```
DIRECTORY
    File: FROM "File" USING [ID, PageNumber],
    FileInternal: FROM "FileInternal" USING [Descriptor, FilePtr, PageGroup],
    Volume: FROM "Volume" USING [ID];

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];

FlushFilesOnVolume: PROCEDURE [lvID: Volume.ID, pin: PinnedAction];

PinnedAction: TYPE = {keep, flush, error};

END.
```

LOG

Time: April 15, 1978 3:46 PM By: Redell Action: Created file
Time: May 13, 1980 3:49 PM By: Luniewski Action: Added FlushFilesOnVolume.

```
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 = LONG POINTER TO Descriptor;

LocalFilePtr: TYPE = LONG POINTER TO local Descriptor;

RemoteFilePtr: TYPE = LONG POINTER TO remote 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.
Time: May 14, 1980 2:58 PM By: Luniewski Action: Made FilePtr's LONG POINTER's. Added LocalFilePtr and RemoteFilePtr.

```
-- NOTE: This defs module should go away and its clients should use PilotDisk directly.

DIRECTORY
    Environment USING [PageOffset],
    File USING [ID, nullCapability, PageNumber, Type],
    PilotDisk USING [GetLabelFilePage, GetLabelType, nullLabel, Label, LabelCheckSum, SetLabelFilePage,
        SetLabelType];

FilePageLabel: DEFINITIONS IMPORTS PilotDisk SHARES File =

BEGIN

Label: TYPE = PilotDisk.Label;

nullLabel: Label = PilotDisk.nullLabel;

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

GetFilePage: PROCEDURE [label: LONG POINTER TO Label] RETURNS [File.PageNumber] =
    INLINE BEGIN
    RETURN[PilotDisk.GetLabelFilePage[label]]
    END;

SetFilePage: PROCEDURE [label: LONG POINTER TO Label, fpn: File.PageNumber] =
    INLINE BEGIN
    PilotDisk.SetLabelFilePage[label, fpn];
    END;

GetType: PROCEDURE [label: LONG POINTER TO Label] RETURNS [File.Type] =
    INLINE BEGIN RETURN [PilotDisk.GetLabelType[label]] END;

SetType: PROCEDURE [label: LONG POINTER TO Label, type: File.Type] =
    INLINE BEGIN PilotDisk.SetLabelType[label, type]; END;

LabelCheckSum: PROCEDURE [label: LONG POINTER TO Label, offset: Environment.PageOffset]
        RETURNS [checkSum: CARDINAL] = INLINE
    BEGIN
    RETURN[PilotDisk.LabelCheckSum[label, offset]]
    END;

END.
```

LOG

Time: July 21, 1979 4:32 PM By: Redell Action: Create file
Time: June 16, 1980 5:50 PM By: McJones Action: Redefine in terms of PilotDisk
Time: August 16, 1980 6:08 PM By: Fay Action: Delete tBootFile.

```
DIRECTORY
    Environment: FROM "Environment" USING [PageCount, PageNumber],
    File: FROM "File" USING [Capability, PageNumber],
    FileInternal: FROM "FileInternal" USING [Operation],
    VM: FROM "VM" USING [Interval];

FilePageTransfer: DEFINITIONS =

BEGIN

Request: TYPE = RECORD[
    file: File.Capability,
    filePage: File.PageNumber,
    memoryPage: Environment.PageNumber,
    count: Environment.PageCount,
    promise: BOOLEAN←FALSE, -- should caller of Initiate then call MStore.Promise[countInitiated]?
    opSpecific: SELECT operation: Operation FROM
        read => [priorityPage: File.PageNumber],
        write => NULL,
        ENDCASE];

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

Initiate: PROCEDURE [req: Request] RETURNS [countInitiated: Environment.PageCount];
    -- starts a I/O operation on a block of pages (and tells you how many pages were initially started transferring).

Wait: PROCEDURE RETURNS [VM.Interval];
    -- waits until the completion of any single page transfer of any previously Initiate'ed operation.

END.
```

LOG

Time: March 10, 1978 3:09 PM	*By: Redell*	*Action: Created file*
Time: September 5, 1979 3:08 PM	*By: McJones*	*AR1593: Added countInitiated result to Initiate, promise to Request*
Time: September 18, 1979 4:59 PM	*By: Knutsen*	*Action: Improved documentation.*
Time: November 26, 1979 12:24 PM	*By: Gobbel*	*Action: Wait now returns VM.Interval instead of PageNumber.*

```
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

```
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
Time: January 30, 1980 4:26 PM By: Gobbel Action: RemotePageTransferImpl deactivated

```
DIRECTORY
    DiskChannel: FROM "DiskChannel" USING [IORequestHandle],
    Environment: FROM "Environment" USING [PageNumber],
    File: FROM "File" USING [PageCount],
    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, count: File.PageCount, 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, countValid: File.PageCount];

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.
Time: November 26, 1979 12:28 PM By: Gobbel Action: Added count to DiskStart.
Time: January 9, 1980 3:51 PM By: Gobbel Action: Added countValid to LabelWait.
Time: January 29, 1980 1:19 PM By: Gobbel Action: Merge with Redell's version.

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

FileTypes: DEFINITIONS =
BEGIN

-- This file is the authoritative definition of File Types for Pilot files for Client 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;

-- These are copied from PilotFIleTYpes.mesa; probably nobody should be using them
PilotFileType:          TYPE = CARDINAL[0..256);
    tUnassigned:        File.Type = [PilotFileType[0]];

-- Allocation of subranges of File Types
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]];

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 ???*
Time: August 29, 1979 9:52 AM	*By: Forrest*	*Action: Moved PilotFileTypes to PilotFileTypes*
Time: September 4, 1979 9:35 AM	*By: Forrest*	*Action: Restored tUnassigned*

```
DIRECTORY
    PhysicalVolume: FROM "PhysicalVolume" USING [Handle];

FloppyChannel: DEFINITIONS =

BEGIN

-- Basic serial read/write access to a floppy disk drive of the SA800 family.  Floppy channel access and Pilot volume operations
    are mutually exclusive and interlocked.  Note that the serial nature of the interface dictates that only one process should
    be using it at a time.


-- TYPES

Handle: TYPE = PhysicalVolume.Handle;

Attributes: TYPE = RECORD [
    deviceType: {SA800, SA850},
        --Indicates what type of drive is connected to the controller.
    numberOfCylinders: [0..256),
        --Number of cylinders available for recording on the drive connected to the controller.
    numberOfHeads: [0..256),
        --Number of read/write heads available for recording on the drive connected to the controller.
    trackLength: CARDINAL];
        --Length in words of the buffer that the client must supply in order to format the diskette.

Buffer: TYPE = RECORD [
        address: LONG POINTER TO UNSPECIFIED,     -- must be quad-word alligned
        length: CARDINAL]; -- must be > = Context.sectorLength*operation count

Context: TYPE = RECORD [
    protect: BOOLEAN,
        --Software write protect available to the client.  The actual write protect status is a logical OR of this variable and the
        physical signal being returned from the drive.
    format: {IBM, Troy},
        --Client's selection of the format type; either an IBM compatible format (which includes STAR), or the Xerox 850 (Troy)
        format.
    density: {single, double},
        --Selection of either FM (single density) or MFM (double density) recording. This is an available selection when formatting
        a new diskette. When accessng a previously recorded diskette, the client must provide the correct setting. (Note that
        track00 on IBM format diskettes and all tracks of Troy format diskettes will be single density.)
    sectorLength: CARDINAL];
        --Length in words of the sectors on the current track. The value must come from a valid set defined as {64, 128, 256, 512}
        for IBM format and {1022} for Troy format.

DiskAddress: TYPE = MACHINE DEPENDENT RECORD [
    cylinder: CARDINAL,
        --must be IN [0..numberOfCylinders]
    head: [0..256),
        --must be IN [0..numberOfHeads]
    sector: [0..256)];
        --must be IN [1..{value depends on sectorLength}]
```

```
Status: TYPE = MACHINE DEPENDENT RECORD [
    diskChanged (0: 0..0): BOOLEAN,
        --(information) The disk drive has apparently gone from ready to not-ready (door open), or from not-ready to ready, one
        or more times since the last operation was performed.
    tbd1 (0: 1..1): BOOLEAN,
        --This status record is unassigned.
    twoSided (0: 2..2): BOOLEAN,
        --(information) The diskette currently installed is double sided.
    tbd2 (0: 3..3): BOOLEAN,
        --This status record is unassigned.
    error (0: 4..4): BOOLEAN,
        --This status record indicates an error; a specific error status is also set.
    inProgress (0: 5..5): BOOLEAN,
        --The operation is not yet complete.
    recalibrateError (0: 6..6): BOOLEAN,
        --A recalibrate failed.
    wrongSizeBuffer (0: 7..7): BOOLEAN,
        --The buffer supplied by the client was shorter than the length of the transfer requested.
    notReady (0: 8..8): BOOLEAN,
        --The drive is not ready.
    writeProtect (0: 9..9): BOOLEAN,
        --(information OR error, depending on the value of the error bit) Logical OR of the context setting of protect and the
        physical signal being returned from the drive.
    deletedData (0: 10..10): BOOLEAN,
        --(information) The ID for the sector contained a deleted data address mark.
    recordNotFound (0: 11..11): BOOLEAN,
        --The record defined by the disk address could not be found.
    crcError (0: 12..12): BOOLEAN,
        --A CRC error was encountered on either the record ID or the data field.
    track00 (0: 13..13): BOOLEAN,
        --(information) The read/write heads are currently positioned over track00, the outermost track.
    hardwareError (0: 14..14): BOOLEAN,
        --An undefined status was returned.
    goodCompletion (0: 15..15): BOOLEAN];
        --The request was completed successfully.

-- PROCEDURES

-- The following six drive-access procedures include seek, error retry, and wait for completion or error. Operations in a single call may
    not cross over from an IBM-format track 0 to the rest of the diskette; no check is made for this.

ReadSectors: PROCEDURE [handle: Handle, address: DiskAddress, buffer: Buffer, count: CARDINAL ← 1, incrementDataPtr: BOOLEAN ←
    TRUE]
    RETURNS [status: Status, countDone: CARDINAL];
    --Reads the specified number of sectors from the diskette beginning at the specified disk address.

WriteSectors: PROCEDURE [handle: Handle, address: DiskAddress, buffer: Buffer, count: CARDINAL ← 1, incrementDataPtr: BOOLEAN ←
    TRUE]
    RETURNS [status: Status, countDone: CARDINAL];
    --Writes the specified number of sectors with data address marks to the diskette beginning at the specified disk address.

WriteDeletedSectors: PROCEDURE [handle: Handle, address: DiskAddress, buffer: Buffer, count: CARDINAL ← 1, incrementDataPtr:
    BOOLEAN ← TRUE]
    RETURNS [status: Status, countDone: CARDINAL];
    --Writes the specified number of sectors with deleted data address marks to the diskette beginning at the specified disk
    address.

ReadID: PROCEDURE [handle: Handle, address: DiskAddress, buffer: Buffer]
    RETURNS [status: Status];
    --Reads to first encountered record ID from the specified disk address. The value of sector in the disk address is ignored.

Nop: PROCEDURE [handle: Handle]
    RETURNS [status: Status];
    --Does a regular drive-access operation without data transfer and returns a valid status.

Recalibrate: PROCEDURE [handle: Handle]
    RETURNS [status: Status];
    --Moves the heads to track 00, sector 00.
```

GetContext: PROCEDURE [handle: Handle]
RETURNS [context: Context];
--Returns the current settings of the context.

GetDeviceAttributes: PROCEDURE [handle: Handle]
RETURNS [attributes: Attributes];
--Returns attributes as defined above.

SetContext: PROCEDURE [handle: Handle, context: Context]
RETURNS [ok: BOOLEAN];
--Sets context as defined above and returns a BOOLEAN indicating whether the settings were accepted.

END...

LOG

Time: June 10, 1980 3:10 PM By: Jose Action: Created file.
Time: October 10, 1980 3:06 PM By: Jose Action: Add incrementDataPtr parameter to transfer procedures, default count and incrementDataPtr.

```
DIRECTORY
    DiskChannel USING [PVHandle],
    FloppyChannel USING [Status],
    SA800Face USING [Buffer, DiskAddress, Function];

FloppyChannelInternal: DEFINITIONS =

BEGIN

-- The part of the basic serial read/write access to a floppy disk drive of the SA800 family which is hidden from the client, but
    not from the Pilot floppy driver.


-- TYPES

OpBlock: TYPE = RECORD [
    device: DiskChannel.PVHandle,
            --pointer to drive object.
    function: SA800Face.Function,
            --e.g., readSector, writeSector, readID, writeDeletedSectors, nop, recovery.
    address: SA800Face.DiskAddress ← nullAddress,
            --disk address.
    buffer: SA800Face.Buffer ← nullBuffer,
            --client data address.
    incrementDataPtr: BOOLEAN ← TRUE,
            --FALSE = use same buffer for all sectors.
    count: CARDINAL ← 1];
            --number of sectors to transfer.


-- CONSTANTS

maxDrives: CARDINAL = 2;
maxSectorsPerTrack: CARDINAL = 15;
maxWordsPerSector: CARDINAL = 256; -- excludes Troy format
nullAddress: SA800Face.DiskAddress = [0, 0, 0];
nullBuffer: SA800Face.Buffer = [NIL, 0];


-- PROCEDURES

DoRequest: PROCEDURE [pOp: POINTER TO OpBlock]
        RETURNS [status: FloppyChannel.Status, countDone: CARDINAL];
        --Does the real work of the request, including communication with the Face and error handling.

END...



LOG

Time: June 17, 1980 4:18 PM      By: Jose  Action: Created file.
Time: September 5, 1980 4:07 PM      By: Jose  Action: Added BasicChangeState.
Time: September 15, 1980 2:22 PM      By: Jose  Action: Added SimpleRequest, deleted BasicChangeState.
Time: October 10, 1980 3:24 PM  By: Jose  Action: Changes for moving runs of pages from driver to head, added constants.
```

```
DIRECTORY
    Boot: FROM "Boot" USING [LVBootFiles, Location],
    Device: FROM "Device" USING [Type];

FMPrograms: DEFINITIONS =

BEGIN

MarkerPageImpl: PROGRAM;

PhysicalVolumeImpl: PROGRAM [bootFile: LONG POINTER TO disk Boot.Location, pLVBootFiles: POINTER TO Boot.LVBootFiles]
    RETURNS [debuggerDeviceType: Device.Type, debuggerDeviceOrdinal: CARDINAL]; -- see StoragePrograms.FileImpl

ScavengeImpl: PROGRAM;

VolAllocMapImpl: PROGRAM;

VolFileMapImpl: PROGRAM;

VolumeImpl: PROGRAM [bootFile: LONG POINTER TO disk Boot.Location, pLVBootFiles: POINTER TO Boot.LVBootFiles,
        debuggerDeviceType: POINTER TO Device.Type, debuggerDeviceOrdinal: POINTER TO CARDINAL]; -- see StoragePrograms.FileImpl
        and StoragePrograms.PhysicalVolumeImpl

END.
```

LOG

Time: June 29, 1978 2:34 PM By: Purcell Action: Create file from InitializeFM
Time: August 5, 1978 11:17 PM By: Redell Action: Replace PageBufferImpl with FilePointImpl
Time: July 31, 1979 5:31 PM By: Forrest Action: Comment out FileImpl; change VolumeImpl
Time: September 18, 1979 2:20 PM By: Forrest Action: Change to use Boot rather than Storage/BootChannel; eliminate obsolete items
Time: October 1, 1979 5:27 PM By: Forrest Action: Add MarkerPageImpl
Time: February 3, 1980 2:29 PM By: McJones Action: Change VolumeImpl results: drop debugClass, replace debuggerDevice with DebuggerDeviceType and debuggerDeviceOrdinal
Time: May 29, 1980 9:54 AM By: Luniewski Action: Added PhysicalVolumeImpl, ScavengeImpl. Made VolumeImpl not return a value and take pointers to debuggerDeviceType and debuggerDeviceOrdinal so it can both set them and access them. Deleted VolumeImplB.

```
Fonts: DEFINITIONS =
    BEGIN

FontRecord:  TYPE = RECORD [      -- Quad-word aligned
    font:    LONG POINTER TO FontBody,
    rgflags: Rgflags,
    height:  CARDINAL
    ];

Rgflags:  TYPE = LONG POINTER TO PACKED ARRAY CHARACTER OF Flags;

Flags:   TYPE = RECORD [
    pad:     BOOLEAN,
    stop:    BOOLEAN];


CharEntry:   TYPE = MACHINE DEPENDENT RECORD [
    leftKern: BOOLEAN,            -- If true, character has one bit hanging out past left edge
    rightKern:    BOOLEAN,            -- If true, character has one bit hanging out past right edge
    offset:  CARDINAL [0..37777B], -- 14 bit offset to first word for this character
    mica:    CARDINAL];           -- Mica width for this characer

FontBody:    TYPE = MACHINE DEPENDENT RECORD [
    char:        ARRAY CHARACTER OF CharEntry,                  -- 512 words
    widths:      PACKED ARRAY CHARACTER OF CARDINAL[0..377B],   -- 128 words
    bits:    ARRAY [0..0) OF UNSPECIFIED];
```

-- Note that the 14 bit offset to the character is measured from the bits origin, not D0Font. This is a word offset to the first scan line of the character. As a temporary implementation restriction, each scan line of the character starts on a word boundary.

```
END.    -- of Fonts
```

LOG
7-25-80 -- Guyton -- Created.
July 30, 1980 10:35 AM -- Jim Frandeen -- added D0FontHandle.
July 30, 1980 10:35 AM -- Jim Frandeen -- Remove D0FontHandle.

```
-- Frame.Mesa  Edited by McJones on April 18, 1980  10:09 AM

DIRECTORY
  PrincOps: FROM "PrincOps" USING [
    ControlLink, FrameHandle, GlobalFrameHandle, returnOffset],
  Mopcodes: FROM "Mopcodes" USING [
    zALLOC, zFREE, zGADRB, zKFCB, zLADRB, zLLB, zSLB],
  SDDefs: FROM "SDDefs" USING [sCopy];

Frame: DEFINITIONS =

  BEGIN OPEN PrincOps;

  GetReturnLink: PROCEDURE RETURNS [ControlLink] =
    MACHINE CODE BEGIN Mopcodes.zLLB, returnOffset END;
  GetReturnFrame: PROCEDURE RETURNS [FrameHandle] = LOOPHOLE[GetReturnLink];
  SetReturnLink: PROCEDURE [ControlLink] =
    MACHINE CODE BEGIN Mopcodes.zSLB, returnOffset END;
  SetReturnFrame: PROCEDURE [FrameHandle] = LOOPHOLE[SetReturnLink];

  MyLocalFrame: PROCEDURE RETURNS [FrameHandle] =
    MACHINE CODE BEGIN Mopcodes.zLADRB, 0 END;
  MyGlobalFrame: PROCEDURE RETURNS [GlobalFrameHandle] =
    MACHINE CODE BEGIN Mopcodes.zGADRB, 0 END;

  Copy: PROCEDURE [old: GlobalFrameHandle] RETURNS [new: GlobalFrameHandle] =
    MACHINE CODE BEGIN Mopcodes.zKFCB, SDDefs.sCopy END;

  Alloc: PROCEDURE [CARDINAL] RETURNS [POINTER] =
    MACHINE CODE BEGIN Mopcodes.zALLOC END;
  Free: PROCEDURE [POINTER] =
    MACHINE CODE BEGIN Mopcodes.zFREE END;

  END.
```

```
DIRECTORY
    BufferDefs: FROM "BufferDefs" USING [Buffer],
    RS232C: FROM "RS232C" USING [ChannelHandle, LineSpeed],
    SpecialSystem: FROM "SpecialSystem" USING [ProcessorID];

HalfDuplex: DEFINITIONS =
BEGIN

-- procedures

Initialize: PROCEDURE [chHandle: RS232C.ChannelHandle, lineSpeed: RS232C.LineSpeed, ourHostNumber: SpecialSystem.
        ProcessorID];
Destroy: PROCEDURE;

WaitToSend: PROCEDURE;
SendCompleted: PROCEDURE [moreToSend: BOOLEAN];
CheckForTurnAround: PROCEDURE [b: BufferDefs.Buffer] RETURNS [throwAway: BOOLEAN];
END.
```

LOG

Time: July 14, 1980 10:24 AM By: Garlick Action: Created file

HeadStartChain: DEFINITIONS =

BEGIN

-- NOTE: When AR 3536 is fixed, the modules HeadStartChain, DriverStartChain, and StoreDriverStartChain may be replace with single module StartChain.

-- Mechanism used to implement starting extensible set of modules (e.g. heads, drivers, test programs)

-- We refer to a member of the set of modules as a "link"; we refer to the module wishing to start the set as the "master link".
-- Each link should export Start with a body consisting simply of a call on Start imported through a second instance of StartChain (e.g. IMPORTS RemainingHeads: StartChain).
-- The master link should export Start with a body which just returns and should also contain a call on Start imported through a second instance of StartChain named EntireStartChain.
-- The machinery is completed with a configuration which sets up a chain of StartChain imports-exports from EntireStartChain, through each of the links, and back to the master link.

Start: PROCEDURE;
    -- Start the (rest of the) chain

END.

LOG

Time: January 31, 1980 11:09 AM By: McJones Action: Create file

```
DIRECTORY
    Space USING [Handle, nullHandle, PageCount, wordsPerPage];

Heap: DEFINITIONS IMPORTS Space =

BEGIN

-- Dynamic storage allocation: support for Mesa uncounted zones.

-- This interface allows the creation and management of an arbitrary number of heaps, or dynamic storage allocation pools, each
corresponding to a Mesa uncounted zone.  A heap may be in hyperspace or in the MDS.

-- It is intended that Mesa's NEW and FREE be used to allocate and deallocate nodes from a heap (although for compatibility with existing
code, explicit procedures for this purpose are also provided).

    -- Types
NWords: TYPE = [0..32766); -- limit on node size

    -- Constants
defaultSwapUnit: Space.PageCount = LAST[Space.PageCount]; -- initial or extension heap is one swap unit
minimumNodeSize: READONLY NWords; -- implementation limit on threshold
systemZone: READONLY UNCOUNTED ZONE;
systemMDSZone: READONLY MDSZone;

    -- Creating and deleting heaps
Create: PROCEDURE [
        initial: Space.PageCount,
        parent: Space.Handle ← Space.nullHandle,
        increment: Space.PageCount ← 1,
        swapUnit: Space.PageCount ← defaultSwapUnit,
        threshold: NWords ← minimumNodeSize, -- smaller allocation request will be rounded up to this size
        largeNodeThreshold: NWords ← Space.wordsPerPage/2, -- larger allocation request will be in separate space
        ownerChecking: BOOLEAN ← FALSE,
        checking: BOOLEAN ← FALSE]
    RETURNS [UNCOUNTED ZONE];
CreateMDS: PROCEDURE [
        initial: Space.PageCount,
        increment: Space.PageCount ← 1,
        swapUnit: Space.PageCount ← defaultSwapUnit,
        threshold: NWords ← minimumNodeSize, -- smaller allocation request will be rounded up to this size
        largeNodeThreshold: NWords ← Space.wordsPerPage/2, -- larger allocation request will be in separate space
        ownerChecking: BOOLEAN ← FALSE,
        checking: BOOLEAN ← FALSE]
    RETURNS [MDSZone];
Delete: PROCEDURE [z: UNCOUNTED ZONE, checkEmpty: BOOLEAN ← FALSE];
DeleteMDS: PROCEDURE [z: MDSZone, checkEmpty: BOOLEAN ← FALSE];

    -- Miscellaneous
GetAttributes: PROCEDURE [z: UNCOUNTED ZONE] RETURNS [parent: Space.Handle, heapPages, largeNodePages, increment, swapUnit:
        Space.PageCount, threshold, largeNodeThreshold: NWords, ownerChecking, checking: BOOLEAN];
GetAttributesMDS: PROCEDURE [z: MDSZone] RETURNS [heapPages, largeNodePages, increment, swapUnit: Space.PageCount,
        threshold, largeNodeThreshold: NWords, ownerChecking, checking: BOOLEAN];
Expand: PROCEDURE [z: UNCOUNTED ZONE, pages: Space.PageCount];
ExpandMDS: PROCEDURE [z: MDSZone, pages: Space.PageCount];
Prune: PROCEDURE [z: UNCOUNTED ZONE];
PruneMDS: PROCEDURE [z: MDSZone];
CheckOwner: PROCEDURE [p: LONG POINTER];
SetChecking: PROCEDURE [z: UNCOUNTED ZONE, checking: BOOLEAN];
SetCheckingMDS: PROCEDURE [z: MDSZone, checking: BOOLEAN];

    -- Errors
Error: ERROR [type: ErrorType];
ErrorType: TYPE = {unsuitableParent, insufficientSpace, invalidHeap, invalidNode, invalidZone, invalidOwner, otherError};

    -- Synonyms
    -- NOTE: These are provided only to ease conversion of existing client code;
    -- it is preferable for clients to use uncounted zones and NEW and FREE directly.
Handle: TYPE = UNCOUNTED ZONE;
MDSHandle: TYPE = MDSZone;
MakeNode: PROCEDURE [z: UNCOUNTED ZONE ← systemZone, n: NWords] RETURNS [LONG POINTER];
FreeNode: PROCEDURE [z: UNCOUNTED ZONE ← systemZone, p: LONG POINTER];
```

```
MakeMDSNode: PROCEDURE [z: MDSZone ← systemMDSZone, n: NWords] RETURNS [POINTER];
FreeMDSNode: PROCEDURE [z: MDSZone ← systemMDSZone, p: POINTER];
MakeString: PROCEDURE [z: UNCOUNTED ZONE ← systemZone, maxlength: CARDINAL] RETURNS [LONG STRING] =
    INLINE {RETURN[z.NEW[StringBody[maxlength]]]};
FreeString: PROCEDURE [z: UNCOUNTED ZONE ← systemZone, s: LONG STRING] =
    INLINE {z.FREE[@s]}; -- note this sets local s to NIL
MakeMDSString: PROCEDURE [z: MDSZone ← systemMDSZone, maxlength: CARDINAL] RETURNS [STRING] =
    INLINE {RETURN[z.NEW[StringBody[maxlength]]]};
FreeMDSString: PROCEDURE [z: MDSZone ← systemMDSZone, s: STRING] =
    INLINE {z.FREE[@s]}; -- note this sets local s to NIL

END.
```

LOG

Time: May 23, 1980 1:14 PM	*By: McJones*	*Action: Created file*
Time: July 8, 1980 5:15 PM	*By: McJones*	*Action: Adapted for new-style uncounted zones*
Time: August 5, 1980 1:59 PM	*By: McJones*	*Action: Added Expand[MDS]; systemZONE = > systemZone; invalidZONE = > invalidZone; converted {Make,Free}[MDS]String to INLINE; added [MDS]Handle synonyms; decreased LAST[NWords]*

```
-- Definitions:
        -- A handle is strict if handle.page is the first page of a space or swap unit.
        -- A handle is relaxed if handle.page is any other page of a space or swap unit.

DIRECTORY
    CachedSpace: FROM "CachedSpace" USING [Desc, Handle],
    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];
    -- Returns first "n'th cousin" starting within count pages of beginning of space, handleNull if none.

GetDescriptor: PROCEDURE [pDescResult: POINTER TO CachedSpace.Desc, handle: CachedSpace.Handle]
                    RETURNS [validSpace: BOOLEAN, validSwapUnit: BOOLEAN];
    -- Various argument handles produce various results, as follows:
    --  handle                  pDescResult                          validSpace  validSwapUnit
    --  strict space handle     desc of space, state~ = missing      TRUE        FALSE
    --  relaxed space handle    desc of space, state~ = missing      FALSE       FALSE
    --  strict swap unit handle desc of parent space, state~ = missing FALSE     TRUE
    --  relaxed swap unit handle desc of parent space, state~ = missing FALSE    FALSE
    --  invalid                 dummy desc, state = missing          FALSE       FALSE

GetInterval: PROCEDURE [handle: CachedSpace.Handle] RETURNS [validSpace: BOOLEAN, validSwapUnit: BOOLEAN, interval:
                    VM.Interval];
    -- Various argument handles produce various results, as follows:
    --  handle                  interval                 validSpace  validSwapUnit
    --  strict space handle     interval of space        TRUE        FALSE
    --  relaxed space handle    interval of space        FALSE       FALSE
    --  strict swap unit handle interval of swap unit    FALSE       TRUE
    --  relaxed swap unit handle interval of swap unit   FALSE       FALSE
    --  invalid                 [0,0]                    FALSE       FALSE

ValidSpaceOrSwapUnit: PROCEDURE [handle: CachedSpace.Handle] RETURNS [BOOLEAN];
    -- Returns TRUE iff handle is a strict space handle or a strict swap unit handle.

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: November 8, 1979 8:48 AM	*By: Knutsen*	*Action: Procedures return* validSpace *and* validSwapUnit *instead of* validHandle. GetState *replaced by* ValidSpaceOrSwapUnit.

```
DIRECTORY
    Environment: FROM "Environment" USING [LongNumber],
    Mopcodes: FROM "Mopcodes" USING [zAND, zBLT, zBLTL, zDIV, zEXCH, zLDIV, zLIB, zLIN1, zLINB, zMUL, zOR, zPOP, zPUSH,
        zSHIFT, zXOR];

Inline: DEFINITIONS =

BEGIN OPEN Mopcodes;

LongCARDINAL: TYPE = LONG CARDINAL;

LongNumber: TYPE = Environment.LongNumber;

COPY: PROCEDURE [from: POINTER, nwords: CARDINAL, to: POINTER] =
    MACHINE CODE BEGIN zBLT END;

LongCOPY: PROCEDURE [from: LONG POINTER, nwords: CARDINAL, to: LONG POINTER] =
    MACHINE CODE BEGIN zBLTL END;

DIVMOD: PROCEDURE [num, den: CARDINAL] RETURNS [quotient, remainder: CARDINAL] =
    MACHINE CODE BEGIN zDIV; zPUSH END;

PUSH: PROCEDURE RETURNS [WORD] = MACHINE CODE BEGIN zPUSH END;

LDIVMOD: PROCEDURE [numlow: WORD, numhigh: CARDINAL, den: CARDINAL] RETURNS [quotient, remainder: CARDINAL] =
    MACHINE CODE BEGIN zLDIV; zPUSH END;

LongMult: PROCEDURE [CARDINAL, CARDINAL] RETURNS [product: LONG CARDINAL] =
    MACHINE CODE BEGIN zMUL; zPUSH END;

LongDiv: PROCEDURE [num: LONG CARDINAL, den: CARDINAL] RETURNS [CARDINAL] =
    MACHINE CODE BEGIN zLDIV END;

LongDivMod: PROCEDURE [num: LONG CARDINAL, den: CARDINAL] RETURNS [quotient, remainder: CARDINAL] =
    MACHINE CODE BEGIN zLDIV; zPUSH END;

BitOp: TYPE = PROCEDURE [UNSPECIFIED, UNSPECIFIED] RETURNS [UNSPECIFIED];

BITAND: BitOp = MACHINE CODE BEGIN zAND END;

BITOR: BitOp = MACHINE CODE BEGIN zOR END;

BITXOR: BitOp = MACHINE CODE BEGIN zXOR END;

BITNOT: PROCEDURE [UNSPECIFIED] RETURNS [UNSPECIFIED] =
    MACHINE CODE BEGIN zLIN1; zXOR END;

BITSHIFT: PROCEDURE [value: UNSPECIFIED, count: INTEGER] RETURNS [UNSPECIFIED] =
    MACHINE CODE BEGIN zSHIFT END;

LowHalf: PROCEDURE [LONG UNSPECIFIED] RETURNS [UNSPECIFIED] =
    MACHINE CODE BEGIN zPOP END;

HighHalf: PROCEDURE [LONG UNSPECIFIED] RETURNS [UNSPECIFIED] =
    MACHINE CODE BEGIN zEXCH; zPOP END;

LowByte: PROCEDURE [UNSPECIFIED] RETURNS [UNSPECIFIED] =
    MACHINE CODE BEGIN zLIB, 377B; zAND END;

HighByte: PROCEDURE [UNSPECIFIED] RETURNS [UNSPECIFIED] =
    MACHINE CODE BEGIN zLINB, 370B; zSHIFT END;

END.
```

LOG

Time: May 22, 1979 4:47 PM By: Lauer Action: Created file by renaming Mesa > System > InlineDefs.mesa
Time: June 7, 1979 5:46 PM By: McJones Action: Fixed shift count in HighByte
Time: January 25, 1980 4:01 PM By: Forrest Action: Equated Long number to Env.longNumber

```
-- File: JLevelIVKeys.mesa  Modified by:
-- Fay, October 23, 1980  4:31 PM  Prepended 'J' to module name.
-- Karlton, Oct 22, 1980 5:09 PM
-- Johnsson, Oct 22, 1980 5:00 PM

DIRECTORY KeyStations;

JLevelIVKeys: DEFINITIONS =
  BEGIN

  DownUp: TYPE = KeyStations.DownUp;
  Bit: TYPE = KeyStations.Bit;
  KeyBits: TYPE = PACKED ARRAY KeyName OF DownUp;

  KeyName: TYPE = MACHINE DEPENDENT {
    Red(KeyStations.M1), Blue(KeyStations.M3), Five(KeyStations.k16),
    Four(KeyStations.k12), Six(KeyStations.k20), E(KeyStations.k10),
    Seven(KeyStations.k24), D(KeyStations.k11), U(KeyStations.k26), V(KeyStations.k17),
    Zero(KeyStations.k36), K(KeyStations.k31), Dash(KeyStations.k40), P(KeyStations.k38),
    Slash(KeyStations.k41), Defaults(KeyStations.T9), Copy(KeyStations.L6),
    BS(KeyStations.A2), Three(KeyStations.k8),Two(KeyStations.k4), W(KeyStations.k6),
    Q(KeyStations.k2), S(KeyStations.k7), A(KeyStations.k3), Nine(KeyStations.k32),
    I(KeyStations.k30), X(KeyStations.k9), O(KeyStations.k34), L(KeyStations.k35),
    Comma(KeyStations.k33), Quote(KeyStations.k43), RightBracket(KeyStations.k45),
    Special(KeyStations.R11), Undo(KeyStations.R6), One(KeyStations.k1),
    Center(KeyStations.T2), ParaTab(KeyStations.A1), F(KeyStations.k15),
    Open(KeyStations.L11), C(KeyStations.k13), J(KeyStations.k27), B(KeyStations.k21),
    Z(KeyStations.k5), LeftHandakuonShift(KeyStations.A5), Period(KeyStations.k37),
    SemiColon(KeyStations.k39), NewPara(KeyStations.A4), Para(KeyStations.k46),
    Delete(KeyStations.L3), Move(KeyStations.L9), R(KeyStations.k14), T(KeyStations.k18),
    G(KeyStations.k19), Y(KeyStations.k22), H(KeyStations.k23), Eight(KeyStations.k28),
    N(KeyStations.k25), M(KeyStations.k29), Lock(KeyStations.A3), Hiragana(KeyStations.A7),
    Half(KeyStations.k42), Equal(KeyStations.k44), RightDakuonShift(KeyStations.A6),
    Stop(KeyStations.R12), Props(KeyStations.L12), Next(KeyStations.R1),
    Margins(KeyStations.R5), English(KeyStations.A9), Katakana(KeyStations.A8),
    SameAs(KeyStations.L8), Find(KeyStations.L5),
    Again(KeyStations.L2), Help(KeyStations.R2), Expand(KeyStations.R7),
    Tab(KeyStations.k48), Bold(KeyStations.T3), Italic(KeyStations.T4),
    Underlined(KeyStations.T5), Superscript(KeyStations.T6), Subscript(KeyStations.T7),
    Smaller(KeyStations.T8), LeftDakuonShift(KeyStations.k47), Font(KeyStations.R8),
    Space(KeyStations.A11), RightHandakuonShift(KeyStations.A12)};


  RightShift: KeyName = RightDakuonShift;
  LeftShift: KeyName = LeftHandakuonShift;

  END.
```

```
-- This interface supplies FileManager-level file facilities for higher-level software within Pilot (PilotControl, TransactionMgr, VMMgr).

DIRECTORY
    Device USING [Type],
    File USING [Capability, ID, PageCount, PageNumber, Type],
    SpecialFile USING [Link],
    System USING [VolumeID],
    Transaction USING [Handle],
    VMMapLog USING [Entry],
    Volume USING [ID];

KernelFile: DEFINITIONS =
BEGIN

ExistingFile: ERROR;

CreateWithID: PROCEDURE [volume: System.VolumeID, initialSize: File.PageCount, type: File.Type, id: File.ID];
    -- Same as File.Create, but uses supplied ID.  May only be used for transaction crash recovery!  Raises ExistingFile if a file is found
    with same ID.

GetBootLocation: PROCEDURE [file: File.Capability, filePage: File.PageNumber ← 0]
                    RETURNS [deviceType: Device.Type, deviceOrdinal: CARDINAL, link: SpecialFile.Link];
    -- Returns device type, ordinal of device within its type, and disk address of given boot file and page.
    -- Only one process may be operating on file at a time.  The FileMgr does not serialize calls to this procedure with other calls to procedures in
    File/KernelFile.

GetFileAttributes: PROCEDURE [file: File.Capability] RETURNS [size: File.PageCount, immutable: BOOLEAN, readOnly: BOOLEAN];
    -- In a single call, returns those attributes of file that are needed by the VMMgr.  (Combines File.GetAttributes, File.GetSize, and
    File.ShowCapability, plus readOnly also reflects any readOnly access to the volume.)
    -- May raise File.Unknown, Volume.Unknown.

GetFilePoint: PROCEDURE [pEntry: LONG POINTER TO VMMapLog.Entry, pFile: POINTER TO File.Capability, filePage: File.PageNumber];
    -- Sets pEntry↑ to description of run of backing-store pages beginning with given filePage;  page field of returned Entry is garbage.
    -- Only one process may be operating on file at a time.  The FileMgr does not serialize calls to this procedure with other calls to procedures in
    File/KernelFile.

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

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

LogContents: PROCEDURE [transaction: Transaction.Handle, file: File.Capability, base: File.PageNumber, count: File.PageCount];
    -- Saves the contents of the count pages of file starting at base in the transaction's log file.  The data to be logged must be currently
    in the client's file.  That is, if the data is mapped in VM and is dirty, it must be forced out to the backing file before calling
    LogContents.
    -- May raise Transaction.InvalidHandle, File.Unknown, Volume.Unknown.

MakeMutable: PROCEDURE [file: File.Capability];
    -- Inverse of File.MakeImmutable.  May only be used for transaction crash recovery and transaction abort!
    -- May raise File.Unknown, Volume.Unknown.

MakeTemporary: PROCEDURE [file: File.Capability];
    -- Inverse of File.MakePermanent.  May only be used for transaction crash recovery and transaction abort!
    -- May raise File.Unknown, Volume.Unknown.

Pin: PROCEDURE [file: File.Capability, page: File.PageNumber ← 0, count: File.PageCount ← defaultPageCount];
    -- Pins the file descriptor of file and page group descriptors for the count pages of the file starting at page into the File Cache.
    defaultPageCount means to the end of the file.

defaultPageCount: File.PageCount = LAST[File.PageCount];

Unpin: PROCEDURE [file: File.Capability, page: File.PageNumber ← 0, count: File.PageCount ← defaultPageCount];
    -- Unpins the file descriptor of file and page group descriptors for the count pages of the file starting at page from the File Cache.
    defaultPageCount means to the end of the file.

END.
```

LOG

(For earlier entries, please see version archived with Pilot 3.0.)

February 3, 1980 1:05 PM	*McJones*	*Replaced device parameter to GetBootLocation with deviceType and deviceHandle.*
May 15, 1980 4:50 PM	*McJones*	*Added page and count parameters to Pin.*

July 1, 1980 8:22 PM	*Gobbel*	*Added MakeMutable and MakeTemporary.*
July 14, 1980 2:07 PM	*Gobbel*	*Deleted* DeleteTemps *and* OpenVolume.
July 22, 1980 5:10 PM	*Luniewski*	*Added GetFileAttributes.*
July 29, 1980 1:16 PM	*Gobbel*	*Added* CreateWithID.
August 21, 1980 9:03 AM	*Knutsen*	*Added* LogContents, Unpin, *deleted* GetFileDescriptor.
September 2, 1980 4:35 PM	*Gobbel*	*Added* ExistingFile.

```
-- This interface defines Space operations for clients inside TestPilotKernel.

DIRECTORY
    Space USING [Handle],
    Transaction USING [Handle];

KernelSpace: DEFINITIONS =

BEGIN

ReleaseFromTransaction: PROCEDURE [space: Space.Handle, transaction: Transaction.Handle, andInvalidate: BOOLEAN ← FALSE];
    -- If space is not part of transaction, this procedure is a no-op.  If space is part of transaction, performs the folowing actions:
    -- (A) If andInvalidate = TRUE, any swap units now in memory will be considered invalid and will be discarded.  They will not be
    written to the backing file even if they are dirty.
    -- (B) After optionally performing the above invalidation, marks space "not part of any transaction".  Any regions of the space which
    were client-writable but were secretly write-protected to catch the client's first attempt to store into them will be put back to
    writable.

END.
```

LOG

August 1, 1980 9:58 AM	*Knutsen*	*Created file.*
August 20, 1980 2:19 PM	*Knutsen*	*Renamed from SpaceInternal to KernelSpace.*
September 15, 1980 9:18 AM	*Knutsen*	*Added transaction parameter.*

```
DIRECTORY
    KeyStations USING [KeyBits];

KeyboardFace: DEFINITIONS =

BEGIN

keyboard: READONLY LONG POINTER TO READONLY KeyStations.KeyBits;

END.
```

LOG

Time: February 6, 1980 5:59 PM By: Redell Action: Created file
Time: July 28, 1980 5:37 PM By: McJones Action: Keystation assignments moved to KeyStations.mesa; keyboard added as correct spelling of Keyboard

```
-- File: Keys.mesa  Modified by:
-- Karlton, May 10, 1980 1:53 PM

DIRECTORY
  KeyStations: FROM "KeyStations";

Keys: DEFINITIONS =
  BEGIN

  DownUp: TYPE = KeyStations.DownUp;
  Bit: TYPE = KeyStations.Bit;
  KeyBits: TYPE = PACKED ARRAY KeyName OF DownUp;

  KeyName: TYPE = MACHINE DEPENDENT {
    Keyset1(KeyStations.KS1), Keyset2(KeyStations.KS2), Keyset3(KeyStations.KS3),
    Keyset4(KeyStations.KS4), Keyset5(KeyStations.KS5), Red(KeyStations.M1),
    Blue(KeyStations.M3), Yellow(KeyStations.M2), Five(KeyStations.k16),
    Four(KeyStations.k12), Six(KeyStations.k20), E(KeyStations.k10), Seven(KeyStations.k24),
    D(KeyStations.k11), U(KeyStations.k26), V(KeyStations.k17), Zero(KeyStations.k36),
    K(KeyStations.k31), Dash(KeyStations.k40), P(KeyStations.k38), Slash(KeyStations.k41),
    BackSlash(KeyStations.T9), LF(KeyStations.L6), BS(KeyStations.A2), Three(KeyStations.k8),
    Two(KeyStations.k4), W(KeyStations.k6), Q(KeyStations.k2), S(KeyStations.k7),
    A(KeyStations.k3), Nine(KeyStations.k32), I(KeyStations.k30), X(KeyStations.k9),
    O(KeyStations.k34), L(KeyStations.k35), Comma(KeyStations.k33), Quote(KeyStations.k43),
    RightBracket(KeyStations.k45), Spare2(KeyStations.R11), Spare1(KeyStations.R6),
    One(KeyStations.k1), ESC(KeyStations.T2), TAB(KeyStations.A1), F(KeyStations.k15),
    Ctrl(KeyStations.L11), C(KeyStations.k13), J(KeyStations.k27), B(KeyStations.k21),
    Z(KeyStations.k5), LeftShift(KeyStations.A5), Period(KeyStations.k37),
    SemiColon(KeyStations.k39), Return(KeyStations.A4), Arrow(KeyStations.k46),
    DEL(KeyStations.L3), FL3(KeyStations.L9), R(KeyStations.k14), T(KeyStations.k18),
    G(KeyStations.k19), Y(KeyStations.k22), H(KeyStations.k23), Eight(KeyStations.k28),
    N(KeyStations.k25), M(KeyStations.k29), Lock(KeyStations.A3), Space(KeyStations.A7),
    LeftBracket(KeyStations.k42), Equal(KeyStations.k44), RightShift(KeyStations.A6),
    Spare3(KeyStations.R12), FL4(KeyStations.L12), FR5(KeyStations.R1), R5(KeyStations.R5),
    R9(KeyStations.R9), L10(KeyStations.L10), L7(KeyStations.L7), L4(KeyStations.L4),
    L1(KeyStations.L1), A9(KeyStations.A9), R10(KeyStations.R10), A8(KeyStations.A8),
    L8(KeyStations.L8), L5(KeyStations.L5), L2(KeyStations.L2), R2(KeyStations.R2),
    R7(KeyStations.R7), R4(KeyStations.R4), D2(KeyStations.D2), D1(KeyStations.D1),
    Key48(KeyStations.k48), T1(KeyStations.T1), T3(KeyStations.T3), T4(KeyStations.T4),
    T5(KeyStations.T5), T6(KeyStations.T6), T7(KeyStations.T7), T8(KeyStations.T8),
    T10(KeyStations.T10), R3(KeyStations.R3), Key47(KeyStations.k47), A10(KeyStations.A10),
    R8(KeyStations.R8)};

-- Alto II names for some keys can be different

  FL1: KeyName = DEL;
  FL2: KeyName = LF;
  BW: KeyName = Spare1;
  FR1: KeyName = Spare3;
  Swat: KeyName = FR1;
  FR2: KeyName = BackSlash;
  FR3: KeyName = Arrow;
  FR4: KeyName = Spare2;

  END.
```

```
-- File: KeyStations.mesa last edited by
-- Karlton; Oct 10, 1980 4:41 PM

KeyStations: DEFINITIONS =
  BEGIN

  DownUp: TYPE = {down, up};
  KeyBits: TYPE = PACKED ARRAY KeyStation OF DownUp;
  KeyStation: TYPE = [0..112);

  Bit: TYPE = KeyStation;  -- for convenience

  -- Positions in Keyboard array correspond to "keystations" common to all keyboards.
  -- Higher-level software must associate characters with each keystation as appropriate.

  -- Keystations are of five types:
  --   1) Typing keys: (alphanumerics, punctuation, tab, CR, etc.)
  --   2) Function Keys: Left, right, and top function groups
  --   3) Mouse buttons
  --   4) Keyset (Not on OIS keyboards)
  --   5) Diagnostic (pseudo-keys for hardware diagnostic purposes)

-- Main typing array

  k1: Bit = 48; k2: Bit = 35;   k3: Bit = 37;   k4: Bit = 33;   k5: Bit = 56;
  k6: Bit = 34; k7: Bit = 36;   k8: Bit = 32;   k9: Bit = 40;   k10: Bit = 19;
  k11: Bit = 21;        k12: Bit = 17;  k13: Bit = 53;  k14: Bit = 64;  k15: Bit = 51;
  k16: Bit = 16;        k17: Bit = 23;  k18: Bit = 65;  k19: Bit = 66;  k20: Bit = 18;
  k21: Bit = 55;        k22: Bit = 67;  k23: Bit = 68;  k24: Bit = 20;  k25: Bit = 70;
  k26: Bit = 22;        k27: Bit = 54;  k28: Bit = 69;  k29: Bit = 71;  k30: Bit = 39;
  k31: Bit = 25;        k32: Bit = 38;  k33: Bit = 43;  k34: Bit = 41;  k35: Bit = 42;
  k36: Bit = 24;        k37: Bit = 58;  k38: Bit = 27;  k39: Bit = 59;  k40: Bit = 26;
  k41: Bit = 28;        k42: Bit = 74;  k43: Bit = 44;  k44: Bit = 75;  k45: Bit = 45;
  k46: Bit = 61;        k47: Bit = 107; k48: Bit = 97;

  A1: Bit = 50; A2: Bit = 31;   A3: Bit = 72;   A4: Bit = 60;   A5: Bit = 57;
  A6: Bit = 76; A7: Bit = 73;   A8: Bit = 88;   A9: Bit = 86;   A10: Bit = 108;
  A11: Bit = 110;       A12: Bit = 111;

-- Function key arrays (left, right, and top)

  L1: Bit = 85; L2: Bit = 91;   L3: Bit = 62;   L4: Bit = 84;   L5: Bit = 90;
  L6: Bit = 30; L7: Bit = 83;   L8: Bit = 89;   L9: Bit = 63;   L10: Bit = 82;
  L11: Bit = 52;        L12: Bit = 78;

  R1: Bit = 79; R2: Bit = 92;   R3: Bit = 106;  R4: Bit = 94;   R5: Bit = 80;
  R6: Bit = 47; R7: Bit = 93;   R8: Bit = 109;  R9: Bit = 81;   R10: Bit = 87;
  R11: Bit = 46;        R12: Bit = 77;

  T1: Bit = 98; T2: Bit = 49;   T3: Bit = 99;   T4: Bit = 100;  T5: Bit = 101;
  T6: Bit = 102;        T7: Bit = 103;  T8: Bit = 104;  T9: Bit = 29;   T10: Bit = 105;

-- Mouse Buttons (left to right)

  M1: Bit = 13; M2: Bit = 15;   M3: Bit = 14;

-- KeySet

  KS1: Bit = 8; KS2: Bit = 9;   KS3: Bit = 10;  KS4: Bit = 11;  KS5: Bit = 12;

-- Diagnostic keystations

  D1: Bit = 96; D2: Bit = 95;

  END.
-- LOG
-- Feb 6, 1980  5:59 PM By: Redell      Created file
-- May 2, 1980  6:05 PM By: Redell      Updated bit assignments
-- May 5, 1980  2:32 PM By: Karlton     Change updown TYPE to DownUp
-- May 8, 1980  5:34 PM By: Karlton     Reflect actual state of Level 4 double keys
-- May 10, 1980 1:35 PM By: Karlton     Renamed to KeyStations & make Definitions Only
-- Oct 10, 1980 4:39 PM By: Karlton     added A11 and A12
```

```
DIRECTORY
    DiskChannel USING [Address, Drive],
    Environment USING [PageNumber],
    File USING [PageCount, PageNumber],
    FileInternal USING [Descriptor, Operation, PageGroup],
    PilotDisk USING [Label],
    VolumeInternal USING [PageNumber];

LabelTransfer: DEFINITIONS =

BEGIN

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

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

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

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

VerifyLabels: PROCEDURE [
    file: FileInternal.Descriptor,
    pageGroup: FileInternal.PageGroup,
    expectErrorAfterFirstPage: BOOLEAN ← FALSE]
     -- mainly for scavenger; TRUE inhibits low level retries if labels don't match AFTER first page of run
    RETURNS [labelsValid: BOOLEAN, countValid: File.PageCount];

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

WriteLabelAndData: PROCEDURE [
    file: FileInternal.Descriptor,
    filePage: File.PageNumber,
    volumePage: VolumeInternal.PageNumber,
    memoryPage: Environment.PageNumber,
    bootChainLink: DiskChannel.Address ← NULL];

LabelStatus: TYPE = {valid, invalid, diskError};

END.
```

LOG

Time: June 15, 1978 11:03 PM By: Redell Action: Create file
Time: March 13, 1979 7:28 PM By: Redell Action: Add ReadRootLabel
Time: August 1, 1979 12:13 PM By: Redell Action: Use FilePageLabel
Time: August 13, 1979 3:18 PM By: Redell Action: Change WriteLabelAndData to do runs of pages with optional chaining; don't loophole drive
Time: September 17, 1979 4:36 PM By: Forrest Action: Add LabelStatus, and caused ReadRootLabel to return it. Should other label procs use this also?
Time: January 9, 1980 2:24 PM By: Gobbel Action: Add countValid to RETURNS of VerifyLabels
Time: August 14, 1980 9:50 AM By: McJones Action: WriteLabelsAndData = > WriteLabelAndData; FilePageLabel = > PilotDisk
Time: October 11, 1980 7:40 PM By: Forrest Action: add crock (for scavenger) that causes VerifyLabels to not retry if at least the first page of the run is OK.

```
-- File: LevelIIIKeys.mesa  Modified by:
-- Karlton, Jul 17, 1980 7:50 PM

DIRECTORY KeyStations;

LevelIIIKeys: DEFINITIONS =
  BEGIN

  DownUp: TYPE = KeyStations.DownUp;
  Bit: TYPE = KeyStations.Bit;
  KeyBits: TYPE = PACKED ARRAY KeyName OF DownUp;

  KeyName: TYPE = MACHINE DEPENDENT {
    Red(KeyStations.M1), Blue(KeyStations.M3), Yellow(KeyStations.M2),
    Five(KeyStations.k16), Four(KeyStations.k12), Six(KeyStations.k20), E(KeyStations.k10),
    Seven(KeyStations.k24), D(KeyStations.k11), U(KeyStations.k26), V(KeyStations.k17),
    Zero(KeyStations.k36), K(KeyStations.k31), Dash(KeyStations.k40), P(KeyStations.k38),
    Slash(KeyStations.k41), Smaller(KeyStations.T9), Copy(KeyStations.L6),
    BS(KeyStations.A2), Three(KeyStations.k8),Two(KeyStations.k4), W(KeyStations.k6),
    Q(KeyStations.k2), S(KeyStations.k7), A(KeyStations.k3), Nine(KeyStations.k32),
    I(KeyStations.k30), X(KeyStations.k9), O(KeyStations.k34), L(KeyStations.k35),
    Comma(KeyStations.k33), Quote(KeyStations.k43), RightBracket(KeyStations.k45),
    Special(KeyStations.R11), R6(KeyStations.R6), One(KeyStations.k1),
    Indent(KeyStations.T2), TAB(KeyStations.A1), F(KeyStations.k15), Again(KeyStations.L11),
    C(KeyStations.k13), J(KeyStations.k27), B(KeyStations.k21), Z(KeyStations.k5),
    LeftShift(KeyStations.A5), Period(KeyStations.k37), SemiColon(KeyStations.k39),
    Return(KeyStations.A4), Para(KeyStations.k46), Delete(KeyStations.L3),
    Move(KeyStations.L9), R(KeyStations.k14), T(KeyStations.k18), G(KeyStations.k19),
    Y(KeyStations.k22), H(KeyStations.k23), Eight(KeyStations.k28), N(KeyStations.k25),
    M(KeyStations.k29), Lock(KeyStations.A3), Space(KeyStations.A7), Half(KeyStations.k42),
    Equal(KeyStations.k44), RightShift(KeyStations.A6), R12(KeyStations.R12),
    Props(KeyStations.L12), Next(KeyStations.R1), Carriage(KeyStations.R5),
    R9(KeyStations.R9), L10(KeyStations.L10), L7(KeyStations.L7), L4(KeyStations.L4),
    L1(KeyStations.L1), A9(KeyStations.A9), A8(KeyStations.A8), GloblRplce(KeyStations.L8),
    Find(KeyStations.L5), Undo(KeyStations.L2), Help(KeyStations.R2), Expand(KeyStations.R7),
    T1(KeyStations.T1), Justify(KeyStations.T3), Center(KeyStations.T4),
    Bold(KeyStations.T5), Italics(KeyStations.T6), Underline(KeyStations.T7),
    Subscript(KeyStations.T8), T10(KeyStations.T10), R3(KeyStations.R3),
    Font(KeyStations.R8)};

  END.
```

```
-- File: LevelIVKeys.mesa  Modified by:
-- Karlton, Jul 17, 1980 7:50 PM

DIRECTORY KeyStations;

LevelIVKeys: DEFINITIONS =
  BEGIN

  DownUp: TYPE = KeyStations.DownUp;
  Bit: TYPE = KeyStations.Bit;
  KeyBits: TYPE = PACKED ARRAY KeyName OF DownUp;

  KeyName: TYPE = MACHINE DEPENDENT {
    Red(KeyStations.M1), Blue(KeyStations.M3), Five(KeyStations.k16),
    Four(KeyStations.k12), Six(KeyStations.k20), E(KeyStations.k10),
    Seven(KeyStations.k24), D(KeyStations.k11), U(KeyStations.k26), V(KeyStations.k17),
    Zero(KeyStations.k36), K(KeyStations.k31), Dash(KeyStations.k40), P(KeyStations.k38),
    Slash(KeyStations.k41), Defaults(KeyStations.T9), Copy(KeyStations.L6),
    BS(KeyStations.A2), Three(KeyStations.k8),Two(KeyStations.k4), W(KeyStations.k6),
    Q(KeyStations.k2), S(KeyStations.k7), A(KeyStations.k3), Nine(KeyStations.k32),
    I(KeyStations.k30), X(KeyStations.k9), O(KeyStations.k34), L(KeyStations.k35),
    Comma(KeyStations.k33), Quote(KeyStations.k43), RightBracket(KeyStations.k45),
    Special(KeyStations.R11), Undo(KeyStations.R6), One(KeyStations.k1),
    Center(KeyStations.T2), ParaTab(KeyStations.A1), F(KeyStations.k15),
    Open(KeyStations.L11), C(KeyStations.k13), J(KeyStations.k27), B(KeyStations.k21),
    Z(KeyStations.k5), LeftShift(KeyStations.A5), Period(KeyStations.k37),
    SemiColon(KeyStations.k39), NewPara(KeyStations.A4), Para(KeyStations.k46),
    Delete(KeyStations.L3), Move(KeyStations.L9), R(KeyStations.k14), T(KeyStations.k18),
    G(KeyStations.k19), Y(KeyStations.k22), H(KeyStations.k23), Eight(KeyStations.k28),
    N(KeyStations.k25), M(KeyStations.k29), Lock(KeyStations.A3), Space(KeyStations.A7),
    Half(KeyStations.k42), Equal(KeyStations.k44), RightShift(KeyStations.A6),
    Stop(KeyStations.R12), Props(KeyStations.L12), Next(KeyStations.R1),
    Margins(KeyStations.R5), SameAs(KeyStations.L8), Find(KeyStations.L5),
    Again(KeyStations.L2), Help(KeyStations.R2), Expand(KeyStations.R7),
    Tab(KeyStations.k48), Bold(KeyStations.T3), Italic(KeyStations.T4),
    Underlined(KeyStations.T5), Superscript(KeyStations.T6), Subscript(KeyStations.T7),
    Smaller(KeyStations.T8), Font(KeyStations.R8)};


  END.
```

```
DIRECTORY
    Boot USING [nullDiskFileID, LVBootFiles, VolumeType],
    File USING [Capability, ID, lastPageNumber, nullCapability, nullID, PageNumber, Type],
    PilotFileTypes USING [PilotRootFileType, tFreePage, tVolumeAllocationMap, tVolumeFileMap],
    Space USING [Handle],
    Volume USING [ID, PageCount],
    VolumeInternal USING [PageNumber];

LogicalVolume: DEFINITIONS =

BEGIN

FileVolumeStatus: TYPE = {ok, volumeUnknown, volumeNotOpen, volumeReadOnly, insufficientSpace, insufficientPermissions};
OpenStatus: TYPE = {ok, wasOpen, Unknown, VolumeNeedsScavenging};
VolumeAccessStatus: TYPE = FileVolumeStatus[ok..volumeReadOnly];

ReadOnlyVolume: ERROR; -- This should really be a public error

BeginScavenging: PROCEDURE [pVID: POINTER TO READONLY Volume.ID];

EndScavenging: PROCEDURE [pVID: POINTER TO READONLY Volume.ID];

OpenLogicalVolume: PROCEDURE [POINTER TO READONLY Volume.ID] RETURNS [OpenStatus];

CloseLogicalVolume: PROCEDURE [POINTER TO READONLY Volume.ID];

PutRootFile: PROCEDURE [pVID: POINTER TO READONLY Volume.ID, type: File.Type, file: POINTER TO READONLY File.Capability];

ScavengeVolume: PROCEDURE [vol: Handle, space: Space.Handle, erase: BOOLEAN];
    -- (Note: A Space.handle is identical to a SimpleSpace.handle.)
VolumeAccess: PROCEDURE [pVID: POINTER TO READONLY Volume.ID, proc: VolumeAccessProc, modify: BOOLEAN ← FALSE] RETURNS
        [VolumeAccessStatus];

VolumeAccessProc: TYPE = PROCEDURE [volume: Handle] RETURNS [updateMarkers: BOOLEAN];

-- The following two procedures are (he says) interim, and used to implement Volume.Open with deleting temps. FileImpl can't export Volume because of a
        conflict in File and Volume Unknown; therefore VolumeImpl.Open is merely a call to FileImpl.OpenAndDeleteTemps, which calls
        VolumeImpl.OpenVolume. Likely, when a volume is closed, context must be flushed from file Impl. Therefore VolumeImpl.Close calls
        FileImpl.CloseAndFlushFiles, which calls VolumeImpl.CloseVolume and goes God only know what else. It's time to rewrite the file manager,
        folks!

OpenVolumeAndDeleteTemps, CloseVolumeAndFlushFiles: PROCEDURE [POINTER TO READONLY Volume.ID];


seal, Seal: CARDINAL = 131313B; -- word zero of valid logical volume root page
currentVersion: CARDINAL = 5; -- increment each time Descriptor below is reformatted

Descriptor: TYPE = MACHINE DEPENDENT RECORD [
        --**~~~~~~~~~~~~~~~~~~~~~~~~ Do not reorder the beginning of this record ~~~~~~~~~~~~~~~~~~~~~~~~~~**
        seal (0): CARDINAL ← Seal, -- absolutely must be 1st field
        version (1): CARDINAL ← currentVersion, -- must be 2nd field
        vID (2): Volume.ID,
        labelLength (7): CARDINAL[0..maxLogicalVolumeLabelLength) ← 0,
        label (10B): PACKED ARRAY [0..maxLogicalVolumeLabelLength) OF CHARACTER ← nullName,
        type (34B): Boot.VolumeType,
        volumeSize (35B): Volume.PageCount,
        bootingInfo (37B): Boot.LVBootFiles ← nullBoot,
        --**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Reorder from here on only~~~~~~~~~~~~~~~~~~~~~~~~ **
        treeLevel (125B): TreeLevel ← LAST[TreeLevel], -- reduce by considering volsize
        changing (126B:0..15): BOOLEAN ← TRUE,
        freePageCount (127B): Volume.PageCount ← 0,
        vamStart (131B): PageNumber ← 1,
        vfmStart (133B): PageNumber ← 2, -- +(volumeSize-1)/(bitsPerWord*wordsPerPage)
        lowerBound (135B): PageNumber ← 3, -- maintained by allocater; first page looked at by allocator
        rootFileID (137B): ARRAY PilotFileTypes.PilotRootFileType OF File.ID ← nullRootFileIDs,
        clientRootFile (214B): File.Capability ← File.nullCapability,
        interval (222B): Intervals ← nullIntervals,
        fill (362B): ARRAY [0..377B-362B) OF WORD ← ALL[0], -- fill to whole page
        checksum (377B): CARDINAL ← 0]; -- Must be the last field

Handle: TYPE = LONG POINTER TO Descriptor;

Interval: TYPE = MACHINE DEPENDENT RECORD [
    key (0): Key,
    volumePage (7): PageNumber,
    nextKey (11B): Key];
```

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

Key: TYPE = MACHINE DEPENDENT RECORD [
    fileID (0): File.ID,
    filePage (5): File.PageNumber];

LSMSeal: CARDINAL = 151515B;                -- word zero of a valid subvolume end marker page
LSMCurrentVersion: CARDINAL = 0;            -- increment each time SubVolEndMarkerDesc is reformatted

-- The subvolume end marker page marks the end of each subvolume on a physical volume and contains non-reconstructable
   information from both the physical and logical (following) volume root pages, and a checksum.
LogicalSubvolumeMarker: TYPE = MACHINE DEPENDENT RECORD [
        seal (0): CARDINAL ← LSMSeal, -- must be 1st field
        version (1): CARDINAL ← LSMCurrentVersion, -- must be 2nd field
        -- "Constant" information, that is set only at logical volume creation
        labelLength (2:0..5): CARDINAL [0..maxLogicalVolumeLabelLength) ← 0,
        type (2:6..7): Boot.VolumeType,
        pad (2:8..15): [0..256) ← 0,
        label (3): PACKED ARRAY [0..maxLogicalVolumeLabelLength) OF CHARACTER ← nullName,
        -- Updatable information
        bootingInfo (27B): Boot.LVBootFiles ← nullBoot,
        clientRootFile (115B): File.Capability ← File.nullCapability];

Level: TYPE = [0..6); -- six levels of worst case branching of 2^4 (= 16) reaches 2^24 pages
TreeLevel: TYPE = Level; -- has two names, TreeLevel and Level

PageNumber: TYPE = VolumeInternal.PageNumber;


-- Procs to get/Set Free, Vam, Vfm (this junk is because we don't have macros!!!!)
-- Following necessary to get around a compiler bug
tFreePage: CARDINAL = LOOPHOLE[PilotFileTypes.tFreePage, CARDINAL];
tVolumeAllocationMap: CARDINAL = LOOPHOLE[PilotFileTypes.tVolumeAllocationMap, CARDINAL];
tVolumeFileMap: CARDINAL = LOOPHOLE[PilotFileTypes.tVolumeFileMap, CARDINAL];
Free: PROCEDURE[v: Handle] RETURNS[File.ID] =
    INLINE BEGIN RETURN[v.rootFileID[tFreePage]]; END;
Vam: PROCEDURE[v: Handle] RETURNS[File.ID] =
    INLINE BEGIN RETURN[v.rootFileID[tVolumeAllocationMap]]; END;
Vfm: PROCEDURE[v: Handle] RETURNS[File.ID] =
    INLINE BEGIN RETURN[v.rootFileID[tVolumeFileMap]]; END;

SetFree: PROCEDURE[v: Handle, f: File.ID] =
    INLINE BEGIN v.rootFileID[tFreePage] ← f; END;
SetVam: PROCEDURE[v: Handle, f: File.ID] =
    INLINE BEGIN v.rootFileID[tVolumeAllocationMap] ← f; END;
SetVfm: PROCEDURE[v: Handle, f: File.ID] =
    INLINE BEGIN v.rootFileID[tVolumeFileMap] ← f; END;

-- Space should really be allocated, not this dribble
FreeVolumePages: PROCEDURE[l: Handle] RETURNS[Volume.PageCount] = INLINE BEGIN RETURN[MAX[LONG[5], l.freePageCount-
    (l.freePageCount/16)]-5]; END;

-- CONSTANTS

descriptorSize: CARDINAL [0..256] = SIZE[Descriptor];          -- compile time check

maxLogicalVolumeLabelLength: CARDINAL = 40;

maxIDRep: RECORD [a, b, c, d, e: WORD] = [177777B, 177777B, 177777B, 177777B, 177777B];

maxID: File.ID = LOOPHOLE[maxIDRep];

maxKey: Key = [maxID, File.lastPageNumber];

nullBoot: Boot.LVBootFiles = ALL[Boot.nullDiskFileID];

nullID: File.ID = File.nullID;

nullInterval: Interval = Interval[nullKey, nullVolumePage, nullKey];

nullIntervals: ARRAY Level OF Interval = ALL[nullInterval];

nullKey: Key = Key[nullID, 0];

nullName: PACKED ARRAY [0..maxLogicalVolumeLabelLength) OF CHARACTER = ALL[0C];
```

```
nullRootFileIDs: ARRAY PilotFileTypes.PilotRootFileType OF File.ID = ALL[nullID];
nullVolumePage: PageNumber = 0;
rootPageNumber: PageNumber = 0;
END.
```

LOG

For earlier log entries see Teak archive version.

Time: February 27, 1980 10:50 AM	*By: Forrest*	*Action: Added OpenVolume, CloseVolume, OpenVolumeAndDeleteTemps, CloseVolumeAndFlushFiles. Sigh.*
Time: March 5, 1980 10:12 AM	*By: Forrest*	*Action: Made OpenVolume return the success boolean. Eliminated op and size from FileAccess Proc*
Time: March 7, 1980 11:38 PM	*By: Forrest*	*Action: Moved FreeVolumePages here from VolumeImplInterface. Changed OpenVolume and VolumeAccess to return status, and defined status. Eliminated below:*

```
-- NOTE: LogicalVolumeOps.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;
```

Time: April 10, 1980 10:36 AM	*By: Knutsen*	*Action: Changed arg of ScavengeVolume from SimpleSpace.handle to Space.handle to reduce compilation dependencies. ScavengeImpl can still use SimpleSpace.handle, which is equivalent.*
Time: April 25, 1980 1:40 PM	*By: Forrest*	*Action: Changes to get around Mesa 6.0 compiler Bug.*
Time: May 30, 1980 3:02 PM	*By: Luniewski*	*Action: {Open Close}Volume = > {Open Close}LogicalVolume to get around a multiple export problem in VolumeImpl.*
Time: June 16, 1980 9:59 AM	*By: Luniewski*	*Action: Added BeginScavenging, EndScavenging.*
Time: June 16, 1980 11:11 AM	*By: McJones*	*Action: Changes for 48-bit processor ids.*
Time: July 22, 1980 4:58 PM	*By: Luniewski*	*Action: Interim addition of ReadOnlyVolume (until made public).*
Time: September 11, 1980 2:44 PM	*By: Luniewski*	*Action: Changes for new logical volume format*

```
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
    DiskChannel USING [Drive],
    Environment USING [wordsPerPage],
    File USING[ID],
    LogicalVolume USING [Handle, LogicalSubvolumeMarker],
    PhysicalVolume USING [ID],
    PhysicalVolumeFormat USING [Handle, PhysicalSubvolumeMarker];

MarkerPage: DEFINITIONS =
BEGIN

-- SubVolumeMarkerPage is set up to occupy a full page, with each half taking half of the page. This allows each part to grow independently. This may not
win much in the long run, but it is cheap ti implement.
SubVolumeMarkerPage: TYPE = MACHINE DEPENDENT RECORD[
    logical: LogicalVolume.LogicalSubvolumeMarker,
     fill1: ARRAY SubVolumeFill1 OF WORD ← NULL,
    physical: PhysicalVolumeFormat.PhysicalSubvolumeMarker,
     fill2: ARRAY SubVolumeFill2 OF WORD ← NULL,
     checksum: CARDINAL ← 0];

CacheKey: TYPE = RECORD[SELECT type:* FROM
    drive =>                [h: DiskChannel.Drive],
    physicalID =>           [id: PhysicalVolume.ID],
    ENDCASE ];

-- Procedures implemented in MarkerPageImpl

CreateMarkerPage: PROCEDURE[pvHandle: PhysicalVolumeFormat.Handle, lvHandle: LogicalVolume.Handle, physicalSubvolumeNumber:
    CARDINAL];

UpdateLogicalMarkerPages: PROCEDURE[lvHandle: LogicalVolume.Handle];

UpdatePhysicalMarkerPages: PROCEDURE[pvHandle: PhysicalVolumeFormat.Handle];

Enter: PROCEDURE[drive: DiskChannel.Drive, physicalID: PhysicalVolume.ID];

EnterMarkerID: PROCEDURE[c: CacheKey, markerID: File.ID];

GetNextPhysicalVolume: PROCEDURE[thisPvID: PhysicalVolume.ID] RETURNS [nextPvID: PhysicalVolume.ID, drive: DiskChannel.Drive];

-- Find is exported because some of Special Volume operations use it.
Find: PROCEDURE[CacheKey] RETURNS [drive: DiskChannel.Drive, physicalID: POINTER TO READONLY PhysicalVolume.ID, markerID:
    POINTER TO READONLY File.ID];

NotFound: ERROR;

Flush: PROCEDURE[CacheKey];

-- Constants
SubVolumeFill1: TYPE = [SIZE[LogicalVolume.LogicalSubvolumeMarker]..Environment.wordsPerPage/2);
SubVolumeFill2: TYPE =
    [Environment.wordsPerPage/2 + SIZE[PhysicalVolumeFormat.PhysicalSubvolumeMarker]..Environment.wordsPerPage-1);

-- Assertion
SubVolumeMarkerSize: CARDINAL[Environment.wordsPerPage..Environment.wordsPerPage] = SIZE[SubVolumeMarkerPage];

END.....

LOG
Time: October 1, 1979  12:20 PM     By: Forrest    Action: Created from old code in volumeImpl.mesa
Time: May 20, 1980  2:50 PM         By: Luniewski          Action: PhysicalVolume = > PhysicalVolumeFormat
Time: July 24, 1980  9:04 AM        By: Luniewski          Action: Added GetNextPhysicalVolume.
```

```
-- File: MiniEthernetDefs.Mesa,  Last Edit: HGM  April 20, 1980  6:23 PM

DIRECTORY
  PupTypes: FROM "PupTypes" USING [PupAddress, PupSocketID, PupType, Pair];

MiniEthernetDefs: DEFINITIONS =
BEGIN

ActivateDriver: PROCEDURE [
  dataBuffer: LONG POINTER, length: CARDINAL,  -- Must be quad word aligned
  iocb: LONG POINTER,  -- 16 word block in first 64k, quad word aligned
  avoidCleanup: BOOLEAN ← FALSE]
  RETURNS [ok: BOOLEAN];

KillDriver: PROCEDURE [avoidCleanup: BOOLEAN ← FALSE];

GetEthernetHostNumber: PROCEDURE RETURNS [CARDINAL];
GetEthernetNetNumber: PROCEDURE RETURNS [CARDINAL];  -- 0 if unknown

DriverNotActive: ERROR;
BufferOverflow: ERROR;

SendPacket: PROCEDURE [
  dest: PupTypes.PupAddress,
  me: PupTypes.PupSocketID,
  type: PupTypes.PupType,
  id: PupTypes.Pair,
  data: LONG POINTER, bytes: CARDINAL];

ReturnPacket: PROCEDURE [
  type: PupTypes.PupType,
  data: LONG POINTER, bytes: CARDINAL];

-- returns -1 if timeout
RecvPacket: PROCEDURE [
  source: LONG POINTER TO PupTypes.PupAddress,
  me: PupTypes.PupSocketID,
  data: LONG POINTER, words: CARDINAL,
  timeout: PROCEDURE RETURNS [BOOLEAN]]
  RETURNS [bytes: CARDINAL, id: PupTypes.Pair, type: PupTypes.PupType];
timedOut: CARDINAL = LAST[CARDINAL];

END.
```

-- LastEdited: August 5, 1980 5:15 PM By: BRD

```
DIRECTORY
    Environment USING [Byte],
    RS232CEnvironment USING [CompletionHandle, PhysicalRecordHandle],
    RS232CFace USING [DeviceStatus, ParamHandle, ParameterOutcome, TransferStatus],
    Zone USING [Base];

MIOCInternalD0: DEFINITIONS =
BEGIN

maxLines: CARDINAL = 3;

-- Interface Definitions

MakeMeANewImpl: PROCEDURE [line: CARDINAL] RETURNS [implHandle: LONG POINTER TO Impl];
MakeMeANewInstance: PROCEDURE [line: CARDINAL] RETURNS [cmdHandle: LONG POINTER TO Cmds, implHandle: LONG POINTER TO
        Impl];

DecrementOnCount: PROCEDURE;
GetCount: PROCEDURE RETURNS [numberOfLines: CARDINAL];
GetCSBPointer: PROCEDURE [line: CARDINAL] RETURNS [csb: LONG POINTER TO ControllerStatusBlock];
IncrementOnCount: PROCEDURE;
IssueChipCommand: PROCEDURE [line: CARDINAL, command: UNSPECIFIED] RETURNS [result: UNSPECIFIED];
LoadTimer: PROCEDURE [line: CARDINAL, timer:Timer, timerMode:TimerMode, count: CARDINAL];
OverlayMicrocode: PROCEDURE [microcode: PROGRAM] RETURNS [BOOLEAN];
ResetRingLatch: PROCEDURE [line: CARDINAL];
StartTransmitter: PROCEDURE  [line: CARDINAL];
SetClocking: PROCEDURE [line: CARDINAL, clocking: ClockingType];
SetLoopback: PROCEDURE [line: CARDINAL, loopback: LoopbackType];
LoadTTYPortRegister: PROCEDURE [lineNumber: CARDINAL, ttyPortLoadWord: TTYPortLoadWord] RETURNS [stat:
        TTYPortStatusWord];

InitializeRS232: PROCEDURE [line: CARDINAL];
InitializePrinter: PROCEDURE [line: CARDINAL, mask: UNSPECIFIED];

X800Init: PROCEDURE [pointer: LONG POINTER] RETURNS [inputStart, outputStart: State];
X850Init: PROCEDURE [pointer: LONG POINTER] RETURNS [inputStart, outputStart: State];
System6Init: PROCEDURE [pointer: LONG POINTER] RETURNS [inputStart, outputStart: State];

RS232CAsyncMicrocodeD0: PROGRAM;
RS232CByteMicrocodeD0: PROGRAM;
RS232CBitMicrocodeD0: PROGRAM;
RS232CTtyMicrocodeD0: PROGRAM;

-- Non-interface definitions

Impl: TYPE = RECORD[
    AbortStatus: PROCEDURE,
    AbortQueue: PROCEDURE [type: QueueType],
    AbortTopIOCB: PROCEDURE [iocb: IOCBPtr],
    Cleanup: PROCEDURE,
    ClearLatchBit: PROCEDURE [type: LatchBitType],
    Get: PROCEDURE [rec: RS232CEnvironment.PhysicalRecordHandle] RETURNS [RS232CEnvironment.CompletionHandle],
    InitializeSyncBlock: PROCEDURE [paramHandle: RS232CFace.ParamHandle],
    Put: PROCEDURE [rec: RS232CEnvironment.PhysicalRecordHandle] RETURNS [RS232CEnvironment.CompletionHandle],
    SetStartStates: PROCEDURE [input: State, output: State],
    StatusWait: PROCEDURE [stat: RS232CFace.DeviceStatus] RETURNS [newstat: RS232CFace.DeviceStatus],
    TransferWait: PROCEDURE [event: RS232CEnvironment.CompletionHandle] RETURNS [byteCount: CARDINAL, status:
        RS232CFace.TransferStatus],
    UpdateStatus: PROCEDURE RETURNS [stat: RS232CFace.DeviceStatus]];

Cmds: TYPE = RECORD[
```

```
    Abort: PROCEDURE [queue: QueueType],
    Cleanup: PROCEDURE,
    ResetLine: PROCEDURE [paramHandle: RS232CFace.ParamHandle] RETURNS [outcome: RS232CFace.ParameterOutcome],
    SetParameters: PROCEDURE [paramHandle: RS232CFace.ParamHandle] RETURNS [outcome: RS232CFace.ParameterOutcome],
    SendBreak: PROCEDURE];

ClockingType: TYPE = {local, external};
LatchBitType: TYPE = {breakDetected, ringHeard, dataLost};
LoopbackType: TYPE = {internal, external};

Timer: TYPE = {rs232cClock, rs232cCommand, printerClock};
TimerMode: TYPE = {interruptOnTerminalCount, programmableOneShot, rateGenerator, squareWaveGenerator,
        softwareTriggerStrobe, hardwareTriggerStrobe};
TimerAddress: TYPE = {unused0, loadCounter0, readCounter0, unused3, unused4, loadCounter2, readCounter2, unused7, unused8,
        loadCounter1, readCounter1, unused11, unused12, writeModeWord, nop, unused15};

TimerModeWord: TYPE = MACHINE DEPENDENT RECORD [
    unused: [0..377B] ← 0,
    timerSelect: Timer,
    readLoadControl: {counterLatching, leastSignificantByteOnly, mostSignificantByteOnly, leastSignificantByteFirst} ←
            leastSignificantByteFirst,
    mode: TimerMode,
    bcdMode: BOOLEAN ← FALSE];

TimerLoadWord: TYPE = MACHINE DEPENDENT RECORD [
    disableTimerGate: BOOLEAN,
    address: TimerAddress,
    unused: [0..7B] ← 0,
    data: [0..377B] ← 0];

TimerStatusWord: TYPE = MACHINE DEPENDENT RECORD [
    busy: BOOLEAN,
    notRead: BOOLEAN,
    notWrite: BOOLEAN,
    notStrobe: BOOLEAN,
    address: TimerAddress,
    data: [0..377B]];

TimerSpeedConstant: TYPE = MACHINE DEPENDENT RECORD [ x1, x16: CARDINAL];

-- The following are constants needed to set the timer chip to generate the appropriate clocks for x16 (asynchronous) and x1
        (synchronous) modes. There values are based on a 1.8432 MHz clock --

bps50: TimerSpeedConstant = [x1: 36864, x16: 2304];
bps75: TimerSpeedConstant = [x1: 24576, x16: 1536];
bps110: TimerSpeedConstant = [x1: 16756, x16: 1047];
bps134p5: TimerSpeedConstant = [x1: 13704, x16: 856];
bps150: TimerSpeedConstant = [x1: 12288, x16: 768];
bps300: TimerSpeedConstant = [x1: 6144, x16: 384];
bps600: TimerSpeedConstant = [x1: 3072, x16: 192];
bps1200: TimerSpeedConstant = [x1: 1536, x16: 96];
bps1800: TimerSpeedConstant = [x1: 1024, x16: 64];
bps2000: TimerSpeedConstant = [x1: 922, x16: 58];
bps2400: TimerSpeedConstant = [x1: 768, x16: 48];
bps3600: TimerSpeedConstant = [x1: 512, x16: 32];
bps4800: TimerSpeedConstant = [x1: 384, x16: 24];
bps7200: TimerSpeedConstant = [x1: 256, x16: 16];
bps9600: TimerSpeedConstant = [x1: 192, x16: 12];
bps19200: TimerSpeedConstant = [x1: 96, x16: 6];

DialerLoadWord: TYPE = MACHINE DEPENDENT RECORD [
    unused: [0..377B],
    callRequest: BOOLEAN,
    digitPresent: BOOLEAN,
    selectLocalTiming: BOOLEAN,
    loopbackEnable: BOOLEAN,
    digit: [0..17B]];

DialerStatusWord: TYPE = MACHINE DEPENDENT RECORD [
```

```
        notDataLineOccupied: BOOLEAN,
        notPresentNextDigit: BOOLEAN,
        notCallOriginationStatus: BOOLEAN,
        notAbandonCallAndRetry: BOOLEAN,
        notDCEPowerIndicator: BOOLEAN,
        ringIndicator: BOOLEAN,
        notPrimaryDataCarrierDetect: BOOLEAN,
        notSecondaryDataCarrierDetect: BOOLEAN,
        notCallRequest: BOOLEAN,
        notDigitPresent: BOOLEAN,
        selectLocalTiming: BOOLEAN,
        loopbackEnable: BOOLEAN,
        digit: [0..17B]];

ControllerStatusBlock: TYPE = MACHINE DEPENDENT RECORD [
        printerWakeup: UNSPECIFIED,
        notUsedYet1: ARRAY [1..3] OF UNSPECIFIED,
        pollerCommand: PollerCommandData,
        pollerResults: PollerCommandData,
        initialTimeoutCount: INTEGER,
        currentTimeoutCount: INTEGER,
        outputChain: IOCBPtr,
        inputChain: IOCBPtr,
        wakeup: UNSPECIFIED,
        notUsedYet2: UNSPECIFIED,
        reg5Data: WriteReg5Data,
        reg3Data: WriteReg3Data,
        notUsedYet4: [0..377B],
        status: CSBStatus,
        tables: MicrocodeLongPointer];

nullTimer: INTEGER = -1; -- Value to disable microcode timer --

CSBStatus: TYPE = MACHINE DEPENDENT RECORD [
        notUsedYet: [0..37B] ← 0,
        synSeen: BOOLEAN ← FALSE,
        dataLost: BOOLEAN ← FALSE,
        event: BOOLEAN ← FALSE];

MicrocodeLongPointer: TYPE = MACHINE DEPENDENT RECORD [
        lowHalf: UNSPECIFIED,
        highHalf: [0..377B],
        highHalfPlusOne: [0..377B]];



IOCB: TYPE = MACHINE DEPENDENT RECORD [
        next: IOCBPtr,
        checksum: UNSPECIFIED,
        currentState: State,
        specialCharacter: Environment.Byte,
        status: ReadReg1,
        buffer: LONG POINTER TO PACKED ARRAY OF Environment.Byte,
        maxCount: CARDINAL,
        offset: CARDINAL,
        start: CARDINAL,
        synch: CONDITION,
        marked: BOOLEAN,
        completed: BOOLEAN,
        type: QueueType,
        transferCompletion: RS232CFace.TransferStatus,
        fill: [0..000777B],
        unused: ARRAY [12..15] OF UNSPECIFIED];

QueueType: TYPE = {input, output};

IOCBPtr: TYPE = Zone.Base RELATIVE POINTER TO IOCB;
LongIOCBPtr: TYPE = LONG POINTER TO IOCB;
```

```
CharTable: TYPE = ARRAY [0..255] OF CARDINAL;
FillCharTable: TYPE = ARRAY [20B..45B] OF PACKED ARRAY [0..1] OF Environment.Byte;
StateTable: TYPE = ARRAY [20B..45B] OF ARRAY [0..17B] OF InputEvent;
State: TYPE = [0..377B];

InputEvent: TYPE = MACHINE DEPENDENT RECORD [
    new: State,
    specialCharacter: Environment.Byte ← 0,
    fillOK: BOOLEAN ← FALSE,
    back: BOOLEAN ← FALSE,
    save: BOOLEAN ← FALSE,
    reload: BOOLEAN ← FALSE,
    crc: BOOLEAN ← FALSE,
    zero: BOOLEAN ← FALSE,
    send: BOOLEAN ← FALSE,
    sendSpecial: BOOLEAN ← FALSE,
    sendCRC: BOOLEAN ← FALSE,
    synSeen: BOOLEAN ← FALSE,
    end: BOOLEAN ← FALSE,
    always: BOOLEAN ← TRUE,
    unused: [0..17B] ← 0];

OutputEvent: TYPE = MACHINE DEPENDENT RECORD [
    new: State,
    specialCharacter: Environment.Byte ← 0,
    fillOK: BOOLEAN ← FALSE,
    back: BOOLEAN ← FALSE,
    save: BOOLEAN ← FALSE,
    reload: BOOLEAN ← FALSE,
    crc: BOOLEAN ← FALSE,
    zero: BOOLEAN ← FALSE,
    send: BOOLEAN ← FALSE,
    sendSpecial: BOOLEAN ← FALSE,
    sendCRC: BOOLEAN ← FALSE,
    synSeen: BOOLEAN ← FALSE,
    end: BOOLEAN ← FALSE,
    always: BOOLEAN ← TRUE,
    unused: [0..17B] ← 0];

Tables: TYPE = MACHINE DEPENDENT RECORD [                    -- Total size = 1238 --
    crcTable: ARRAY [0..255] OF UNSPECIFIED,                 -- [0B..377B] --
    charTable: CharTable,                                    -- [400B..777B] --
    stateTables: StateTable,                                 -- [1000B..2277B] --
    fillCharTable: FillCharTable];                           -- [2300B..2326B] --

LoadCommCommandWord: TYPE = MACHINE DEPENDENT RECORD [
    channel: Channel,
    operation: {writeChip, unused1, readCommand, interruptAcknowledge, writeData, unused5, readData, unused7},
    unused: [0..17B],
    data: [0..377B]];

WriteReg0: TYPE = MACHINE DEPENDENT RECORD [
    command: PollerCommand,
    data: WriteReg0Data];

WriteReg0Data: TYPE = MACHINE DEPENDENT RECORD [
    commandCRC: {null, resetReceiverCRC, resetTransmitterCRC, resetTransmitterUnderrun},
    commandChip: {null, sendAbort, resetExternalStatus, channelReset, enableReceiverNext, resetTransmitterInterrupt, errorReset,
        returnFromInterrupt},
    registerSelect: [0..7]];

WriteReg1: TYPE = MACHINE DEPENDENT RECORD [
```

```
        command: PollerCommand,
        data: WriteReg1Data];

    WriteReg1Data: TYPE = MACHINE DEPENDENT RECORD [
        waitReadyEnable: BOOLEAN,
        readyFunctionEnable: BOOLEAN,
        waitReadyonRT: BOOLEAN ,
        receiverInterrupts: {disabled, firstCharacterOnly, allWithParity, allWithoutParity},
        statusAffectsVector: BOOLEAN,
        transmitterInterruptEnable: BOOLEAN,
        externalInterruptEnable: BOOLEAN];

    WriteReg2: TYPE = MACHINE DEPENDENT RECORD [
        command: PollerCommand,
        interruptVector: Environment.Byte];

    WriteReg3: TYPE = MACHINE DEPENDENT RECORD [
        command: PollerCommand,
        data: WriteReg3Data];

    WriteReg3Data: TYPE = MACHINE DEPENDENT RECORD [
        characterLength: CharacterLength,
        autoEnable: BOOLEAN,
        enterHuntPhase: BOOLEAN,
        receiverCRCEnable: BOOLEAN,
        addressSearch: BOOLEAN,
        syncLoadInhibit: BOOLEAN,
        receiverEnable: BOOLEAN];

    WriteReg4: TYPE = MACHINE DEPENDENT RECORD [
        command: PollerCommand,
        data: WriteReg4Data];

    WriteReg4Data: TYPE = MACHINE DEPENDENT RECORD [
        clockMode: {times1, times16, times32, times64},
        syncLength: {singleSync, doubleSync, sdlcSync, externalSync},
        stopBits: {none, one, oneAndHalf, two},
        parity: {none, odd, unused, even}];

    WriteReg5: TYPE = MACHINE DEPENDENT RECORD [
        command: PollerCommand,
        data: WriteReg5Data];

    WriteReg5Data: TYPE = MACHINE DEPENDENT RECORD [
        dtr: BOOLEAN,
        characterLength: CharacterLength,
        sendBreak: BOOLEAN,
        transmitterEnable: BOOLEAN,
        crcType: {crcSDLC, crc16},
        rts: BOOLEAN,
        transmitterCRCEnable: BOOLEAN];

    WriteReg6: TYPE = MACHINE DEPENDENT RECORD [
        command: PollerCommand,
        syncChar: Environment.Byte];

    WriteReg7: TYPE = MACHINE DEPENDENT RECORD [
        command: PollerCommand,
        syncChar: Environment.Byte];

    ReadReg0: TYPE = MACHINE DEPENDENT RECORD [
        commChipActive: BOOLEAN,
        ringIndicatorLatch: BOOLEAN,
        notDataSetReady: BOOLEAN,
        notPrimaryClearToSend: BOOLEAN,
        notSecondaryClearToSend: BOOLEAN,
        notPrimaryDataCarrierDetect: BOOLEAN,
        notSecondaryDataCarrierDetect: BOOLEAN,
```

```
    selectLocalTiming: BOOLEAN,
    break: BOOLEAN,
    underrunEOM: BOOLEAN,
    clearToSend: BOOLEAN,
    syncHunt: BOOLEAN,
    carrierDetect: BOOLEAN,
    transmitterBufferEmpty: BOOLEAN,
    interruptPending: BOOLEAN,
    receiverHasCharacters: BOOLEAN];

ReadReg1: TYPE = MACHINE DEPENDENT RECORD [
    unused: [0..377B],
    endOfFrame: BOOLEAN,
    crcFramingError: BOOLEAN,
    receiverOverrun: BOOLEAN,
    parityError: BOOLEAN,
    residueBits: [0..7B],
    allSent: BOOLEAN];

CharacterLength: TYPE = {lengthIs5Bits, lengthIs7Bits, lengthIs6Bits, lengthIs8Bits};

ResetStatusCommand: TYPE = MACHINE DEPENDENT RECORD [
    commandPart: PollerCommand,
    dataPart: CSBStatus ← [0,FALSE,FALSE,FALSE]];

PollerCommand: TYPE = MACHINE DEPENDENT RECORD [
    channel: Channel,
    command: CommandCode,
    register: Register];
PollerCommandData: TYPE = MACHINE DEPENDENT RECORD [
    commandPart: PollerCommand,
    dataPart: Environment.Byte];
Channel: TYPE = {channelA, channelB};
CommandCode: TYPE = {noCommand, startTransmitter, writeChip, unused3, nop, initializeOverlay, readChip, resetStatus,
        loadRegister5};
Register: TYPE = [0..7B];

TTYPortLoadWord: TYPE = MACHINE DEPENDENT RECORD [ -- to send cmd/data to 8251 (TTYPort)
    unused: [0..17B],
    ptA0CTSToTTYPort: BOOLEAN, -- this is PTClearToSend to device; reflects CTS in lineRecord
    cmdSelect: {unused0, writeTTYPort, readTTYPort, unused3, unused4, writeUsart, readUsart, unused7},
    dataBits: [0..377B] ← 0];

TTYPortStatusWord: TYPE = MACHINE DEPENDENT RECORD [ -- to receive status/data from 8251 (TTYPort)
    busy: BOOLEAN, -- this is PTCS (chip select)
    ptA0: BOOLEAN, -- this is PTClearToSend to device
    ptA1: BOOLEAN, -- this is Command/Data' to Usart
    notPtRD: BOOLEAN,
    notPtWR: BOOLEAN,
    notPtRTSFromTTYPort: BOOLEAN, -- RTS comes in on pin 16 on 16-pin dip, pin 4 on 25-pin; this with TxEnable allows
            transmission to device
    notPtCTSToTTYPort: BOOLEAN, -- this is just ptA0' to device
    notPtDTRFromTTYPort: BOOLEAN, -- DTR comes in on pin 14 on 16-pin dip, pin 20 on 25-pin
    dataBits: SELECT OVERLAID * FROM
        readTTYPort => [char: CHARACTER],
        readUsart => [stat: UsartStatus],
        ENDCASE];

UsartStatus: TYPE = MACHINE DEPENDENT RECORD [
    -- TTYPortStatusWord.dataBits on readUsart
    dsr: BOOLEAN, -- on if DSR' is 0, on MIOC means something is plugged in
    breakDetect: BOOLEAN,
    framingError: BOOLEAN,
    overrunError: BOOLEAN,
    parityError: BOOLEAN,
    txmtEmpty: BOOLEAN, -- output buffer empty
    rcvReady: BOOLEAN, -- input buffer full (masked/held Reset by RxEN off) = Wakeup
    txmtRegReady: BOOLEAN -- output buffer empty (TxRdy pin = Wakeup = empty + TxEN + CTS' low) -- ];

END. -- MIOCInternalD0
```

LOG

Time: September 25,,1979 1:27 PM By: Bill Danielson Action: Created file
Time: January 9, 1980 4:15 PM By: Bill Danielson Action: Changed CSB to allow microcode timeouts for asynchronous mode and byte-synchrnous fill characters.
Time: January 14, 1980 1:45 PM By: Bill Danielson Action: Modified for .bcd microcode overlays.
Time: January 15, 1980 2:21 PM By: Bill Danielson Action: Combined MIOCCommDefs, MIOCTimerDefs, and RS232CTables into MIOCInternal.
Time: January 17, 1980 1:11 PM By: Bill Danielson Action: Modified for new face.
Time: January 21, 1980 6:30 PM By: Bill Danielson Action: New status code.
Time: January 22, 1980 2:28 PM By: Bill Danielson Action: Mods for RS232CEnvironment.
Time: January 31, 1980 11:12 AM By: Bill Danielson Action: Added X850Init for Xerox 850 tables.
Time: February 27, 1980 11:25 AM By: Bill Danielson Action: Added tty microcode variant.
Time: March 19. 1980 10:20 AM By: Bill Danielson Action: Modified state table entries for intraframe fill code.
Time: March 24, 1980 11:56 AM By: Bill Danielson Action: Added Overlay microcode to allow microcode reload on debugger world swaps.
Time: April 17, 1980 5:07 PM By: Bill Danielson Action: Changed CSB.tables to be a microcode type of LONG POINTER.
Time: April 23, 1980 2:13 PM By: Bill Danielson Action: Fixed ODD parity.
Time: May 8, 1980 1:59 PM By: Bill Danielson Action: Added stuff for Printer.
Time: July 22, 1980 2:24 PM By: Bill Danielson Action: Added Reg5Data to CSB.
Time: July 22, 1980 3:33 PM By: Bill Danielson Action: TTY port definitions added.
Time: August 5, 1980 11:55 AM By: Bill Danielson Action: Removed Suspend and Restart.

```
-- MiscAlpha.mesa   Last edited by Johnsson on July 21, 1980  12:00 PM

MiscAlpha: DEFINITIONS =
  BEGIN
  alpha: TYPE = [0..400B);

  aASSOC: alpha = 0B;
  aSETF: alpha = 1B;
  aREADRAM: alpha = 2B;
  aLOADRAMJ: alpha = 3B;
  a4: alpha = 4B; -- unused
  aINPUT: alpha = 5B;
  aOUTPUT: alpha = 6B;
  aCHKSUM: alpha = 7B;
  aSETMP: alpha = 10B;
  aRCLK: alpha = 11B;
  aRPRINTER: alpha = 12B;
  aWPRINTER: alpha = 13B;
  aBANDBLT: alpha = 14B;
  aTEXTBLT: alpha = 15B;
  aGETF: alpha = 16B;
  a17: alpha = 17B; -- unused

  -- Floating Point (20B-57B are reserved)
  aFADD: alpha = 20B;
  aFSUB: alpha = 21B;
  aFMUL: alpha = 22B;
  aFDIV: alpha = 23B;
  aFCOMP: alpha = 24B;
  aFIX: alpha = 25B;
  aFLOAT: alpha = 26B;
  aFIXI: alpha = 27B;
  aFIXC: alpha = 30B;
  aFSTICKY: alpha = 31B;
  aFREM: alpha = 32B;
  aROUND: alpha = 33B;
  aROUNDI: alpha = 34B;
  aROUNDC: alpha = 35B;
  END...
```

```
MiscPrograms: DEFINITIONS =
BEGIN
InitializeHeap: PROCEDURE;
InitializeResidentHeap: PROCEDURE;
InitializeStream: PROCEDURE;
InitializeUtilities: PROCEDURE;
InitializeZone: PROCEDURE;
END.
```

LOG

Time: April 25, 1979 4:00 PM	*By: Lauer*	*Action: Create file*
Time: April 30, 1979 9:18 PM	*By: Lauer*	*Action: Add ResidentHeapImpl*
Time: September 6, 1979 8:43 PM	*By: Ladner*	*Action: Add PPDataImpl*
Time: April 14, 1980 8:48 AM	*By: Knutsen*	*Action: Change PROGRAMs to Initialize**
Time: June 11, 1980 9:07 AM	*By: McJones*	*Action: Add InitializeHeap*
Time: August 29, 1980 5:15 PM	*By: Forrest*	*Action: Add Kill PPDataImpl*

```
-- generated by OpDefsGenerator 12-Jan-79 17:00
-- Edited by Sandman on June 30, 1980  4:41 PM

Mopcodes: DEFINITIONS =
  BEGIN
  op: TYPE = [0..400B);

  zNOOP: op = 0B;
  zME: op = 1B;
  zMRE: op = 2B;
  zMXW: op = 3B;
  zMXD: op = 4B;
  zNOTIFY: op = 5B;
  zBCAST: op = 6B;
  zREQUEUE: op = 7B;
  zLL0: op = 10B;
  zLL1: op = 11B;
  zLL2: op = 12B;
  zLL3: op = 13B;
  zLL4: op = 14B;
  zLL5: op = 15B;
  zLL6: op = 16B;
  zLL7: op = 17B;
  zLLB: op = 20B;
  zLLDB: op = 21B;
  zSL0: op = 22B;
  zSL1: op = 23B;
  zSL2: op = 24B;
  zSL3: op = 25B;
  zSL4: op = 26B;
  zSL5: op = 27B;
  zSL6: op = 30B;
  zSL7: op = 31B;
  zSLB: op = 32B;
  zPL0: op = 33B;
  zPL1: op = 34B;
  zPL2: op = 35B;
  zPL3: op = 36B;
  zLG0: op = 37B;
  zLG1: op = 40B;
  zLG2: op = 41B;
  zLG3: op = 42B;
  zLG4: op = 43B;
  zLG5: op = 44B;
  zLG6: op = 45B;
  zLG7: op = 46B;
  zLGB: op = 47B;
  zLGDB: op = 50B;
  zSG0: op = 51B;
  zSG1: op = 52B;
  zSG2: op = 53B;
  zSG3: op = 54B;
  zSGB: op = 55B;
  zLI0: op = 56B;
  zLI1: op = 57B;
  zLI2: op = 60B;
  zLI3: op = 61B;
  zLI4: op = 62B;
  zLI5: op = 63B;
  zLI6: op = 64B;
  zLIN1: op = 65B;
  zLINI: op = 66B;
  zLIB: op = 67B;
  zLIW: op = 70B;
  zLINB: op = 71B;
  zLADRB: op = 72B;
  zGADRB: op = 73B;
  zLCO: op = 74B;
  zR0: op = 100B;
  zR1: op = 101B;
  zR2: op = 102B;
  zR3: op = 103B;
  zR4: op = 104B;
  zRB: op = 105B;
  zW0: op = 106B;
  zW1: op = 107B;
  zW2: op = 110B;
  zWB: op = 111B;
```

```
  zRF: op = 112B;
  zWF: op = 113B;
  zRDB: op = 114B;
  zRD0: op = 115B;
  zWDB: op = 116B;
  zWD0: op = 117B;
  zRSTR: op = 120B;
  zWSTR: op = 121B;
  zRXLP: op = 122B;
  zWXLP: op = 123B;
  zRILP: op = 124B;
  zRIGP: op = 125B;
  zWILP: op = 126B;
  zRIL0: op = 127B;
  zWS0: op = 130B;
  zWSB: op = 131B;
  zWSF: op = 132B;
  zWSDB: op = 133B;
  zRFC: op = 134B;
  zRFS: op = 135B;
  zWFS: op = 136B;
  zRBL: op = 137B;
  zWBL: op = 140B;
  zRDBL: op = 141B;
  zWDBL: op = 142B;
  zRXLPL: op = 143B;
  zWXLPL: op = 144B;
  zRXGPL: op = 145B;
  zWXGPL: op = 146B;
  zRILPL: op = 147B;
  zWILPL: op = 150B;
  zRIGPL: op = 151B;
  zWIGPL: op = 152B;
  zRSTRL: op = 153B;
  zWSTRL: op = 154B;
  zRFL: op = 155B;
  zWFL: op = 156B;
  zRFSL: op = 157B;
  zWFSL: op = 160B;
  zLP: op = 161B;
  zSLDB: op = 162B;
  zSGDB: op = 163B;
  zPUSH: op = 164B;
  zPOP: op = 165B;
  zEXCH: op = 166B;
  zLINKB: op = 167B;
  zDUP: op = 170B;
  zNILCK: op = 171B;
  zNILCKL: op = 172B;
  zBNDCK: op = 173B;
  zJ2: op = 200B;
  zJ3: op = 201B;
  zJ4: op = 202B;
  zJ5: op = 203B;
  zJ6: op = 204B;
  zJ7: op = 205B;
  zJ8: op = 206B;
  zJ9: op = 207B;
  zJB: op = 210B;
  zJW: op = 211B;
  zJEQ2: op = 212B;
  zJEQ3: op = 213B;
  zJEQ4: op = 214B;
  zJEQ5: op = 215B;
  zJEQ6: op = 216B;
  zJEQ7: op = 217B;
  zJEQ8: op = 220B;
  zJEQ9: op = 221B;
  zJEQB: op = 222B;
  zJNE2: op = 223B;
  zJNE3: op = 224B;
  zJNE4: op = 225B;
  zJNE5: op = 226B;
  zJNE6: op = 227B;
  zJNE7: op = 230B;
  zJNE8: op = 231B;
  zJNE9: op = 232B;
  zJNEB: op = 233B;
```

```
  zJLB: op = 234B;
  zJGEB: op = 235B;
  zJGB: op = 236B;
  zJLEB: op = 237B;
  zJULB: op = 240B;
  zJUGEB: op = 241B;
  zJUGB: op = 242B;
  zJULEB: op = 243B;
  zJZEQB: op = 244B;
  zJZNEB: op = 245B;
  zJIB: op = 246B;
  zJIW: op = 247B;
  zADD: op = 250B;
  zSUB: op = 251B;
  zMUL: op = 252B;
  zDBL: op = 253B;
  zDIV: op = 254B;
  zLDIV: op = 255B;
  zNEG: op = 256B;
  zINC: op = 257B;
  zAND: op = 260B;
  zOR: op = 261B;
  zXOR: op = 262B;
  zSHIFT: op = 263B;
  zDADD: op = 264B;
  zDSUB: op = 265B;
  zDCOMP: op = 266B;
  zDUCOMP: op = 267B;
  zADD01: op = 270B;
  zEFC0: op = 300B;
  zEFC1: op = 301B;
  zEFC2: op = 302B;
  zEFC3: op = 303B;
  zEFC4: op = 304B;
  zEFC5: op = 305B;
  zEFC6: op = 306B;
  zEFC7: op = 307B;
  zEFC8: op = 310B;
  zEFC9: op = 311B;
  zEFC10: op = 312B;
  zEFC11: op = 313B;
  zEFC12: op = 314B;
  zEFC13: op = 315B;
  zEFC14: op = 316B;
  zEFC15: op = 317B;
  zEFCB: op = 320B;
  zLFC1: op = 321B;
  zLFC2: op = 322B;
  zLFC3: op = 323B;
  zLFC4: op = 324B;
  zLFC5: op = 325B;
  zLFC6: op = 326B;
  zLFC7: op = 327B;
  zLFC8: op = 330B;
  zLFC9: op = 331B;
  zLFC10: op = 332B;
  zLFC11: op = 333B;
  zLFC12: op = 334B;
  zLFC13: op = 335B;
  zLFC14: op = 336B;
  zLFC15: op = 337B;
  zLFC16: op = 340B;
  zLFCB: op = 341B;
  zSFC: op = 342B;
  zRET: op = 343B;
  zLLKB: op = 344B;
  zPORTO: op = 345B;
  zPORTI: op = 346B;
  zKFCB: op = 347B;
  zDESCB: op = 350B;
  zDESCBS: op = 351B;
  zBLT: op = 352B;
  zBLTL: op = 353B;
  zBLTC: op = 354B;
  zBLTCL: op = 355B;
  zALLOC: op = 356B;
  zFREE: op = 357B;
  zIWDC: op = 360B;
```

```
  zDWDC: op = 361B;
  zSTOP: op = 362B;
  zCATCH: op = 363B;
  zMISC: op = 364B;
  zBITBLT: op = 365B;
  zSTARTIO: op = 366B;
  zJRAM: op = 367B;
  zDST: op = 370B;
  zLST: op = 371B;
  zLSTF: op = 372B;
  zWR: op = 374B;
  zRR: op = 375B;
  zBRK: op = 376B;
  END...
```

```
-- This interface defines operations for managing main storage (real memory).

-- Some MStore procedures are involved in recovering from the frame heap becoming exhausted. Because of this, invoking these
   procedures must not cause any frame allocations. This means that these procedures (and any that they call) must be INLINEs or
   coroutines. Since there is no such thing as an ENTRY coroutine, it must be simulated by an ENTRY INLINE procedure (which acquires
   the monitor lock) which itself calls the coroutine. To have an ENTRY INLINE PROCEDURE, the monitor lock must be available in the
   DEFINITIONS module. To allow a coroutine to be called as a public procedure, the procedure descriptor must be bundled into a record
   to force it to be a procedure variable.

DIRECTORY
    PageMap: FROM "PageMap" USING [Flags],
    VM: FROM "VM" USING [Interval, PageCount, PageNumber];

MStore: DEFINITIONS
    LOCKS mStoreLock =

BEGIN OPEN VM;

Allocate: PROCEDURE [interval: Interval];
    -- Allocate and map real memory to all of the specified interval as possible

AllocateIfFree: ENTRY PROCEDURE [interval: Interval] RETURNS [countAllocated: PageCount] = INLINE
    -- Allocates and maps real memory to as much of the specified interval as possible
    -- Guaranteed not to do an ALLOC from the frame heap.
    BEGIN RETURN[allocateIfFreeInternal[interval]] END;

AwaitBelowThreshold: PROCEDURE RETURNS [newCycle: BOOLEAN];
    -- Returns when the amount of free and promised real memory is less than the current threshold setting.
    -- newCycle is TRUE if, when AwaitBelowThreshold was called, the amount of memory was greater/equal than threshold. newCycle
       is FALSE if, when AwaitBelowThreshold was called, the amount of memory was already less than threshold.

Deallocate: ENTRY PROCEDURE [interval: Interval, promised: BOOLEAN] = INLINE
    -- Frees any real memory currently mapped to pages of the specified virtual memory interval. The interval may have unmapped
       holes.
    -- Guaranteed not to do an ALLOC from the frame heap.
    BEGIN deallocateInternal[interval, promised] END;

GetState: ENTRY PROCEDURE [page: PageNumber, writeProtected: BOOLEAN] RETURNS [flags: PageMap.Flags, vacant: BOOLEAN] =
                                                                    INLINE
    -- (Since there is no machine instruction which just reads the page flags, you must know whether it is acceptable for the page to be
       (at least temporarily) write protected. Sorry!)
    -- Guaranteed not to do an ALLOC from the frame heap.
    BEGIN [flags, vacant] ← getStateInternal[page, writeProtected] END;

Promise: PROCEDURE [count: PageCount];
    -- Increases the amount of promised real memory by count pages

Relocate: PROCEDURE [interval: Interval, pageDest: PageNumber, flagsKeep, flagsAdd: PageMap.Flags]
                RETURNS [flags: PageMap.Flags, anyVacant: BOOLEAN];
    -- For each page in interval, clear those flags not in flagsKeep (a bit mask), then set those flags in flagsAdd, and then relocate the
       (real) pages in interval to their corresponding spots in the interval [pageDest, interval.count].
    -- 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.

SetThreshold: PROCEDURE [count: PageCount];
    -- Sets the threshold value used by AwaitBelowThreshold


mStoreLock: PRIVATE MONITORLOCK;

AllocateIfFreeInternal: PRIVATE TYPE = RECORD[
    proc: PROCEDURE [interval: Interval] RETURNS [countAllocated: PageCount]];
allocateIfFreeInternal: PRIVATE AllocateIfFreeInternal;

DeallocateInternal: PRIVATE TYPE = RECORD[
    proc: PROCEDURE [interval: Interval, promised: BOOLEAN]];
deallocateInternal: PRIVATE DeallocateInternal;

GetStateInternal: PRIVATE TYPE = RECORD[
    proc: PROCEDURE [page: PageNumber, writeProtected: BOOLEAN] RETURNS [flags: PageMap.Flags, vacant: BOOLEAN]];
```

```
getStateInternal: PRIVATE GetStateInternal;

END.
```

LOG

Time: March 1978	*By: McJones*	*Action: Create file*
Time: September 5, 1978 3:40 PM	*By: Redell*	*Action: Add anyVacant result to Relocate*
Time: September 26, 1979 3:32 PM	*By: Knutsen*	*Action: Add GetState, documentation*
Time: October 25, 1979 1:32 PM	*By: McJones*	*AR2449: Add Recover (; add AllocateIfFree)*
Time: November 8, 1979 11:56 AM	*By: McJones*	*AR2768: Allocate must not wait in fixed frame*
Time: March 20, 1980 9:35 AM	*By: Knutsen*	*Action: Make Deallocate, GetState ENTRY INLINEs. Added newCycle to AwaitBelowThreshold. Recover moved to StoragePrograms.*

```
DIRECTORY
    KeyStations USING [DownUp, M1, M2, M3];

MouseFace: DEFINITIONS =

BEGIN

-- Processor-independent interface to the mouse (position and buttons).

-- (For historical reasons, the mouse buttons are also available through KeyboardFace.keyboard.)

position: READONLY LONG POINTER TO READONLY Point;
SetPosition: PROCEDURE [Point];

buttons: READONLY LONG POINTER TO READONLY Buttons;

Buttons: TYPE = PACKED ARRAY ButtonName OF DownUp;

Point: TYPE = RECORD [x, y: INTEGER];

ButtonName: TYPE = MACHINE DEPENDENT {
    Mouse1(KeyStations.M1), Mouse3(KeyStations.M3), Mouse2(KeyStations.M2)};

DownUp: TYPE = KeyStations.DownUp;

END.
```

LOG

Time: July 29, 1980 10:57 AM By: McJones Action: Created file from DisplayFace

```
-- NetworkStream.mesa (last edited by: Garlick/BLyon on: October 10, 1980 4:46 PM)

DIRECTORY
  Stream USING [Handle, SubSequenceType],
  System USING [NetworkAddress];

NetworkStream: DEFINITIONS =
BEGIN
-- definitions

-- various types and constants used by Network Stream clients.
WaitTime: TYPE = LONG CARDINAL; -- msecs
defaultWaitTime: WaitTime = 60000; -- msecs
SuspendReason: TYPE = {
  notSuspended, transmissionTimeout, noRouteToDestination, remoteReject};
FailureReason: TYPE = {timeout, noRouteToDestination, -- all media
    -- the following apply only to circuit-oreinted networks (e.g. phone network)
    noAnswerOrBusy, -- auto-dial case only
    noTranslationForDestination, -- no phone number for this destination
    circuitInUse, -- being used to talk to another destination
    circuitNotReady, -- dial the phone or connect modems (non-auto-dial case)
    noDialingHardware,
    dialerHardwareProblem};
ListenerHandle: TYPE [2];
uniqueNetworkAddr: READONLY System.NetworkAddress;
ConnectionID: TYPE = RECORD [WORD];
uniqueConnID: ConnectionID = [0];
unknownConnID: ConnectionID = [0];

-- interface

-- exported by NetworkStreamMgr
-- procedures
-- This procedure creates a network stream to the specified remote address.
Create: PROCEDURE [remote: System.NetworkAddress, timeout: WaitTime]
  RETURNS [Stream.Handle];
-- This procedure creates the sequenced packet transducer with all its parameters
CreateTransducer: PROCEDURE [local, remote: System.NetworkAddress, localConnID, remoteConnID: ConnectionID,
  activelyEstablish: BOOLEAN, timeout: WaitTime] RETURNS [Stream.Handle];
AssignNetworkAddress: PROCEDURE RETURNS [System.NetworkAddress];
FindAddresses: PROCEDURE [sH: Stream.Handle] RETURNS [local, remote: System.NetworkAddress];
SetWaitTime: PROCEDURE [sH: Stream.Handle, time: WaitTime];
CreateListener: PROCEDURE [addr: System.NetworkAddress] RETURNS [ListenerHandle];
DeleteListener: PROCEDURE [listenerH: ListenerHandle];
Listen: PROCEDURE [listenerH: ListenerHandle, listenTimeout, streamTimeout: WaitTime] RETURNS [Stream.Handle];
-- errors and signals
ConnectionSuspended: ERROR [why: SuspendReason];
ConnectionFailed: SIGNAL [why: FailureReason];
IllegalAddress: ERROR; -- illegal network address for CreateListener
AttentionTimeout: ERROR; -- temporary until Process.Abort works

-- optional close protocol using subsequence types
-- definitions
CloseStatus: TYPE = {good, noReply, incomplete};
closeSST: Stream.SubSequenceType = 254;
closeReplySST: Stream.SubSequenceType = 255;
-- interface
-- exported by NetworkStreamMgr
-- procedures
Close: PROCEDURE [sH: Stream.Handle] RETURNS [CloseStatus];
CloseReply: PROCEDURE [sH: Stream.Handle] RETURNS [CloseStatus];

END.

LOG

(trimmed to Amargosa)
Time: August 6, 1980 10:21 AM  By: Garlick  Action: Added several FailureReason's for circuit-oriented networks.
Time: October 10, 1980 4:47 PM  By: Garlick  Action: Added FailureReason noAnswerOrBusy.
```

```
-- Function: The internal definitions module for the Pliot Network Streams.

DIRECTORY
  Stream USING [Object],
  PacketStream USING [Handle];

NetworkStreamInternal: DEFINITIONS =
BEGIN

--definitions

-- used by the implementation modules for the Network Streams
ControlHandle: TYPE = POINTER TO ControlObject;
ControlObject: TYPE = RECORD [
  streamObject: Stream.Object,
  psH: PacketStream.Handle];

-- states for optional close protocol using subsequence types
CloseState: TYPE = {
  sendClose, waitCloseReply, sendCloseReply, waitCloseReplyReply, closed};

END.

LOG

Time: January 26, 1980  2:37 PM  By: Dalal  Action: created file.
Time: January 26, 1980  2:37 PM  By: Dalal  Action: split SPPInternal into two.
```

```
-- Function: The definitions module for buffer/queue/packet length and start/stop.

DIRECTORY
 BufferDefs USING [OisBuffer, SppBuffer, Queue, QueueObject, BufferPool],
 OISCPTypes USING [OisAddress, OisHostID, OisNetID, OisSocketID];

OISCPDefs: DEFINITIONS =
BEGIN

-- TYPEs and definitions

-- Copied things from OISCPTypes.
OisAddress: TYPE = OISCPTypes.OisAddress;
OisNetID: TYPE = OISCPTypes.OisNetID;
OisHostID: TYPE = OISCPTypes.OisHostID;
OisSocketID: TYPE = OISCPTypes.OisSocketID;

-- Copied things from BufferDefs.
OisBuffer: TYPE = BufferDefs.OisBuffer;
SppBuffer: TYPE = BufferDefs.SppBuffer;
Queue: TYPE = BufferDefs.Queue;
QueueObject: TYPE = BufferDefs.QueueObject;
BufferPool: TYPE = BufferDefs.BufferPool;

-- interface

-- Buffer/Queue procedures from the queue package modules
EnqueueOis: PROCEDURE [q: Queue, b: OisBuffer];
DequeueOis: PROCEDURE [q: Queue] RETURNS [OisBuffer];
ExtractOisFromQueue: PROCEDURE [q: Queue, b: OisBuffer] RETURNS [OisBuffer];
GetFreeOisBuffer: PROCEDURE RETURNS [OisBuffer];
ReturnFreeOisBuffer: PROCEDURE [b: OisBuffer];
GetOisBufferFromPool: PROCEDURE [pool: BufferPool] RETURNS [OisBuffer];
GetFreeSendOisBufferFromPool: PROCEDURE [pool: BufferPool] RETURNS [OisBuffer];
GetFreeReceiveOisBufferFromPool: PROCEDURE [pool: BufferPool] RETURNS [OisBuffer];
MaybeGetOisBufferFromPool: PROCEDURE [pool: BufferPool] RETURNS [OisBuffer];
MaybeGetFreeSendOisBufferFromPool: PROCEDURE [pool: BufferPool] RETURNS [OisBuffer];
MaybeGetFreeReceiveOisBufferFromPool: PROCEDURE [pool: BufferPool] RETURNS [OisBuffer];
ReturnOisBufferToPool: PROCEDURE [pool: BufferPool, b: OisBuffer];
ReturnSendOisBufferToPool: PROCEDURE [pool: BufferPool, b: OisBuffer];
ReturnReceiveOisBufferToPool: PROCEDURE [pool: BufferPool, b: OisBuffer];
EnqueueSpp: PROCEDURE [q: Queue, b: SppBuffer];
DequeueSpp: PROCEDURE [q: Queue] RETURNS [SppBuffer];
ExtractSppFromQueue: PROCEDURE [q: Queue, b: SppBuffer] RETURNS [SppBuffer];
GetSppBufferFromPool: PROCEDURE [pool: BufferPool] RETURNS [SppBuffer];
GetFreeSendSppBufferFromPool: PROCEDURE [pool: BufferPool] RETURNS [SppBuffer];
GetFreeReceiveSppBufferFromPool: PROCEDURE [pool: BufferPool] RETURNS [SppBuffer];
MaybeGetSppBufferFromPool: PROCEDURE [pool: BufferPool] RETURNS [SppBuffer];
MaybeGetFreeSendSppBufferFromPool: PROCEDURE [pool: BufferPool] RETURNS [SppBuffer];
MaybeGetFreeReceiveSppBufferFromPool: PROCEDURE [pool: BufferPool] RETURNS [SppBuffer];
ReturnSppBufferToPool: PROCEDURE [pool: BufferPool, b: SppBuffer];
ReturnSendSppBufferToPool: PROCEDURE [pool: BufferPool, b: SppBuffer];
ReturnReceiveSppBufferToPool: PROCEDURE [pool: BufferPool, b: SppBuffer];
-- length setting
GetOisPacketTextLength: PROCEDURE [OisBuffer] RETURNS [CARDINAL];
SetOisPacketTextLength: PROCEDURE [OisBuffer, CARDINAL];
SetOisPacketLength: PROCEDURE [OisBuffer, CARDINAL];
-- whether internal statistics is on or not
GetDoStats: PROCEDURE RETURNS [BOOLEAN];

-- start procedures from CommunicationControl
OiscpPackageMake: PROCEDURE;
OiscpPackageReady: PROCEDURE;
OiscpPackageDestroy: PROCEDURE;

END. -- OISCPDefs

LOG

Time: May 9, 1979 11:00 AM By: Dalal Action: conversion to Pilot 3.0.
Time: January 20, 1980 10:55 AM By: Dalal Action: conversion for Amargosa.
Time: March 12, 1980 10:05 AM By: BLyon Action: Added GetFreeSendOisBufferFromPool, GetFreeReceiveOisBufferFromPool,
```

MaybeGetFreeSendOisBufferFromPool, MaybeGetFreeReceiveOisBufferFromPool, ReturnSendOisBufferToPool, ReturnReceiveOisBufferToPool, GetFreeSendSppBufferFromPool, GetFreeReceiveSppBufferFromPool, MaybeGetFreeSendSppBufferFromPool, MaybeGetFreeReceiveSppBufferFromPool, ReturnSendSppBufferToPool, ReturnReceiveSppBufferToPool. Also considering removing GetOisBufferFromPool, ReturnOisBufferToPool, GetFreeOisBuffer, ReturnFreeOisBuffer.

```
DIRECTORY
 SpecialSystem USING [
  broadcastHostNumber, HostNumber, NetworkAddress, NetworkNumber, nullHostNumber,
  SocketNumber];

OISCPTypes: DEFINITIONS =
BEGIN

-- Addresses and connection IDs defined by the OIS Communication Protocol (OISCP).
OisAddress: TYPE = SpecialSystem.NetworkAddress;
OisNetID: TYPE = SpecialSystem.NetworkNumber;
OisHostID: TYPE = SpecialSystem.HostNumber;
OisSocketID: TYPE = SpecialSystem.SocketNumber;
OisConnectionID: TYPE = RECORD [WORD];

-- Magic numbers for the OIS communication (transport) protocols.

-- OIS packet header lengths.
maxBytesPerOisPkt: CARDINAL = 576;
bytesPerOisPktHeader: CARDINAL = 30;
bytesPerOisPktText: CARDINAL = maxBytesPerOisPkt-bytesPerOisPktHeader;
bytesPerLevel2SppHeader: CARDINAL = 12;
maxDataWordsPerSpp: CARDINAL = (bytesPerOisPktText-bytesPerLevel2SppHeader)/2;

-- Constants describing addresses and connections.
unknownNetID: OisNetID = [0, 0];
unknownHostID: OisHostID = SpecialSystem.nullHostNumber; -- all zeros
allHostIDs: OisHostID = SpecialSystem.broadcastHostNumber; -- all ones
uniqueAddress: OisAddress = [unknownNetID, unknownHostID, unknownSocketID];
unknownConnID: OisConnectionID = [0];
uniqueConnID: OisConnectionID = [0];

-- Fixed network IDs and ranges.
phoneNetID: OisNetID = [0, 256];
maxPupNetID: OisNetID = [0, 255];
minXeroxProductNetID: OisNetID = [0, 512]; -- above this used to number customer nets

-- Well known sockets.
unknownSocketID: OisSocketID = [0];
unqiueSocketID: OisSocketID = [0];
routingInformationSocket: OisSocketID = [1];
echoerSocket: OisSocketID = [2];
errorSocket: OisSocketID = [3];
envoySocket: OisSocketID = [4];
srpcpSocket: OisSocketID = [5];
```

```
OisPacketType: TYPE = {
-- This is an attempt to get the TYPE checker to help us.  Unfortunately, the
-- values are a bit sparce.  Be sure to have enough values to make it an 8 bit field.
-- 000-077 OCTAL!
private,        routingInformation,        echo,        error,        envoyTransaction,        sequencedPacket,
                                                                                  pt006,       pt007,
pt010,          pt011,          pt012,     pt013,       pt014,        pt015,     pt016,       pt017,
pt020,          pt021,          pt022,     pt023,       pt024,        pt025,     pt026,       pt027,
pt030,          pt031,          pt032,     pt033,       pt034,        pt035,     pt036,       pt037,
pt040,          pt041,          pt042,     pt043,       pt044,        pt045,     pt046,       pt047,
pt050,          pt051,          pt052,     pt053,       pt054,        pt055,     pt056,       pt057,
pt060,          pt061,          pt062,     pt063,       pt064,        pt065,     pt066,       pt067,
pt070,          pt071,          pt072,     pt073,       pt074,        pt075,     pt076,       pt077,
-- 100-177
pt100,          pt101,          pt102,     pt103,       pt104,        pt105,     pt106,       pt107,
pt110,          pt111,          pt112,     pt113,       pt114,        pt115,     pt116,       pt117,
pt120,          pt121,          pt122,     pt123,       pt124,        pt125,     pt126,       pt127,
pt130,          pt131,          pt132,     pt133,       pt134,        pt135,     pt136,       pt137,
pt140,          pt141,          pt142,     pt143,       pt144,        pt145,     pt146,       pt147,
pt150,          pt151,          pt152,     pt153,       pt154,        pt155,     pt156,       pt157,
pt160,          pt161,          pt162,     pt163,       pt164,        pt165,     pt166,       pt167,
pt170,          pt171,          pt172,     pt173,       pt174,        pt175,     pt176,       pt177,
-- 200+
pt200,          pt201,          pt202,     pt203,       pt204,        pt205,     pt206,       pt207,
pt210,          pt211,          pt212,     pt213,       pt214,        pt215,     pt216,       pt217,
pt220,          pt221,          pt222,     pt223,       pt224,        pt225,     pt226,       pt227,
pt230,          pt231,          pt232,     pt233,       pt234,        pt235,     pt236,       pt237,
pt240,          pt241,          pt242,     pt243,       pt244,        pt245,     pt246,       pt247};

END.
```

LOG

Time: July 20, 1979 4:01 PM By: Dalal Action: Conversion to Pilot 3.0.
Time: June 2, 1980 1:11 PM By: BLyon Action: put in OISCPTypes Dependencies.
Time: August 6, 1980 10:38 AM By: Garlick Action: added a constant net id for the phone network, phoneNetID. Also defined the minimum net id to be used in allocating customer net numbers.

```
DIRECTORY
    RS232C USING [ChannelHandle],
    RS232CManager USING [CommParamObject];

OISTransporter: DEFINITIONS =
BEGIN
Initialize: PROCEDURE [chHandle: RS232C.ChannelHandle, commParams: RS232CManager.CommParamObject];
Destroy: PROCEDURE [chHandle: RS232C.ChannelHandle];
END.
```

LOG

Time: April 3, 1978 10:46 AM By: yourName Action: Created file
Time: August 5, 1980 5:14 PM By: Victor Schwartz Action: Define a Destroy routine, to allow the RS232CManger to shut off OIS communication when the channel owner calls RS232CManger.ReleaseChannel

```
-- Function: The internal definitions module for Pliot Packet Streams.

DIRECTORY
 BufferDefs USING [ BufferPool],
 Environment USING [Byte],
 OISCPDefs USING [
  OisAddress, SppBuffer, GetFreeSendSppBufferFromPool, ReturnSendSppBufferToPool,
  ReturnReceiveSppBufferToPool],
 OISCPTypes USING [
  bytesPerOisPktHeader, bytesPerOisPktText, bytesPerLevel2SppHeader,
  maxBytesPerOisPkt, OisConnectionID, uniqueConnID, unknownConnID, uniqueAddress],
 NetworkStream USING [
  defaultWaitTime, FailureReason, SuspendReason, WaitTime];

PacketStream: DEFINITIONS
 IMPORTS OISCPDefs =
BEGIN

--definitions

Handle: TYPE = POINTER TO Object;
Object: TYPE = RECORD [
 pool: BufferDefs.BufferPool,
 put: PROCEDURE [OISCPDefs.SppBuffer],
 get: PROCEDURE RETURNS [OISCPDefs.SppBuffer],
 waitForAttention: PROCEDURE RETURNS [OISCPDefs.SppBuffer],
 setWaitTime: PROCEDURE [WaitTime],
 findAddresses: PROCEDURE RETURNS [local, remote: OISCPDefs.OisAddress],
 --getSendSppBuffer: PROCEDURE RETURNS [OISCPDefs.SppBuffer],
 --returnSendSppBuffer: PROCEDURE [OISCPDefs.SppBuffer],
 getSenderSizeLimit: PROCEDURE RETURNS [CARDINAL],
 returnGetSppDataBuffer: PROCEDURE [OISCPDefs.SppBuffer]
 --returnSppBuffer: PROCEDURE [OISCPDefs.SppBuffer]
 ];

Byte: TYPE = Environment.Byte;
OisAddress: TYPE = OISCPDefs.OisAddress;
uniqueAddress: OisAddress = OISCPTypes.uniqueAddress;
SppBuffer: TYPE = OISCPDefs.SppBuffer;
OisConnectionID: TYPE = OISCPTypes.OisConnectionID;
uniqueConnID: OisConnectionID = OISCPTypes.uniqueConnID;
unknownConnID: OisConnectionID = OISCPTypes.unknownConnID;
maxBytesPerOisPkt: CARDINAL = OISCPTypes.maxBytesPerOisPkt;
bytesPerOisPktText: CARDINAL = OISCPTypes.bytesPerOisPktText;
bytesPerLevel2SppHeader: CARDINAL = OISCPTypes.bytesPerLevel2SppHeader;
bytesPerSequencedPktHeader: CARDINAL = bytesPerLevel2SppHeader + OISCPTypes.bytesPerOisPktHeader;
WaitTime: TYPE = NetworkStream.WaitTime;
defaultWaitTime: WaitTime = NetworkStream.defaultWaitTime;

State: TYPE = {
 unestablished, activeEstablish, waitEstablish, established, open, terminating, suspended};
SuspendReason: TYPE = NetworkStream.SuspendReason;
FailureReason: TYPE = NetworkStream.FailureReason;

-- interface

-- exported by PacketStreamMgr
Make: PROCEDURE [local, remote: OisAddress, localConnID, remoteConnID:
 OisConnectionID, establishConnection: BOOLEAN, timeout: WaitTime] RETURNS [Handle];
Destroy: PROCEDURE [Handle];
-- these are INLINEs in order to make call Pilot procedure calls similar in style.
Put: PROCEDURE [psH: Handle, b: OISCPDefs.SppBuffer] = INLINE
 BEGIN
 psH.put[b];
 END;
Get: PROCEDURE [psH: Handle] RETURNS [OISCPDefs.SppBuffer] = INLINE
 BEGIN
 RETURN[psH.get[]];
 END;
WaitForAttention: PROCEDURE [psH: Handle] RETURNS [OISCPDefs.SppBuffer] = INLINE
 BEGIN
 RETURN[psH.waitForAttention[]];
```

```
 END;
SetWaitTime: PROCEDURE [psH: Handle, time: WaitTime] = INLINE
 BEGIN
 psH.setWaitTime[time];
 END;
FindAddresses: PROCEDURE [psH: Handle] RETURNS [local, remote: OISCPDefs.OisAddress] = INLINE
 BEGIN
 [local, remote] ← psH.findAddresses[];
 END;
GetSendSppBuffer: PROCEDURE [psH: Handle] RETURNS [OISCPDefs.SppBuffer] = INLINE
 BEGIN
 RETURN [OISCPDefs.GetFreeSendSppBufferFromPool[psH.pool]];
 END;
ReturnSendSppBuffer: PROCEDURE [psH: Handle, b: OISCPDefs.SppBuffer] = INLINE
 BEGIN
 OISCPDefs.ReturnSendSppBufferToPool[psH.pool, b];
 END;
GetSenderSizeLimit: PROCEDURE [psH: Handle] RETURNS [CARDINAL] = INLINE
 BEGIN
 RETURN[psH.getSenderSizeLimit[]];
 END;
ReturnGetSppBuffer: PROCEDURE [psH: Handle, b: OISCPDefs.SppBuffer] = INLINE
 BEGIN
 OISCPDefs.ReturnReceiveSppBufferToPool[psH.pool, b];
 END;
ReturnGetSppDataBuffer: PROCEDURE [psH: Handle, b: OISCPDefs.SppBuffer] = INLINE
 BEGIN
 psH.returnGetSppDataBuffer[b];
 END;
-- connection table routines
InsertIntoConnectionTable: PROCEDURE [remote: OisAddress, remoteConnID: OisConnectionID];
RemoveFromConnectionTable: PROCEDURE [remote: OisAddress, remoteConnID: OisConnectionID];
ConnectionAlreadyThere: PROCEDURE [remote: OisAddress, remoteConnID: OisConnectionID]
 RETURNS [BOOLEAN];
-- errors and signals
ConnectionSuspended: ERROR [why: SuspendReason];
ConnectionFailed: SIGNAL [why: FailureReason];
AttentionTimeout: ERROR;

END.
```

LOG

Time: May 9, 1978 12:03 PM By: Dalal Action: created file.
Time: January 26, 1980 1:52 PM By: Dalal Action: replaced SPPInternal.
Time: BLyon on: April 14, 1980 6:11 PM By: BLyon Action: Uses new level1 and queue level stuff.

```
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

```
-- Inline access to the Principles of Operation page map.

-- Use a higher level interface unless you cannot call a procedure AND you will not interfere with Pilot's use of this interface!

DIRECTORY
    Environment USING [PageNumber],
    MiscAlpha USING [aASSOC, aGETF, aSETF],
    Mopcodes USING [zMISC];

PageMap: DEFINITIONS =

BEGIN

Flags: TYPE = MACHINE DEPENDENT RECORD [writeProtected, dirty, referenced: BOOLEAN];

maxRealPages: CARDINAL = 4096; -- really determined by size of realPage field of Value

RealPageNumber: TYPE = [0..maxRealPages);

Value: TYPE = MACHINE DEPENDENT RECORD [
    logSingleError: BOOLEAN,
    flags: Flags,
    realPage: RealPageNumber];


flagsClean: Flags = [writeProtected: FALSE, dirty: FALSE, referenced: FALSE];
flagsDirty: Flags = [writeProtected: FALSE, dirty: TRUE, referenced: FALSE];
flagsVacant: Flags = [writeProtected: TRUE, dirty: TRUE, referenced: FALSE];
flagsWriteProtected: Flags = [writeProtected: TRUE, dirty: FALSE, referenced: FALSE];


-- Bit masks for isolation of fields of a Flags:
flagsAll: Flags = [writeProtected: LOOPHOLE[1], dirty: LOOPHOLE[1], referenced: LOOPHOLE[1]];
flagsNone: Flags = [writeProtected: LOOPHOLE[0], dirty: LOOPHOLE[0], referenced: LOOPHOLE[0]];
flagsNotReferenced: Flags = [writeProtected: LOOPHOLE[1], dirty: LOOPHOLE[1], referenced: LOOPHOLE[0]];
flagsReferenced: Flags = [writeProtected: LOOPHOLE[0], dirty: LOOPHOLE[0], referenced: LOOPHOLE[1]];


valueAny: Value = valueVacant;
valueClean: Value = [logSingleError: FALSE, flags: flagsClean, realPage: 0];
valueVacant: Value = [logSingleError: FALSE, flags: flagsVacant, realPage: 0];
valueWriteProtected: Value = [logSingleError: FALSE, flags: flagsWriteProtected, realPage: 0];


-- Bit masks for isolation of fields of a Value:
maskWriteProtectedDirty: Value = [logSingleError: LOOPHOLE[0],
        flags: [writeProtected: LOOPHOLE[1], dirty: LOOPHOLE[1], referenced: LOOPHOLE[0]], realPage: 0];
maskNotDirtyNotReferenced: Value = [logSingleError: LOOPHOLE[1],
        flags: [writeProtected: LOOPHOLE[1], dirty: LOOPHOLE[0], referenced: LOOPHOLE[0]], realPage: LOOPHOLE[7777B]];
maskDirtyReferenced: Value = [logSingleError: LOOPHOLE[0],
        flags: [writeProtected: LOOPHOLE[0], dirty: LOOPHOLE[1], referenced: LOOPHOLE[1]], realPage: 0];


-- Machine instructions from the Principles of Operations:

Assoc: PRIVATE PROCEDURE [page: Environment.PageNumber, value: Value] =
    MACHINE CODE BEGIN Mopcodes.zMISC, MiscAlpha.aASSOC END;


GetF: PRIVATE PROCEDURE [page: Environment.PageNumber] RETURNS [valueOld: Value] =
    MACHINE CODE BEGIN Mopcodes.zMISC, MiscAlpha.aGETF END;


SetF: PRIVATE PROCEDURE [page: Environment.PageNumber, flagsNew: Value] RETURNS [valueOld: Value] =
    MACHINE CODE BEGIN Mopcodes.zMISC, MiscAlpha.aSETF END;

END.
```

LOG

Time: December 11, 1978 2:14 PM	*By: McJones*	*Action: Created file*
Time: January 21, 1980 8:57 AM	*By: Knutsen*	*Action: Added mask values.*
Time: July 21, 1980 2:46 PM	*By: Knutsen*	*Action: Added GetFlags.*

```
PerformancePrograms: DEFINITIONS =
BEGIN
InitializePilotCounter: PROCEDURE;
InitializePilotPerfMonitor: PROCEDURE;
END.
```

LOG

Time: October 10, 1979 8:06 AM *By: Johnsson* *Action: Created file.*
Time: April 14, 1980 11:47 AM *By: Knutsen* *Action: Modules now STARTed by Initialize*[].*

```
-- File: PerfPrivate.mesa
-- Edited by Sandman on September 19, 1980  11:51 AM

DIRECTORY
  PerfStructures USING [
    FrameHandle, GlobalFrameHandle, LongNumber, NullGlobalFrame, PsbHandle, RealPC];

PerfPrivate: DEFINITIONS =
  BEGIN OPEN PerfStructures;

  Number: TYPE = LONG CARDINAL;
  LongNumber: TYPE = PerfStructures.LongNumber;
  OverFlowNumber: Number = 20000000B;

  NodeID: TYPE = RECORD [pc: RealPC, frame: GlobalFrameHandle];

  NullID: NodeID = [[0], NullGlobalFrame];

  CallClass: TYPE = {perf, normal};
  LegAddClass: TYPE = {none, successor};
  LegTrackClass: TYPE = {none, successor, all};

  VersionID: CARDINAL = 09190;

  PerfControlRecord: TYPE = MACHINE DEPENDENT RECORD [
    measuringNow(0): BOOLEAN,
    nodeTable(1): POINTER TO NodeTab,
    legTable(2): POINTER TO LegTab,
    process(3): PsbHandle,
    lastCall(4): CallClass,
    trackLeg(5): LegTrackClass,
    addLeg(6): LegAddClass,
    totalBreaks(7): LONG CARDINAL,
    perfTime(9): LONG CARDINAL,
    totalTime(11): LONG CARDINAL,
    lastID(13): NodeIndex,
    nextLeg(14): CARDINAL,
    nextNode(15): CARDINAL,
    histFree(16): HistIndex,
    histBase(17): HistBase,
    saveBreakHandler(18): FrameHandle,
    self(19): FrameHandle,
    newSession(20): BOOLEAN,
    version(21): CARDINAL,
    pulseConversion(22): LONG CARDINAL ← NULL];

  MaxNodes: CARDINAL = 20; -- maximum number of nodes tracked
  NodeIndex: TYPE = CARDINAL [0..MaxNodes + 1];
  NullNode: NodeIndex = MaxNodes;
  AllNodes: NodeIndex = MaxNodes + 1;
  HighBits: CARDINAL = 77B;
  SubNodeIndex: TYPE = NodeIndex [0..MaxNodes);

  Node: TYPE = RECORD [
    id: NodeID,
    hitsLow: CARDINAL,
    hist: HistIndex,
    overflowed: BOOLEAN,
    hitsHigh: [0..HighBits]];

  NodeTab: TYPE = ARRAY SubNodeIndex OF Node;

  MaxLegs: CARDINAL = 40; -- maximum number of legs tracked
  NoLegSlot: LegIndex = MaxLegs + 1;
  LegIndex: TYPE = CARDINAL [0..MaxLegs + 1];
  SubLegIndex: TYPE = LegIndex [0..MaxLegs);

  Leg: TYPE = RECORD [
    start: Number,
    lock: BOOLEAN,
    from, to: NodeIndex,
    sum: Number,
    owner: UNSPECIFIED,
    hitsLow: CARDINAL,
    hist: HistIndex,
    overflowed, someIgnored: BOOLEAN,
    hitsHigh: [0..HighBits]];
```

```
  LegTab: TYPE = ARRAY SubLegIndex OF Leg;

  HistClass: TYPE = {linear, log};
  HistType: TYPE = {node, leg};

  Histogram: TYPE = RECORD [
    count, scale: CARDINAL,
    sum: Number,
    type: HistType,
    class: HistClass,
    nBuckets: [0..377B],
    overflow: CARDINAL,
    underflow: CARDINAL,
    base: Number,
    buckets: ARRAY [0..0) OF CARDINAL];

  HistBase: TYPE = BASE POINTER;
  HistIndex: TYPE = HistBase RELATIVE ORDERED POINTER [0..HistSpaceSize) TO
    Histogram;
  NullHist: HistIndex = LAST[HistIndex];
  HistSpaceSize: CARDINAL = 256;

  PCR: TYPE = POINTER TO PerfControlRecord;
  ReadWrite: TYPE = {read, write};

  END.
```

```
-- File: PerfStructures.mesa
-- Edited by Johnsson on September 19, 1980  10:46 AM

DIRECTORY
  Inline USING [LongNumber],
  PrincOps USING [
    FrameHandle, GlobalFrameHandle, NullFrame, NullGlobalFrame, BytePC],
  ProcessOperations USING [HandleToIndex, IndexToHandle],
  PSB USING [Handle, Index, nullHandle];

PerfStructures: DEFINITIONS IMPORTS ProcessOperations =
  BEGIN

  FrameHandle: TYPE = PrincOps.FrameHandle;
  GlobalFrameHandle: TYPE = PrincOps.GlobalFrameHandle;
  NullGlobalFrame: GlobalFrameHandle = PrincOps.NullGlobalFrame;
  NullFrame: FrameHandle = PrincOps.NullFrame;
  RealPC: TYPE = PrincOps.BytePC;

  LongNumber: TYPE = Inline.LongNumber;

  PsbIndex: TYPE = PSB.Index;
  PsbHandle: TYPE = PSB.Handle;
  NullPsbHandle: PsbHandle = PSB.nullHandle;
  IndexToHandle: PROCEDURE [i: PsbIndex] RETURNS [PsbHandle] = INLINE
  {RETURN[ProcessOperations.IndexToHandle[i]]};
  HandleToIndex: PROCEDURE [h: PsbHandle] RETURNS [PsbIndex] = INLINE
  {RETURN[ProcessOperations.HandleToIndex[h]]};

  END.
```

```
DIRECTORY
    SpecialSystem: FROM "SpecialSystem" USING [HostNumber],
    RS232CManager: FROM "RS232CManager" USING [CommParamHandle];

PhoneNetwork: DEFINITIONS =
BEGIN

-- Procedures

    KnowAboutPhonePath: PROCEDURE[OISUniqueAddress: SpecialSystem.HostNumber, commParamHandle:
        RS232CManager.CommParamHandle, phoneNumber: AccessAddress];

    ForgetAboutPhonePath: PROCEDURE [OISUniqueAddress: SpecialSystem.HostNumber];

    FindPhonePath: PROCEDURE [OISUniqueAddress: SpecialSystem.HostNumber] RETURNS [commParamHandle:
        RS232CManager.CommParamHandle, phoneNumber: AccessAddress];

-- Types
    AccessAddress: TYPE = STRING;

-- Signals and Errors
    UnknownPath: ERROR; -- host number not in path table
END.
```

LOG

Time: January 30, 1980 2:18 PM By: Garlick Action: Created file from a subset of RS232CManager and converted to use SpecialSystem for the ProcessorID.

Time: August 5, 1980 5:37 PM By: Garlick Action: Changed all SpecialSystem.ProcessorIDs to HostNumber so we could accept broadcastHostNumber.

```
DIRECTORY
    Device USING [nullType, Type],
    System USING [nullID, PhysicalVolumeID],
    Volume USING [ID, PageCount, Type];

PhysicalVolume: DEFINITIONS =
BEGIN

-- Handles are used to talk about drives
Handle: TYPE [3];

-- Types to deal with Physical Volumes
ID: TYPE = System.PhysicalVolumeID;
Layout: TYPE = {partialLogicalVolume, singleLogicalVolume, multipleLogicalVolumes};
PageNumber: TYPE = LONG CARDINAL;
VolumeType: TYPE = {notPilot, probablyNotPilot, probablyPilot, isPilot};

maxNameLength: CARDINAL = 40; -- maximum Physical Volume name length
nullBadPage: PageNumber = LAST[PageNumber];
nullDeviceIndex: CARDINAL = LAST[CARDINAL];
nullID: ID = [System.nullID];

ErrorType: TYPE = {badDisk, badSpotTableFull, containsOpenVolumes, diskReadError, hardwareError, hasPilotVolume,
    alreadyAsserted, insufficientSpace, invalidHandle, nameRequired, notReady, noSuchDrive, noSuchLogicalVolume,
    pageCountTooSmallForVolume, physicalVolumeUnknown, subvolumeHasTooManyBadPages, tooManySubvolumes,
    writeProtected, wrongFormat};
Error: ERROR [error: ErrorType];
CanNotScavenge: ERROR;

-- Find out about the current physical configuration
GetNextDrive: PROC [type: Device.Type, index: CARDINAL] RETURNS [nextType: Device.Type, nextIndex: CARDINAL]; -- Stateless
    enumeration of drives on the system element.  Begins with [Device.nullType, nullDeviceIndex] as arguments and ends with the same pair as
    results.  No guarantees are made as to the order of enumeration.
GetHandle: PROC [type: Device.Type, index: CARDINAL] RETURNS [Handle];
InterpretHandle: PROC [instance: Handle] RETURNS [type: Device.Type, index: CARDINAL];
IsReady: PROC [instance: Handle] RETURNS [ready: BOOLEAN];

-- Monitor state changes
AwaitStateChange: PROC [changeCount: CARDINAL, type: Device.Type ← Device.nullType, index: CARDINAL ← nullDeviceIndex]
    RETURNS [currentChangeCount: CARDINAL];

-- Find out about what is out there on that drive
GetHints: PROC [instance: Handle, label: STRING ← NIL]
    RETURNS [pvID: ID, volumeType: VolumeType];

-- Make volumes accessible for either Pilot or non-Pilot access
AssertPilotVolume: PROC [instance: Handle] RETURNS [ID]; -- Can raise CanNotScavenge
AssertNotAPilotVolume: PROC [instance: Handle];

-- Finish with non-Pilot access
FinishWithNonPilotVolume: PROC [instance: Handle];

-- Deal with current known (on-line) Pilot volumes
CreatePhysicalVolume: PROC [instance: Handle, name: STRING] RETURNS [ID];
GetAttributes: PROC [pvID: ID, label: STRING ← NIL] RETURNS [instance: Handle, layout: Layout];
GetContainingPhysicalVolume: PROC [lvID: Volume.ID] RETURNS [pvID: ID];
    -- Returns the Physical ID of the physical volume containing the FIRST subvolume of this logical volume.
GetNext: PROC [pvID: ID] RETURNS [ID];
    -- Stateless enumeration of online physical volumes, beginning with a nullID argument and ending with a nullID result.
GetNextBadPage: PROC [pvID: ID, thisBadPageNumber: PageNumber] RETURNS [nextBadPageNumber: PageNumber];
GetNextLogicalVolume: PROC [pvID: ID, lvID: Volume.ID] RETURNS [Volume.ID];
    -- Stateless enumeration, beginning with a Volume.nullID argument and ending with a Volume.nullID result, of the accessible logical volumes on
    pvID.  It takes into account the existence of inaccessible volumes (e.g., Debugger volumes while Pilot is running) by not enumerating them: Logical
    volumes with multiple subvolumes on the same physical volume will be enumerated only once.
MarkPageBad: PROC [pvID:ID, badPage: PageNumber];
Offline: PROC [pvID: ID]; -- finish up with physical volume.

-- Create additional logical volumes (or erase current)
maxSubvolumesOnPhysicalVolume: READONLY CARDINAL;
    -- Currenty there is one subvolume per Logical Volume.  When ExtendLogicalVolume is implemented there will be potentially many subvolumes
    per Logical Volume.
```

CreateLogicalVolume: PROC [pvID: ID, size: Volume.PageCount, name: STRING, type: Volume.Type, minPVPageNumber: PageNumber ← 1] RETURNS [Volume.ID];
EraseLogicalVolume: PROC [lvID: Volume.ID]; -- *see Scavenger to scavenge volumes.*
END.

LOG

Time: May 29, 1980 11:24 AM By: Luniewski Action: Created file
Time: June 10, 1980 10:52 AM By: Luniewski Action: Changed Handle to an exported type. Made the truth for ID's be in System.
Time: June 28, 1980 5:47 PM By: Forrest Action: Moved in many procs from old SpecialVolume. Re-arranged. Made nullID compile-time constant.
Time: July 25, 1980 9:17 AM By: Luniewski Action: Updated ErrorType.
Time: September 2, 1980 5:11 PM By: Jose Action: added to ErrorType.

```
DIRECTORY
    Boot USING [nullDiskFileID, PVBootFiles],
    Environment USING [wordsPerPage],
    Inline USING [BITXOR],
    System USING [FileID, nullID, PhysicalVolumeID, VolumeID];

PhysicalVolumeFormat: DEFINITIONS IMPORTS Inline =
BEGIN

Handle: TYPE = LONG POINTER TO Descriptor;

seal, Seal: CARDINAL = 121212B;          -- word zero of a valid physical volume descriptor (root page)
currentVersion: CARDINAL = 6;            -- increment each time Descriptor below is reformatted
```

-- BEWARE: The descriptors defined below cannot be reformatted without invalidating ALL outstanding Pilot volumes. Remember also that non-reconstructable information added to the physical volume root page descriptor must also be added to the subvolume end marker page descriptor below.

```
Descriptor: TYPE = MACHINE DEPENDENT RECORD [
    -- The first page of the descriptor only holds information that is relatively static
    seal (0): CARDINAL ← Seal,                                  -- must be 1st field
    version (1): CARDINAL ← currentVersion,                     -- must be 2nd field
    labelLength (2): CARDINAL [0..physicalVolumeLabelLength) ← 0,
    pvID (3): System.PhysicalVolumeID,
    bootingInfo (10B): Boot.PVBootFiles ← nullPVBootFiles, -- must be at this offset as the microcode knows where to find it.
    label (54B): PACKED ARRAY [0..physicalVolumeLabelLength) OF CHARACTER ← nullLabel | NULL,
    subVolumeCount (100B): CARDINAL [0..maxSubVols),
    subVolumeMarkerID (101B): System.FileID ← [System.nullID],
    badPageCount (106B): PageCount ← 0,
    maxBadPages (110B): PageCount ← Environment.wordsPerPage/SIZE[PageNumber], -- TEMPORARY until multi-page bad page
            tables are implemented
    onLineCount (112B): CARDINAL ← 0, -- TEMPORARILY unused.
    subVolumes (113B): ARRAY [0..maxSubVols) OF SubVolumeDesc,
    fill1 (231B): ARRAY [0..377B-231B) OF WORD ← ALL[0], -- fill to whole page
    checksum (377B): CARDINAL ← 0, -- MUST be the last field of this page
    -- followed, on immediately following pages, by a BadPageList with maxBadPages entries
    badPageList (400B): BadPageListArray]; -- second page is the bad page list. Must fill to two full pages.
    -- Placing the bad page table within Descriptor is a temporary hack to get the bad page list out of the root page. Eventually, it will be a separate
            entity and will be able to be longer than one page. However, this requires that much code be changed. Whenever the change is
            made, the following must be maintained: the bad page list must have the same ID and type as the root page and they must
            immediately follow the root page on the physical volume.


-- The following describes the bad page table
BadPageList: TYPE = LONG POINTER TO BadPageListArray;
BadPageListArray: TYPE = ARRAY [0..maxBadPages) OF PageNumber; -- actual length is Descriptor.maxBadPages
maxBadPages: CARDINAL = Environment.wordsPerPage/SIZE[PageNumber]; -- current maximum number of bad pages permitted on a volume.
        This number may only be increased and never decreased.

-- SubVolumeDesc as stored on disk in physical volume root page. See SubVolume.Descriptor for in-memory form.
SubVolumeDesc: TYPE = MACHINE DEPENDENT RECORD [
    lvID (0): System.VolumeID,
    lvSize (5): PageCount,
    lvPage (7): PageNumber,
    pvPage (11B): PageNumber,
    nPages (13B): PageCount];

PSMSeal: CARDINAL = 141414B;             -- word zero of a valid subvolume end marker page
PSMCurrentVersion: CARDINAL = 0;         -- increment each time SubVolEndMarkerDesc is reformatted

-- The subvolume end marker page marks the end of each subvolume on a physical volume and contains non-reconstructable
   information from both the physical (following) and logical volume root pages, and a checksum.
PhysicalSubvolumeMarker: TYPE = MACHINE DEPENDENT RECORD [
    seal (0): CARDINAL ← PSMSeal,                               -- must be 1st field
    version (1): CARDINAL ← PSMCurrentVersion,                  -- must be 2nd field
    -- Information (mainly) Describing the Physical Volume
    pvID (2): System.PhysicalVolumeID,
    label (7): PACKED ARRAY [0..physicalVolumeLabelLength) OF CHARACTER ← nullLabel,
    bootingInfo (33B): Boot.PVBootFiles ← nullPVBootFiles,
    maxBadPages (77B): PageCount, -- size of bad page table for the physical volume
    labelLength (101B: 0..5): CARDINAL [0..physicalVolumeLabelLength) ← 0,
    -- Information describing the preceding subvolume
    fill (101B: 6..12): [0..128) ← 0,
```

```
        svNumber (101B:13..15): CARDINAL [0..maxSubVols),  -- ordinal number of preceding subvolume
        descriptor (102B): SubVolumeDesc];                 -- copy of descriptor for preceding subvolume

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

descriptorSize: CARDINAL [2*Environment.wordsPerPage..2*Environment.wordsPerPage] = SIZE[Descriptor]; -- compiler check
maxSubVols: CARDINAL [4..7] = 6;                   -- maximum subvolumes on a physical volume.  Because the subVolumes array is just before the
        Fill area in the Descriptor, it is possible to increment this field so that the subVOlumes array effectively grows into the Fill area.
nullBadPage: PageNumber = 0;
nullLabel: PACKED ARRAY [0..physicalVolumeLabelLength) OF CHARACTER = ALL[0C];
nullPVBootFiles: Boot.PVBootFiles = ALL[Boot.nullDiskFileID];
physicalVolumeLabelLength: CARDINAL = 40;
rootPageNumber: PageNumber = 0;

IDCheckSum: PROCEDURE [pvID: System.PhysicalVolumeID] RETURNS [CARDINAL] =
    INLINE BEGIN OPEN LOOPHOLE[pvID, RECORD [a, b, c, d, e: WORD]], Inline;
    RETURN[BITXOR[BITXOR[BITXOR[BITXOR[a, b], c], d], e]]
    END;

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)
Time: August 17, 1979 1:33 PM By: Redell Action: Separated declarations of offline and online SubVolume Descriptors. Added VolumeIDCheckSum.
Time: September 18, 1979 11:41 AM By: Fay Action: Added badPageList, boot file IDs (microcode, germ, pilot), and checksum to root page descriptor; added subvolume end marker page descriptor.
Time: September 18, 1979 2:23 PM By: Forrest Action: Split off part of subvolumeEndMarker and moved to LogicalVolume. Renamed and messed with lotts of minor things.
Time: September 18, 1979 3:44 PM By: Forrest Action: Changed to use PVBootFiles.
Time: September 19, 1979 8:22 AM By: Forrest Action: Added a couple of comments to Marker.
Time: September 19, 1979 1:42 PM By: Forrest Action: Added many default fields.
Time: September 20, 1979 10:42 AM By: Forrest Action: Changed [1..n]'s to [0..n)'s.
Time: January 25, 1980 7:29 PM By: Forrest Action: Changed ID to be SpecialVolume.PhyscialVolumeID.
Time: January 27, 1980 7:43 AM By: Forrest Action: Deleted nullID and UniversalID from USINGS of System.
Time: May 20, 1980 2:46 PM By: Luniewski Action: PhysicalVolume = > PhysicalVolumeFormat. Used ALL's in initializations.
Time: June 16, 1980 10:32 AM By: McJones Action: Changes to accommodate 48-bit processor ids, including additional XOR in IDCheckSum!
Time: September 17, 1980 2:18 PM By: Luniewski Action: Changes to move bad page list off of the root page.

```
PilotClient: DEFINITIONS =
BEGIN
Run: PROCEDURE;
END.
```

LOG

Time: May 8, 1978 8:00 PM By: Horsley Action: Created file
Time: September 14, 1978 10:41 AM By: Lauer Action: Changed interface from a PROGRAM to a PROCEDURE

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

DIRECTORY
    Environment USING [PageOffset],
    File USING [ID, nullID, PageNumber, Type],
    Inline USING [BITXOR, LongNumber];

PilotDisk: DEFINITIONS IMPORTS Inline =

BEGIN

-- Definitions common to faces for all disks supported as Pilot volumes.
-- The procedures defined here may be called from face implementations, including those used by the germ.

    -- Generalized disk drive handles
    -- To be used as "device handle" type by face for any disk supported as Pilot volume.
    -- Unique only within drives of same face.
    -- Use of this type ensures room in Pilot disk driver tables for device handle.

Handle: TYPE = PRIVATE RECORD [UNSPECIFIED];

    -- Disk addresses
    -- Used as place holder within boot chain link field of page label (see below).
    -- "Exported" type; contents may vary from disk to disk.

Address: TYPE = PRIVATE RECORD [a, b: UNSPECIFIED];
nullAddress: Address = [a: LAST[CARDINAL], b: LAST[CARDINAL]]; -- flags end of boot chain

    -- Page labels
    -- Every page (sector) of a Pilot disk contains a label record.  Disk operations may verify the contents of the label before transferring
    the actual data record.  To support transfers of runs of pages at disk speeds, the disk device (i.e. electronics, controller microcode,
    and/or head software) must know something about Pilot's label format to verify the contents and to compute the delta from one
    label to the next.

Label: TYPE = MACHINE DEPENDENT RECORD [
    fileID (0): File.ID,                                    -- valid in label of every page
    filePageLo (5): CARDINAL, filePageHi (6:0..6): [0..128), -- 23 bit page pumber, valid in label of every page
    pad1 (6:7..12): [0..64) ← 0,                            -- always zero
    immutable (6:13..13): BOOLEAN,                          -- valid only in label of page 0
    temporary (6:14..14): BOOLEAN,                          -- valid only in label of page 0
    zeroSize (6:15..15): BOOLEAN,                           -- valid only in label of page 0
    type (7): File.Type,                                    -- valid in label of every page
    bootChainLink (10B): Address];                          -- valid in label of every page

nullLabel: Label = [
    fileID: File.nullID,
    filePageLo: 0, filePageHi: 0,
    immutable: FALSE,
    temporary: FALSE,
    zeroSize: FALSE,
    pad1: 0,
    type: [0],
    bootChainLink: [0, 0]];

LN: PRIVATE PROCEDURE [lowbits, highbits: CARDINAL] RETURNS [LONG CARDINAL] =
    MACHINE CODE BEGIN END;

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

SetLabelFilePage: 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;

MatchLabels: PROCEDURE [p1, p2: LONG POINTER TO Label] RETURNS [match: BOOLEAN] =
    INLINE BEGIN
    -- Test whether significant fields of p1↑ and p2↑ are equal.
    -- All fields except bootChainLink are significant.
    -- NOTE: We use two-word compares since the compiler expands them into inline machine code rather than a kernel function call.
```

```
    MatchableLabel: TYPE = MACHINE DEPENDENT RECORD [
        a, b, c, d, dontCare: RECORD [UNSPECIFIED, UNSPECIFIED]];
    BEGIN OPEN m1: LOOPHOLE[p1↑, MatchableLabel], m2: LOOPHOLE[p2↑, MatchableLabel];
    RETURN[m1.a = m2.a AND m1.b = m2.b AND m1.c = m2.c AND m1.d = m2.d]
    END
    END;

NextLabel: PROCEDURE [p: LONG POINTER TO Label] =
    INLINE BEGIN
    -- Modify p to match next page label in run.
    IF (p.filePageLo ← p.filePageLo + 1) = 0 THEN p.filePageHi ← p.filePageHi + 1;
    p.immutable ← p.temporary ← p.zeroSize ← FALSE;
    END;

--- GetLabelType, SetLabelType are for compatibility with old FilePageLabel; remove when convenient:
GetLabelType: PROCEDURE [label: LONG POINTER TO Label] RETURNS [File.Type] =
    INLINE BEGIN RETURN [label.type] END;

SetLabelType: PROCEDURE [label: LONG POINTER TO Label, type: File.Type] =
    INLINE BEGIN label.type ← type END;

LabelCheckSum: PROCEDURE [label: LONG POINTER TO Label, offset: Environment.PageOffset]
        RETURNS [checkSum: CARDINAL] =
    INLINE BEGIN OPEN fID: LOOPHOLE[label.fileID, MACHINE DEPENDENT RECORD [a, b, c, d, e: UNSPECIFIED]];
    adjFilePage: Inline.LongNumber = LOOPHOLE[GetLabelFilePage[label]-offset];
    RETURN[
        Inline.BITXOR[fID.a,
            Inline.BITXOR[fID.b,
                Inline.BITXOR[fID.c,
                    Inline.BITXOR[fID.d,
                        Inline.BITXOR[fID.e,
                            Inline.BITXOR[adjFilePage.lowbits, adjFilePage.highbits]]]]]]]
    END;

END.
```

LOG

Time: May 9, 1980 5:30 PM By: McJones Action: Create file from OISDisk (last edited by McJones on February 7, 1980 9:40 AM) and from FilePageLabel (last edited by Redell on August 17, 1979 6:43 PM)

```
-- This file is the definition of file types used by Pilot ITSELF.  See FileTypes.mesa for client ranges.

DIRECTORY File USING [Type];

PilotFileTypes: DEFINITIONS =
BEGIN

PilotFileType: TYPE = CARDINAL[0..256);
tUnassigned:                                File.Type = [PilotFileType[0]];

-- A volume file, with type in PilotVFileType, does not appear in any volume file map; it is only accessible via pinned file and page group
   descriptors.  File.Create will not accept types in this range.
-- A root file, with type in PilotRootFileType, has a special slot in the logical volume root page where its id is saved; there should be at
   most one file of a given root file type on a logical volume.  If new root types are to be added, space must be found in the logical
   volume root page.
-- Not all root file types are volume file types, and vice versa.

-- Beginning of volume file types (which are not root file types):
PilotVFileType: TYPE = PilotFileType[1..8];
tPhysicalVolumeRootPage:                    File.Type = [PilotVFileType[1]];
tBadPageList:                               File.Type = [PilotVFileType[2]];
tBadPage:                                   File.Type = [PilotVFileType[3]];
tSubVolumeMarkerPage:                       File.Type = [PilotVFileType[4]];
tLogicalVolumeRootPage:                     File.Type = [PilotVFileType[5]];

-- Beginning of types which are both volume files and root files :
PilotRootFileType: TYPE = PilotFileType[6..14];
tFreePage:                                  File.Type = [PilotRootFileType[6]];
tVolumeAllocationMap:                       File.Type = [PilotRootFileType[7]];
tVolumeFileMap:                             File.Type = [PilotRootFileType[8]];

-- Beginning of root file types (which are not volume file types):
tScavengerLog:                              File.Type = [PilotRootFileType[9]];
    -- scavenger log describing its own volume
tTempFileList:                              File.Type = [PilotRootFileType[10]];
tTransactionStateFile:                      File.Type = [PilotRootFileType[11]];
tVMBackingFile:                             File.Type = [PilotRootFileType[12]];
tBeingMoved:                                File.Type = [PilotRootFileType[13]];
tBeingReplicated:                           File.Type = [PilotRootFileType[14]];
-- (Additional root file types would go here.)

-- Beginning of types which are neither volume files nor root files :
tAnonymousFile:                             File.Type = [PilotFileType[15]];
tTransactionLog:                            File.Type = [PilotFileType[16]];
tScavengerLogOtherVolume:                   File.Type = [PilotFileType[17]];
    -- scavenger log describing other than its own volume

END.
```

LOG

(For earlier log entries see Pilot 4.0 archive version.)

Time: May 19, 1980 3:16 PM	*By: Gobbel*	*Action: Renamed tCodeFile to be tTransactionStateFile*
Time: June 2, 1980 5:01 PM	*By: McJones*	*Action: Rearranged root file types to exclude physical volume files; added tBadPageList; deleted tSpaceBTree; added tTransactionLog*
Time: August 15, 1980 3:01 PM	*By: McJones*	*Action: Deleted tBootFile*
Time: October 9, 1980 1:45 PM	*By: Forrest*	*Action: Added tScavengerLogOtherVolume*

```
DIRECTORY
    Environment USING [wordsPerPage],
    FloppyChannel USING [Context];

PilotFloppyFormat: DEFINITIONS =

BEGIN

-- Constants and locations of physical landmarks for a floppy disk Pilot volume.

-- TYPES
FloppyPageNumber: TYPE = CARDINAL;
        -- Short version of DiskChannel.DiskPageNumber. Assumes # pages on floppy is < max short cardinal.

-- CONSTANTS

-- Pilot volumes are supported only on diskettes with IBM (vs. Troy) format using the 512-byte sector size option, on double density.
        Track 0 is ignored. Labels begin with track 1, and logical pages visible to Pilot follow (and exclude) the labels. A single copy
        of the labels is assumed here; adjust logicalTrack0 and DriveObject's idea of number of cylinders for a second copy.

firstRealCylinder: CARDINAL = 1;
        -- Track 0 is ignored entirely.

labelBasePage: FloppyPageNumber = 0;
        -- Beginning of physical track 1, which is the beginning of addressable space.

logicalTrack0: CARDINAL = 4 - firstRealCylinder;
        -- Beginning of physical track 4. This excludes track 0 and the label table space.

pilotContext: FloppyChannel.Context = [protect: FALSE, format: IBM, density: single, sectorLength: 256];
        -- Density is single for Dolphin (temporary), double for Dandelion.

wordsPerSector: CARDINAL = Environment.wordsPerPage;
        END.
```

LOG

Time: June 17, 1980 4:42 PM By: Jose Action: Created file.
Time: September 15, 1980 2:03 PM By: Jose Action: Added pilotContext.

```
-- PilotLoaderOps.mesa   Last Modified by
-- Sandman,  August 4, 1980  8:13 AM
-- Karlton,  Feb 27, 1980 11:36 AM
-- Forrest,  May 17, 1980  6:48 PM

DIRECTORY
  BcdDefs USING [EXPIndex, MTIndex, NameRecord],
  BcdOps USING [BcdBase, MTHandle, NameString],
  PrincOps USING [ControlLink, ControlModule, GFTIndex, GlobalFrameHandle, MaxNGfi],
  PilotLoadStateOps USING [Map];

PilotLoaderOps: DEFINITIONS =

  BEGIN OPEN BcdDefs, BcdOps, PrincOps;

  BindLink: TYPE = RECORD [
    whichgfi: [0..MaxNGfi),
    body: SELECT type: * FROM
      interface => [eti: BcdDefs.EXPIndex],
      module => [mti: BcdDefs.MTIndex],
      notbound => NULL,
      ENDCASE];

  Binding: TYPE = DESCRIPTOR FOR ARRAY OF BindLink;
    -- BASE[Binding] is biased by bcd.firstdummy


  InitBinding: PROCEDURE [bcd: BcdOps.BcdBase] RETURNS [binding: Binding];
  ReleaseBinding: PROCEDURE [bcd: BcdOps.BcdBase, binding: Binding]
    RETURNS [null: Binding];

  PilotLoaderSupport: PROGRAM;

  FindCode: PROCEDURE [bcd: BcdOps.BcdBase, map: PilotLoadStateOps.Map];
  BadCode: SIGNAL [name: STRING];
  VersionMismatch: SIGNAL [name: STRING];

  FrameList: TYPE = POINTER TO FrameItem;
  FrameItem: TYPE = RECORD [frame: POINTER, next: FrameList];
  AllocateFrames: PROCEDURE [size: CARDINAL, single, resident: BOOLEAN]
    RETURNS [FrameList];
  ReleaseFrames: PROCEDURE [
    bcd: BcdOps.BcdBase, frames: FrameList, map: PilotLoadStateOps.Map];

  InitializeMap: PROCEDURE [bcd: BcdOps.BcdBase]
    RETURNS [map: PilotLoadStateOps.Map];
  DestroyMap: PROCEDURE [map: PilotLoadStateOps.Map];

  OpenLinkSpace: PROCEDURE [frame: GlobalFrameHandle, mth: MTHandle];
  WriteLink: PROCEDURE [offset: CARDINAL, link: ControlLink];
  ReadLink: PROCEDURE [offset: CARDINAL] RETURNS [link: ControlLink];
  CloseLinkSpace: PROCEDURE [frame: GlobalFrameHandle];

  GetSpace: PROCEDURE [nwords: CARDINAL] RETURNS [p: POINTER];
  FreeSpace: PROCEDURE [p: POINTER];
  AppendName: PROCEDURE [s: STRING, ssb: NameString, name: NameRecord];

  New: PROCEDURE [bcd: BcdOps.BcdBase, framelinks: BOOLEAN]
    RETURNS [frame: PrincOps.ControlModule];

  END....
```

```
-- PilotLoadStateFormat.mesa
-- Last Modified by McJones,  April 18, 1980  1:26 PM

DIRECTORY
  Environment: FROM "Environment" USING [wordsPerPage],
  PrincOps: FROM "PrincOps" USING [GFTIndex];

PilotLoadStateFormat: DEFINITIONS =

  BEGIN

  LoadState: TYPE = POINTER TO LoadStateObject;

--  AltoVersionID: CARDINAL = 01280;
  PilotVersionID: CARDINAL = 01290;

  LoadStateObject: TYPE = MACHINE DEPENDENT RECORD [
    versionident: CARDINAL,
    nBcds: CARDINAL,
    bcds: ARRAY [0..MaxBcds) OF BcdObject,
    gft: ARRAY [0..0) OF ModuleInfo];

  ModuleTable: TYPE = DESCRIPTOR FOR ARRAY OF ModuleInfo;

  ConfigIndex: TYPE = [0..37B];
    NullConfig: ConfigIndex = LAST[ConfigIndex];

  ModuleInfo: TYPE = MACHINE DEPENDENT RECORD [
    resolved: BOOLEAN,
    config: ConfigIndex,
    gfi: PrincOps.GFTIndex];

  NullModule: ModuleInfo = [resolved: FALSE, config: NullConfig, gfi: 0];

  BcdObject: TYPE = MACHINE DEPENDENT RECORD [
    fill: [0..77B] ← 0,
    exports, typeExported: BOOLEAN,
    pages: [1..256],
    base: LONG POINTER];

  Bcd: TYPE = POINTER TO BcdObject;

  MaxBcds: CARDINAL = (Environment.wordsPerPage-2)/SIZE[BcdObject];
  BcdArrayLength: CARDINAL = SIZE[BcdObject]*MaxBcds;

  BcdPageCount: TYPE = [1..256];

  END.
```

```
-- PilotLoadStateOps.mesa
-- Last Modified by McJones,  April 23, 1980  5:04 PM

DIRECTORY
  BcdOps: FROM "BcdOps" USING [BcdBase],
  PilotLoadStateFormat: FROM "PilotLoadStateFormat" USING [
    ConfigIndex, ModuleInfo, NullConfig],
  PrincOps: FROM "PrincOps" USING [GFTIndex];

PilotLoadStateOps: DEFINITIONS =

  BEGIN OPEN PrincOps, PilotLoadStateFormat;

  ConfigIndex: TYPE = PilotLoadStateFormat.ConfigIndex;
    NullConfig: ConfigIndex = PilotLoadStateFormat.NullConfig;

  MapConfigToReal: PROCEDURE [cgfi: GFTIndex, config: ConfigIndex]
    RETURNS [rgfi: GFTIndex];
  MapRealToConfig: PROCEDURE [rgfi: GFTIndex]
    RETURNS [cgfi: GFTIndex, config: ConfigIndex];
  EnterModule: PROCEDURE [rgfi: GFTIndex, module: ModuleInfo];
  GetModule: PROCEDURE [rgfi: GFTIndex] RETURNS [module: ModuleInfo];

  Map: TYPE = DESCRIPTOR FOR ARRAY OF GFTIndex;

  GetMap: PROCEDURE [config: ConfigIndex] RETURNS [map: Map];
  ReleaseMap: PROCEDURE [map: Map];

  InputLoadState: PROCEDURE RETURNS [nbcds: ConfigIndex];
  ReleaseLoadState: PROCEDURE;
  UpdateLoadState: PROCEDURE [config: ConfigIndex, bcd: BcdOps.BcdBase];
  RemoveConfig: PROCEDURE [map: Map, config: ConfigIndex];
  LoadStateInvalid: SIGNAL;
  LoadStateFull: SIGNAL;

  AcquireBcd: PROCEDURE [config: ConfigIndex] RETURNS [bcd: BcdOps.BcdBase];
  ReleaseBcd: PROCEDURE [bcd: BcdOps.BcdBase];
  BcdUnresolved: PROCEDURE [bcd: ConfigIndex] RETURNS [BOOLEAN];
  BcdExports: PROCEDURE [bcd: ConfigIndex] RETURNS [BOOLEAN];
  BcdExportsTypes: PROCEDURE [bcd: ConfigIndex] RETURNS [BOOLEAN];

  EnumerationDirection: TYPE = {recentfirst, recentlast};

  EnumerateModules: PROCEDURE [
    proc: PROCEDURE [rgfi: GFTIndex, module: ModuleInfo] RETURNS [BOOLEAN]]
    RETURNS [GFTIndex];
  EnumerateBcds: PROCEDURE [dir: EnumerationDirection,
    proc: PROCEDURE [ConfigIndex] RETURNS [BOOLEAN]]
    RETURNS [config: ConfigIndex];

  SetLoadState: PROCEDURE [newState: LONG POINTER];

  END.
```

```
PilotMP: DEFINITIONS =

BEGIN

-- Maintenance Panel codes displayed by Pilot (e.g. when going to debugger isn't appropriate).

-- This is the source for Pilot codes found in MPCodes.bravo.

-- This module has no interface items or records.  Do not include it in any other interfaces, and then it can be recompiled at any time.

Code: TYPE = [0..10000);

cGerm: Code = 900; -- germ entered
  cGermAllocTrap: Code = 901; -- out of frames (Pilot bug)
  cGermControlFault: Code = 902; -- unexpected trap or kernel function call (Pilot bug)
  cGermStartFault: Code = 903; -- attempt to start an already started module (Pilot bug)
  cGermMemoryFault: Code = 904; -- page or write protect fault encountered (Pilot bug)
  cGermERROR: Code = 909; -- unnamed ERROR (Pilot bug)
cGermAction: Code = 910; -- germ action running (e.g. inLoad, outLoad)
  cGermBadPhysicalVolume: Code = 911; -- germ and physical volume have incompatible version numbers
  cGermBadBootFile: Code = 912; -- germ and boot file have incompatible version numbers
  cGermNoPhysicalBootFile: Code = 913; -- no physical boot file installed
  cTeledebugServer: Code = 915; -- germ's teledebug server has control
  cGermFinished: Code = 919; -- germ transferred control back to caller (who has hung)
 cGermDriver: Code = 920; -- germ driver running (e.g. disk, ether)
  cGermDeviceError: Code = 921; -- hard error on device being booted
  cGermTimeout: Code = 922; -- operation on boot device no completed in expected time
  cGermLabelCheck: Code = 923; -- broken link in chained boot file (try reinstalling)
  cGermNoServer: Code = 924; -- no response to germ's request for ether boot file
  cGermFunnyPacket: Code = 925; -- e.g. unexpected sequence number or size
cControl: Code = 930; -- Pilot Control and MesaRuntime components being initialized
  cBadBootFile: Code = 931; -- Pilot and StartPilot have incompatible version numbers
  cEarlyTrap: Code = 932; -- runtime trap before appropriate trap handler set up (Pilot bug)
  cCantSwap: Code = 935; -- debugger not correctly installed or too early in startup to find debugger
  cHang: Code = 936; -- swap to debugger canceled do to use of "H" Keyswitch
  cCleanup: Code = 938; -- running cleanup procedures, e.g. before world swap
  cPowerOff: Code = 939; -- ProcessorFace.PowerOff called, and no power control relay
cStorage: Code = 940; -- Pilot Store component being initialized
  cDriveNotReady: Code = 947; -- waiting for disk drive to become ready
  cScavenging: Code = 950; -- logical volume being scavenged, if necessary
  cDeleteTemps: Code = 960; -- temporary files from previous run being deleted
cMap: Code = 970; -- client and other non-bootloaded code being mapped
cTransactionCrashRecovery: Code = 975; -- Transaction log being processed
cCommunication: Code = 980; -- Pilot Communication component being initialized
cClient: Code = 990; -- PilotClient.Run called

END.
```

LOG

Time: June 7, 1979 10:53 AM By: McJones Action: Created file
Time: April 10, 1980 11:49 AM By: Forrest Action: Added cTeledebugServer
Time: July 31, 1980 5:15 PM By: Forrest Action: Added cHang
Time: August 27, 1980 4:36 PM By: McJones Action: Added cGermFinished, cCleanup
Time: September 4, 1980 9:42 PM By: Forrest Action: Transactin Recovery

```
DIRECTORY
    TemporaryBooting: FROM "TemporaryBooting" USING [Switches];

PilotSwitches: DEFINITIONS =

BEGIN

-- Array of Booleans available for parameter setting within Pilot

-- Clear use of particular switch through manager of SDD System Software Pilot group to avoid conflicts

switches: READONLY TemporaryBooting.Switches; -- in Mesa 6 this will be a packed array of Boolean

END.
```

LOG

Time: August 15, 1979 1:42 PM By: McJones Action: Create file
Time: February 18, 1980 2:10 PM By: McJones Action: Action: Keys.Keybits = > TemporaryBooting.Switches

```
-- PrincOps.Mesa  Edited by Sandman on August 26, 1980  12:18 PM

PrincOps: DEFINITIONS =
  BEGIN

  BYTE: TYPE = [0..377B];

  -- control link definitions

  ControlLinkTag: TYPE = {frame, procedure, indirect, rep};

  ControlLink: TYPE = MACHINE DEPENDENT RECORD [
    SELECT OVERLAID ControlLinkTag FROM
      frame => [frame: FrameHandle],
      procedure => [gfi: GFTIndex, ep: EPIndex, tag: BOOLEAN],
      indirect => [
        SELECT OVERLAID * FROM
          port => [port: PortHandle],
          link => [link: POINTER TO ControlLink],
          ENDCASE],
      rep => [fill: [0..37777B], indirect: BOOLEAN, proc: BOOLEAN],
      ENDCASE];

  ProcDesc, SignalDesc: TYPE = procedure ControlLink;

  TrapLink, NullLink: ControlLink = ControlLink[frame[NullFrame]];
  UnboundLink: ControlLink = ControlLink[procedure[gfi: 0, ep: 0, tag: TRUE]];

  PortHandle: TYPE = POINTER TO Port;

  Port: TYPE = MACHINE DEPENDENT RECORD [
    SELECT OVERLAID * FROM
      representation => [in, out: UNSPECIFIED],
      links => [frame: FrameHandle, dest: ControlLink],
      ENDCASE];

  -- frame definitions

  MaxFrameSize: CARDINAL = 4096 - 4;
  MaxSmallFrameIndex: CARDINAL = 17;

  FrameVec: ARRAY [0..LastAVSlot] OF [0..MaxFrameSize] =
    [7, 11, 15, 19, 23, 27, 31, 39, 47, 55, 67, 79, 95, 111, 127, 147, 171, 203,
      252, 508, 764, 1020, 1276, 1532, 1788, 2044, 2554, 3068, 3580, 4092];

  FrameClass: TYPE = {global, local, signal, catch, dying};

  Frame: TYPE = MACHINE DEPENDENT RECORD [
    accesslink: GlobalFrameHandle,
    pc: BytePC,
    returnlink: ControlLink,
    extensions:
      SELECT OVERLAID FrameClass FROM
        local => [unused: UNSPECIFIED, local: ARRAY [0..0) OF UNSPECIFIED],
        signal => [mark: BOOLEAN, unused: [0..77777B]],
        catch => [
          unused: UNSPECIFIED, staticlink: FrameHandle, messageval: UNSPECIFIED],
        dying => [state: {dead, alive}, unused: [0..77777B]],
        ENDCASE];

  FrameHandle: TYPE = POINTER TO Frame;
  NullFrame: FrameHandle = LOOPHOLE[0];

  GlobalFrame: TYPE = MACHINE DEPENDENT RECORD [
    gfi: GFTIndex,
    copied, alloced, shared, started: BOOLEAN,
    trapxfers, codelinks: BOOLEAN,
    code: FrameCodeBase,
    global: ARRAY [0..0) OF UNSPECIFIED];

  FrameCodeBase: TYPE = MACHINE DEPENDENT RECORD [
    SELECT OVERLAID * FROM
      long => [longbase: LONG POINTER],
      short => [shortbase: POINTER, handle: POINTER],
      offset => [offset: CARDINAL, highHalf: UNSPECIFIED],
      either => [fill: [0..77777B], out: BOOLEAN, highByte, otherByte: BYTE],
      ENDCASE];
```

```
  GlobalFrameHandle: TYPE = POINTER TO GlobalFrame;
  NullGlobalFrame: GlobalFrameHandle = LOOPHOLE[0];

  -- The following offsets are used by the compiler and MUST
  -- reflect the field offsets in the definitions of Frame
  -- local frames

  accessOffset: CARDINAL = 0;
  pcOffset: CARDINAL = 1;
  returnOffset: CARDINAL = 2;

  -- global frames

  gfiOffset: CARDINAL = 0;
  codebaseOffset: CARDINAL = 1;

  -- efficiently addressable portion of frames

  globalbase: CARDINAL = SIZE[GlobalFrame];
  localbase: CARDINAL = SIZE[local Frame];
  framelink: CARDINAL = localbase;

  -- code segments

  BytePC: TYPE = RECORD [CARDINAL];

  InstWord: TYPE = MACHINE DEPENDENT RECORD [evenbyte, oddbyte: BYTE];

  FieldDescriptor: TYPE = MACHINE DEPENDENT RECORD [
    offset: BYTE, posn: [0..16), size: [1..16]];

  EPRange: CARDINAL = 32;
  EPIndex: TYPE = [0..EPRange);

  -- Global frame size preceeds body zero

  CSegPrefix: TYPE = MACHINE DEPENDENT RECORD [
    header: PrefixHeader, entry: ARRAY [0..0) OF EntryVectorItem];

  PrefixHandle: TYPE = POINTER TO CSegPrefix;

  PrefixHeader: TYPE = MACHINE DEPENDENT RECORD [
    swapinfo: WORD, info: PrefixInfo];

  PrefixInfo: TYPE = MACHINE DEPENDENT RECORD [
    stops: BOOLEAN,
    altoCode: BOOLEAN,
    fill: [0..17B],
    ngfi: [1..MaxNGfi],
    nlinks: [0..MaxNLinks]];

  EntryVectorItem: TYPE = MACHINE DEPENDENT RECORD [
    initialpc: BytePC, info: EntryInfo];

  EntryInfo: TYPE = MACHINE DEPENDENT RECORD [
    defaults: BOOLEAN, nparams: [0..177B], framesize: [0..377B]];

  MaxNLinks: CARDINAL = 255;
  MainBodyIndex: CARDINAL = 0;

  -- Global Frame Table definitions

  MaxNGfi: CARDINAL = 4;

  --  GFTHandle: TYPE = POINTER TO ARRAY [0..0) OF GFTItem;
  --  GFT: GFTHandle = LOOPHOLE[1400B];
  --  GFTItem: TYPE = MACHINE DEPENDENT RECORD [
  --    frame: GlobalFrameHandle,
  --    epbase: CARDINAL];

  GFTIndex: TYPE = [0..1777B];
  GFTNull: GFTIndex = 0;

  --  NullEpBase: CARDINAL = LAST[CARDINAL];
  --  MaxGFTSize: CARDINAL = (LAST[GFTIndex]+1)*SIZE[GFTItem];
  -- system frame allocation vector

  AllocTag: TYPE = {frame, empty, indirect, enable};
```

```
  AVItem: TYPE = MACHINE DEPENDENT RECORD [
    SELECT OVERLAID * FROM
      data => [fsi: [0..37777B], tag: AllocTag],
      link => [link: POINTER TO AVItem],
      frame => [frame: FrameHandle],
      ENDCASE];

  AllocationVectorSize: CARDINAL = 40B;
  LastAVSlot: CARDINAL = AllocationVectorSize - 3;
  AllocationVector: TYPE = ARRAY [0..AllocationVectorSize) OF AVItem;
  AVHandle: TYPE = POINTER TO AllocationVector;
  AV: AVHandle = LOOPHOLE[1000B];
  LargeReturnSlot: CARDINAL = AllocationVectorSize - 2;
  SpecialReturnSlot: CARDINAL = AllocationVectorSize - 1;

  -- control module stuff

  ControlModule: TYPE = MACHINE DEPENDENT RECORD [
    SELECT OVERLAID * FROM
      frame => [frame: GlobalFrameHandle],
      list => [list: POINTER TO FrameList],
      tag => [fill: [0..77777B], multiple: BOOLEAN],
      ENDCASE];

  FrameList: TYPE = MACHINE DEPENDENT RECORD [
    nModules: CARDINAL, frames: ARRAY [0..0) OF GlobalFrameHandle];

  NullControl: ControlModule = [frame[NullGlobalFrame]];

  -- control registers
  -- known by the microcode

  PSBreg: CARDINAL = 0; -- current process

  WDCreg: CARDINAL = 1; -- wakeup disable counter

  XTSreg: CARDINAL = 2; -- xfer trap status

  XTPreg: CARDINAL = 3; -- xfer trap parameter

  ATPreg: CARDINAL = 4; -- alloc trap parameter

  OTPreg: CARDINAL = 5; -- other trap parameter

  MDSreg: CARDINAL = 6; -- main data space

  PDAreg: CARDINAL = 7; -- process data area

  PTCreg: CARDINAL = 8; -- process tick count

  SVPointer: TYPE = POINTER TO StateVector;

  StateVector: TYPE = MACHINE DEPENDENT RECORD [
    stk: ARRAY [0..7] OF UNSPECIFIED,
    instbyte: BYTE,
    stkptr: BYTE, -- 0 => empty stack
    dest, source: UNSPECIFIED];

  MaxParamsInStack: CARDINAL = 5;

  END.
```

```
-- PrincOpsRuntime.mesa (last edited by: McJones on: September 16, 1980  11:14 AM)

DIRECTORY
  Mopcodes USING [zAND],
  PrincOps USING [GFTIndex, GlobalFrameHandle];

PrincOpsRuntime: DEFINITIONS =
  BEGIN

  -- Global Frame Table definitions

  GFTHandle: TYPE = POINTER TO ARRAY [0..0) OF GFTItem;
  GFT: GFTHandle = LOOPHOLE[1400B];

  GFTItem: TYPE = MACHINE DEPENDENT RECORD [
    SELECT OVERLAID * FROM
      frame => [framePtr: PrincOps.GlobalFrameHandle],
      ep => [data: [0..37777B], epbias: [0..3]],
      ENDCASE];

  EmptyGFTItem: GFTItem = [ep[data: 0, epbias: 0]];
  FreedGFTItem: GFTItem = [ep[data: 0, epbias: 3]];

  -- Procedure for easy access of frame handle
  GetFrame: PROC [g: GFTItem] RETURNS [PrincOps.GlobalFrameHandle] =
    INLINE BEGIN
    And: PROC [g: GFTItem, mask: WORD] RETURNS[PrincOps.GlobalFrameHandle] =
      MACHINE CODE BEGIN Mopcodes.zAND; END;
    RETURN[And[g, 177774B]];
    END;

  MaxGFTSize: CARDINAL =  (LAST[PrincOps.GFTIndex] + 1)*SIZE[GFTItem];

  END.

June 30, 1980  3:00 PM  Sandman Created file
July 20, 1980  8:24 PM  Forrest ?
September 16, 1980  11:14 AM    McJones Switched to 1-word GFT entries
```

```
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];
EnableAborts: PROCEDURE [LONG POINTER TO CONDITION];

    --Control of Scheduling
Pause: PROCEDURE [ticks: Ticks];
Yield: PROCEDURE;

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

    --Signals and errors generated by the Process machinery
-- Aborted will be removed from the interface in a future version of Pilot.
Aborted: ERROR = --ABORTED-- LOOPHOLE[5]; -- compiler bug prevents direct equation
TooManyProcesses: ERROR;
END.
```

LOG

Time: April 24, 1980 3:51 PM By: Forrest Action: Equated Aborted to ABORTED. Trimmed log to Amargosa.
Time: May 19, 1980 4:17 PM By: Forrest Action: Added Pause.
Time: October 10, 1980 4:09 PM By: Fay Action: Added EnableAborts; changed Aborted from LOOPHOLE[3] to LOOPHOLE[5].

```
DIRECTORY
    Mopcodes USING [zDWDC, zIWDC];

ProcessInternal: DEFINITIONS =

BEGIN

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


    -- Naked notifies
AllocateNakedCondition: PROCEDURE RETURNS [cv: LONG POINTER TO CONDITION, mask: WORD];
    -- Allocate one of the sixteen naked notify condition variables, initialize it with the default timeout, and return a long pointer and
            wakeup mask with an appropriate bit set.
DeallocateNakedCondition: PROCEDURE [cv: LONG POINTER TO CONDITION];
    -- Deallocate a naked notify condition variable previously allocated with AllocateNakedCondition.

END.
```

LOG

Time: May 16, 1980 1:56 PM By: McJones Action: Added default for priority to AllocateNakedNotifyLevel
Time: June 2, 1980 5:50 PM By: McJones Action: Added DeallocateNakedNotifyLevel
Time: July 19, 1980 2:55 PM By: Forrest Action: Added Allocate/DeallocateNakedCondition
Time: September 2, 1980 3:07 PM By: McJones Action: Deleted old-style naked notify items

```
-- ProcessOperations.mesa (last edited by: McJones on: September 3, 1980  5:59 PM)

DIRECTORY
  PrincOps USING [PSBreg, PTCreg, WDCreg],
  Mopcodes USING [
    zBCAST, zDWDC, zME, zMRE, zMXD, zMXW, zNOTIFY, zREQUEUE, zRR, zWR],
  PSB USING [PSB, Handle, Index, Queue];

ProcessOperations: DEFINITIONS =

  BEGIN OPEN PSB;

  QueueHandle: TYPE = LONG POINTER TO Queue;


  Enter: PROCEDURE [POINTER TO MONITORLOCK] RETURNS [success: BOOLEAN] = MACHINE
    CODE BEGIN Mopcodes.zME END;

  Exit: PROCEDURE [POINTER TO MONITORLOCK] = MACHINE CODE BEGIN Mopcodes.zMXD END;

  Wait: PROCEDURE [POINTER TO MONITORLOCK, POINTER TO CONDITION, CARDINAL] =
    MACHINE CODE BEGIN Mopcodes.zMXW END;

  ReEnter: PROCEDURE [POINTER TO MONITORLOCK, POINTER TO CONDITION]
    RETURNS [success: BOOLEAN] = MACHINE CODE BEGIN Mopcodes.zMRE END;

  Notify: PROCEDURE [POINTER TO CONDITION] = MACHINE CODE
    BEGIN Mopcodes.zNOTIFY END;

  Broadcast: PROCEDURE [POINTER TO CONDITION] = MACHINE CODE
    BEGIN Mopcodes.zBCAST END;

  Requeue: PROCEDURE [from: QueueHandle, to: QueueHandle, p: Handle] = MACHINE
    CODE BEGIN Mopcodes.zREQUEUE END;
  -- Note: this depends on having one instruction after enabling:

  EnableAndRequeue: PROCEDURE [QueueHandle, QueueHandle, Handle] = MACHINE CODE
    BEGIN Mopcodes.zDWDC; Mopcodes.zREQUEUE END;

  ReadPSB: PROCEDURE RETURNS [Handle] = MACHINE CODE
    BEGIN Mopcodes.zRR, PrincOps.PSBreg END;

  WritePSB: PROCEDURE [Handle] = MACHINE CODE
    BEGIN Mopcodes.zWR, PrincOps.PSBreg END;

  HandleToIndex: PROCEDURE [handle: Handle] RETURNS [Index] = INLINE {
    RETURN[LOOPHOLE[handle, CARDINAL]/SIZE[PSB]]};

  IndexToHandle: PROCEDURE [index: Index] RETURNS [Handle] = INLINE {
    RETURN[LOOPHOLE[index*SIZE[PSB]]]};

  ReadPTC: PROCEDURE RETURNS [CARDINAL] = MACHINE CODE
    BEGIN Mopcodes.zRR, PrincOps.PTCreg END;

  WritePTC: PROCEDURE [CARDINAL] = MACHINE CODE
    BEGIN Mopcodes.zWR, PrincOps.PTCreg END;

  ReadWDC: PROCEDURE RETURNS [CARDINAL] = MACHINE CODE
    BEGIN Mopcodes.zRR, PrincOps.WDCreg END;

  WriteWDC: PROCEDURE [CARDINAL] = MACHINE CODE
    BEGIN Mopcodes.zWR, PrincOps.WDCreg END;

  END.
```

```
DIRECTORY
    Environment USING [PageCount, PageNumber],
    MiscAlpha USING [aRCLK, aSETMP],
    Mopcodes USING [zMISC];

ProcessorFace: DEFINITIONS =

BEGIN

    -- Initialization
Start: PROCEDURE;
    -- Initialize the implementation of ProcessorFace.  After Start has been called all exported variables in this interface are defined.

    -- Maintenance panel
SetMP: PROCEDURE [CARDINAL] =
    MACHINE CODE BEGIN Mopcodes.zMISC, MiscAlpha.aSETMP END; -- the truth is in the Principles of Operation
    -- Set maintenance panel display to given value.

    -- Processor id
    -- At any moment in time, no two processors will have the same processor id.  The processor id remains constant as long as a
    system element remains running, but may change across system restarts.  Programs should make no assumptions about the
    structure of a processor id.
ProcessorID: TYPE = PRIVATE RECORD [a, b, c: UNSPECIFIED];
processorID: READONLY ProcessorID;
    -- Note: processorID = [0, 0, 0] indicates that the processor has not had a unique processor id assigned.  This is likely to prevent
    many client systems (e.g. Pilot) from running.

    -- Real memory configuration
    -- Some processors may have dedicated real memory not placed in virtual memory at initialization.  Various device face
    implementations will use, and perhaps place into client-provided virtual memory, this real memory.
dedicatedRealMemory: READONLY Environment.PageCount;
    -- Amount of dedicated real memory, if any.  This may be useful, for example, in allocating secondary storage to hold a snapshot of
    the processor state.  Note: dedicatedRealMemory remains constant independent of whether the dedicated memory is in use or not.

    -- Virtual memory layout
GetNextAvailableVM: PROCEDURE [page: Environment.PageNumber]
    RETURNS [firstPage: Environment.PageNumber, count: Environment.PageCount];
    -- Return next area of virtual memory, beginning at or after given page, which is available on this processor.  Return a count of zero
    if no such available area exists.  The available areas exclude pages used by device face implementations and unimplemented page
    map entries but not pages allocated in the Principles of Operation (e.g. process data area, AV, SD, GFT).

    -- Greenwich mean time
    -- A greenwich mean time t represents the time which is t-gmtEpoch (mod 2^32) seconds after midnight, 1 January 1968, where
    gmtEpoch (defined below) is the number of seconds between midnight, 1 January 1968, and midnight, 1 January 1901.  This
    representation was chosen because it assigns the same bit pattern to times as does the Alto time standard for times between 1968
    and 2037 but provides an additional 67 years of range.
    -- NOTE: Before operating on greenwich mean times (e.g. converting to external representation, comparing for other than
    equality), it is best to convert them to "seconds since epoch", by subtracting gmtEpoch.
GreenwichMeanTime: TYPE = LONG CARDINAL;
gmtEpoch: LONG CARDINAL = 2114294400;
    -- (67years*365daysPerYear + 16leapDays)*24hoursPerDay*60minutesPerHour*60secondsPerMinute
GetGreenwichMeanTime: PROCEDURE RETURNS [GreenwichMeanTime];
    -- Return the current value of the system's greenwich mean time clock.  This clock runs continuously and, once set, maintains
    reasonable accuracy.
    -- NOTE: If the clock has not been set, GetGreenwichMeanTime returns gmtEpoch.
SetGreenwichMeanTime: PROCEDURE [GreenwichMeanTime];
    -- Set the system's greenwich mean time clock to the given value.

    -- Interval time
    -- The interval timer is 32 bits long, and is incremented by one at a constant rate, but this rate may vary from processor to
    processor.  It should be somewhere between 10^4/second and 10^6/second, so that reasonable assumptions about precision and
    period may be made by timing software.
    -- NOTE: The interval timer is considered to be unsigned, so one may measure an interval up to 2^32 pulses simply by subtracting
    initial from final timer reading, with no concern for wrap-around.
GetClockPulses: PROCEDURE RETURNS [LONG CARDINAL] =
    MACHINE CODE BEGIN Mopcodes.zMISC, MiscAlpha.aRCLK END; -- the truth is in the Principles of Operation
    -- Return the current value of the interval timer.
microsecondsPerHundredPulses: READONLY CARDINAL; -- max error is .33%
```

```
        -- Naked notifies
reservedNakedNotifyMask: READONLY WORD;
    -- A bit-mask with a one in each position corresponding to a naked-notify "channel" used for some private purpose below the
    Principles of Operation and therefore not available to normal programs.

        -- Condition variable time
    -- A timeout value may be specified for any condition variable, in units of "ticks". The size of a tick should be between about 15 and
    50 milliseconds.
millisecondsPerTick: READONLY CARDINAL;

        -- Booting and power control
BootButton: PROCEDURE;
    -- Simulate pressing the processor "boot button".
PowerOff: PROCEDURE;
    -- If the processor is suitably equipped, turn off electrical power to the system.
    -- NOTE: On processors not equipped with a power control relay, this may be implemented as disabling interrupts, turning off
    devices using DeviceCleanup.Perform[turnOff], and spinning.
SetAutomaticPowerOn: PROCEDURE [gmt: GreenwichMeanTime, externalEvent: BOOLEAN];
    -- If the processor is suitably equipped, cause system electrical power to be turned on at or after the given time: if externalEvent is
    FALSE, turn power on at the given time; if externalEvent is TRUE, turn power on in response to the first external event occurring after
    the given time.
    -- NOTE: On processors not equipped with an independently powered alarm clock connected to a power control relay, this may be
    implemented by causing a subsequent PowerOff to wait for the appropriate condition and then execute BootButton.
ResetAutomaticPowerOn: PROCEDURE;
    -- Cancel a previous SetAutomaticPowerOn.

END.
```

LOG

Time: August 7, 1979 12:05 PM By: Redell Action: Create file from White's OISProcFace; replace DeviceNumber machinery with DeviceHandles and GetNextDevice
Time: January 30, 1980 2:16 PM By: McJones Action: Delete all but GreenwichMeanTime; add maintenance panel, process id, virtual memory layout, greenwich mean time, interval time, and power control facilities
Time: May 14, 1980 3:59 PM By: McJones Action: Add Start, reservedNakedNotifyMask
Time: May 30, 1980 10:00 AM By: McJones Action: Change ProcessorID from 32 to 48 bits; drop OIS from interface name
Time: August 12, 1980 10:01 AM By: McJones Action: Add dedicatedRealMemory; use MiscAlpha

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

Projection: DEFINITIONS =

BEGIN

DeleteSwapUnits: PROCEDURE [pageMember: VM.PageNumber];
    -- Set region to hasSwapUnits = FALSE.
    -- NOTE: modifies projection file only, does not update region cache; thus, the region desc must be flushed from the cache first.

ForceOut: PROCEDURE [pageMember: VM.PageNumber];
    -- If descriptor for projection region containing pageMember is cached and dirty, clean it.

Get: PROCEDURE [pageMember: VM.PageNumber] RETURNS [CachedRegion.Desc];
    -- Returns descriptor for projection region containing pageMember. Also, inserts it in the region cache.

Merge: PROCEDURE [pageNext: VM.PageNumber];
    -- Removes the projection region boundary at pageNext.
    -- NOTE: modifies projection file only, does not update region cache; thus, the region desc must be flushed from the cache first.

Split: 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; thus, the region desc must be flushed from the cache first.

Touch: PROCEDURE [pageMember: VM.PageNumber];
    -- Ensure descriptor for projection region containing pageMember is cached

TranslateLevel: 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; thus, the region desc must be flushed from the cache first.

END.
```

LOG

Time: Sometime in April 1978	*By: McJones*	*Action: Created file*
Time: June 24, 1978 5:26 PM	*By: McJones*	*Action: PageNumber moved to VM*
Time: September 28, 1979 3:50 PM	*By: Knutsen*	*Action: Added documentation.*
Time: February 25, 1980 3:50 PM	*By: Knutsen*	*Action: Added DeleteSwapUnits.*

```
-- PSB.mesa (last edited by: McJones on: September 15, 1980  3:32 PM)

DIRECTORY
  PrincOps USING [ControlLink, FrameHandle];

PSB: DEFINITIONS =
  BEGIN

  MonitorLock: TYPE = MACHINE DEPENDENT RECORD [
    reserved1 (0:0..3): [0..17B] ← 0,
    tail (0:4..13): Index,
    reserved2 (0:14..14): [0..1] ← 0,
    lock (0:15..15): {unlocked, locked}];

  MonitorHandle: TYPE = POINTER TO MonitorLock;
  LockedEmpty: MonitorLock = [tail: Empty, lock: locked];
  UnlockedEmpty: MonitorLock = [tail: Empty, lock: unlocked];

  Condition: TYPE = MACHINE DEPENDENT RECORD [
    reserved (0:0..3): [0..17B] ← 0,
    tail (0:4..13): Index,
    enableAborts (0:14..14): BOOLEAN,
    wakeup (0:15..15): BOOLEAN,
    timeout (1:0..15): Ticks];

  ConditionHandle: TYPE = POINTER TO Condition;

  Index: TYPE = CARDINAL [0..1024);
  nullIndex: Index = 0;

  PSB: TYPE = MACHINE DEPENDENT RECORD [
    link (0): Link,
    flags (1): Flags,
    state (2): PrincOps.FrameHandle,
    timeout (3): Ticks];

  Link: TYPE = MACHINE DEPENDENT RECORD [
    enterFailed (0:0..0): BOOLEAN,
    priority (0:1..3): Priority,
    next (0:4..13): Index,
    reserved (0:14..14): [0..1] ← 0,
    stateVector (0:15..15): BOOLEAN];

  Flags: TYPE = MACHINE DEPENDENT RECORD [
    state (0: 0..1): MACHINE DEPENDENT {frameReady, frameTaken, dead, alive},
    reserved (0: 2..3): [0..3] ← 0,
    cleanup (0:4..13): Index,
    detached (0:14..14): BOOLEAN,
    abortPending (0:15..15): BOOLEAN];

  Priority: TYPE = [0..7];

  Ticks: TYPE = CARDINAL;

  Empty: Index = FIRST[Index];

  StateHandle: TYPE = POINTER TO StateVector;
  StateVector: TYPE = MACHINE DEPENDENT RECORD [
    stack (0): ARRAY [0..8) OF UNSPECIFIED,
    stkptr (8): CARDINAL,
    frame (9): PrincOps.ControlLink,
    parm (10): ARRAY [0..2) OF UNSPECIFIED];

  PDABase: TYPE = LONG BASE POINTER TO ProcessDataArea;

  PDA: PDABase = LOOPHOLE[200000B];

  Handle: TYPE = PDABase RELATIVE POINTER TO PSB;
  nullHandle: Handle = LOOPHOLE[0];

  ProcessDataArea: TYPE = MACHINE DEPENDENT RECORD [
    vp (0): SELECT OVERLAID * FROM
      header => [
        ready (0): Queue,
        count (1): CARDINAL,
        currentState (2): PDABase RELATIVE POINTER TO StateVector,
        reserved1 (3): UNSPECIFIED ← 0,
        externalStateVector (4): LONG POINTER,
```

```
        -- TO CPSwapDefs.ExternalStateVector (debugger communication)
      data2 (6): LONG POINTER, -- reserved for software
      interrupt (10B): ConditionVector,
      states (50B): ARRAY Priority OF StateVector],
    blocks => [block (0): ARRAY [0..0) OF PSB],
    ENDCASE];

assertion: CARDINAL [0..0] = SIZE[ProcessDataArea] MOD SIZE[PSB];

StartPsb: Index = SIZE[ProcessDataArea]/SIZE[PSB];

ConditionVector: TYPE = ARRAY [0..16) OF Condition;

Queue: TYPE = MACHINE DEPENDENT RECORD [
  reserved1 (0:0..3): [0..17B] ← 0,
  tail (0:4..13): Index,
  reserved2 (0:14..15): [0..3] ← 0];

END.
```

```
-- File: PupDefs.Mesa,  Last Edit: HGM  August 12, 1980  3:28 PM

DIRECTORY
  BufferDefs USING [PupBuffer, Queue, QueueObject, RppBuffer],
  PupTypes USING [
    maxDataWordsPerGatewayPup, Pair, PupType,
    PupAddress, PupErrorCode, PupHostID, PupNetID, PupSocketID];

PupDefs: DEFINITIONS = BEGIN

Byte: TYPE = [0..377B];

-- TYPEs Copied from BufferDefs
PupBuffer: TYPE = BufferDefs.PupBuffer;
RppBuffer: TYPE = BufferDefs.RppBuffer;
Queue: TYPE = BufferDefs.Queue;
QueueObject: TYPE = BufferDefs.QueueObject;

-- TYPEs Copied things from PupTypes
PupAddress: TYPE = PupTypes.PupAddress;
PupNetID: TYPE = PupTypes.PupNetID;
PupHostID: TYPE = PupTypes.PupHostID;
PupSocketID: TYPE = PupTypes.PupSocketID;
Pair: TYPE = PupTypes.Pair;


-- Buffer/Queue procedures
EnqueuePup: PROCEDURE [Queue, PupBuffer];
DequeuePup: PROCEDURE [Queue] RETURNS [PupBuffer];
GetFreePupBuffer: PROCEDURE RETURNS [PupBuffer];
ReturnFreePupBuffer: PROCEDURE [PupBuffer];
EnqueueRpp: PROCEDURE [Queue, RppBuffer];
DequeueRpp: PROCEDURE [Queue] RETURNS [RppBuffer];
GetFreeRppBuffer: PROCEDURE RETURNS [RppBuffer];
ReturnFreeRppBuffer: PROCEDURE [RppBuffer];
GetClumpOfPupBuffers: PROCEDURE [q: Queue, n: CARDINAL, wait: BOOLEAN];
GetClumpOfRppBuffers: PROCEDURE [q: Queue, n: CARDINAL, wait: BOOLEAN];
ExtractPupFromQueue: PROCEDURE [Queue, PupBuffer] RETURNS [PupBuffer];
ExtractRppFromQueue: PROCEDURE [Queue, RppBuffer] RETURNS [RppBuffer];


-- Pup Routing
InspectStrayPups: PROCEDURE [
    on: BOOLEAN, seeBroadcast: BOOLEAN,
    proc: PROCEDURE [PupBuffer] RETURNS [error: BOOLEAN] ];
PupRouterBroadcastThis: PROCEDURE [PupBuffer];
PupRouterSendThis: PROCEDURE [PupBuffer];

-- Start of things like this
GetHopsToNetwork: PROCEDURE [PupNetID] RETURNS [CARDINAL];  -- returns a big number if can't get there
FastPath: PROCEDURE [PupAddress] RETURNS [BOOLEAN];

-- default parameters collected here
defaultNumberOfNetworks: CARDINAL = 84;  -- one page from the heap
defaultDataWordsPerPup: CARDINAL = PupTypes.maxDataWordsPerGatewayPup;
defaultPupsToAllocate: CARDINAL = 5;  -- allocates for PktStreams


-- Misc utilities
GetLocalPupAddress: PROCEDURE [local: PupSocketID, remote: POINTER TO PupAddress] RETURNS [PupAddress];
UniqueLocalPupSocketID: PROCEDURE RETURNS [PupSocketID];
UniqueLocalPupAddress: PROCEDURE [POINTER TO PupAddress] RETURNS [PupAddress];
SwapPupSourceAndDest: PROCEDURE [PupBuffer];  -- fixes up broadcast too

SendPup: PROCEDURE [b: PupBuffer, type: PupTypes.PupType, bytes: CARDINAL];
ReturnPup: PROCEDURE [b: PupBuffer, type: PupTypes.PupType, bytes: CARDINAL];


-- Timeout things
Tocks: TYPE = RECORD[CARDINAL];
  veryLongWait: Tocks = [LAST[CARDINAL]];
  veryShortWait: Tocks = [0];
SecondsToTocks: PROCEDURE [CARDINAL] RETURNS [Tocks];
MsToTocks: PROCEDURE [CARDINAL] RETURNS [Tocks];

-- PupBuffer level utilities
GetPupContentsBytes: PROCEDURE [PupBuffer] RETURNS [CARDINAL];
SetPupContentsBytes: PROCEDURE [PupBuffer, CARDINAL];
```

```
SetPupContentsWords: PROCEDURE [ PupBuffer, CARDINAL ];
DataWordsPerPupBuffer: PROCEDURE RETURNS [CARDINAL];

-- Socket Interface
PupSocketMake: PROCEDURE [
    local: PupSocketID, remote: PupAddress, ticks: Tocks, id: Pair ← [0,0]]
  RETURNS [PupSocket];
PupSocketKick: PROCEDURE [PupSocket];
PupSocketDestroy: PROCEDURE [PupSocket];
PupSocket: TYPE = POINTER TO PupSocketObject;
PupSocketObject: TYPE = RECORD [
  put: PROCEDURE [PupBuffer],
  get: PROCEDURE RETURNS [PupBuffer],  -- if none, wait, then return NIL
  setRemoteAddress: PROCEDURE [PupAddress],
  getLocalAddress: PROCEDURE RETURNS [PupAddress] ];


-- Start and Stop
AdjustBufferParms: PROCEDURE [bufferPoolSize, bufferSize: CARDINAL];
GetBufferParms: PROCEDURE RETURNS [bufferPoolSize, bufferSize: CARDINAL];
PupPackageMake: PROCEDURE;
PupPackageReady: PROCEDURE;
PupPackageDestroy: PROCEDURE;
GetPupPackageUseCount: PROCEDURE RETURNS [CARDINAL];
UseAltoChecksumMicrocode: PROCEDURE;

-- NameConversion and STRING diddling
GetPupAddress: PROCEDURE [POINTER TO PupAddress, STRING];
PupNameLookup: PROCEDURE [POINTER TO PupAddress, STRING];
  NameLookupErrorCode: TYPE = {noRoute, noResponse, errorFromServer};  -- if error
ParsePupAddressConstant: PROCEDURE [POINTER TO PupAddress, STRING] RETURNS [BOOLEAN];
EnumeratePupAddresses: PROCEDURE [
    s: STRING, filter: PROCEDURE [PupAddress] RETURNS [BOOLEAN] ]
  RETURNS [hit: BOOLEAN];
PupAddressLookup: PROCEDURE [STRING, PupAddress];
AppendPupAddress: PROCEDURE [STRING, PupAddress];
AppendHostName: PROCEDURE [STRING, PupAddress];
AppendMyName: PROCEDURE [STRING];
MoveStringBodyToPupBuffer: PROCEDURE [PupBuffer, STRING];
AppendStringBodyToPupBuffer: PROCEDURE [PupBuffer, STRING];
AppendErrorPup: PROCEDURE [STRING, PupBuffer];

-- Does SwapPupSourceAndDest and such, then PupRouterSendThis
SendErrorPup: PROCEDURE [PupBuffer, PupTypes.PupErrorCode, STRING];


-- ERRORs
PupNameTrouble: ERROR [e: STRING, code: NameLookupErrorCode];
TockConversionTroubles: ERROR;

-- Debugging
SetLocalOnly: PROCEDURE [BOOLEAN];
SetPupCheckit: PROCEDURE [BOOLEAN];
SetPupStormy: PROCEDURE [BOOLEAN];  -- lightning in Router
SetErrorSuppression: PROCEDURE [dontSendErrors: BOOLEAN];
SetBufferFullSuppression: PROCEDURE [dontSend: BOOLEAN];
GetDoStats: PROCEDURE RETURNS [BOOLEAN];
CaptureErrors: PROCEDURE [PROCEDURE [ERROR]];

InspectIncomingPups: PROCEDURE [BOOLEAN, PROCEDURE[CARDINAL,PupBuffer]];
InspectOutgoingPups: PROCEDURE [BOOLEAN, PROCEDURE[CARDINAL,PupBuffer]];
ShowPupBuffer: PROCEDURE [CARDINAL, PupBuffer];  -- in PupShow.bcd
  incomingPup: CARDINAL = 100;  -- code numbers for Show
  outgoingPup: CARDINAL = 101;
  zappedIncomingPup: CARDINAL = 102;
  zappedOutgoingPup: CARDINAL = 103;

END.
```

```
-- File: PupTypes.Mesa,  Last Edit: HGM  May 28, 1980  8:28 PM

PupTypes: DEFINITIONS =
BEGIN

Byte: TYPE = [0..377B];
Pair: TYPE = MACHINE DEPENDENT RECORD [a, b: CARDINAL];

PupHostID: TYPE = RECORD [Byte];
PupNetID: TYPE = RECORD [Byte];
PupSocketID: TYPE = RECORD [ a, b: WORD ];

PupAddress: TYPE = MACHINE DEPENDENT RECORD [
  net: PupNetID, host: PupHostID, socket: PupSocketID ];


RppAddress: TYPE = MACHINE DEPENDENT RECORD [
  net: PupNetID, host: PupHostID, socket: WORD, connectionID: WORD ];

-- Addressing Constants
fillInNetID: PupNetID = [377B];
fillInHostID: PupHostID = [377B];
fillInSocketID: PupSocketID = [0,0];
fillInPupAddress: PupAddress = [fillInNetID,fillInHostID,fillInSocketID];
allNets: PupNetID = [0];  -- host must be allHosts too
allHosts: PupHostID = [0];  -- can be directed to a specific network

-- Buffer size Constants
maxDataWordsPerGatewayPup: CARDINAL = 266;
maxDataBytesPerGatewayPup: CARDINAL = 2*maxDataWordsPerGatewayPup;
maxDataWordsPerRoutingPup: CARDINAL = 64;
maxDataBytesPerRoutingPup: CARDINAL = 2*maxDataWordsPerRoutingPup;


-- Well Known Sockets:  See [MAXC]<System>Pup-Network.txt
  telnetSoc: PupSocketID = [0,1];
  gatewaySoc: PupSocketID = [0,2];
  ftpSoc: PupSocketID = [0,3];
  miscSrvSoc: PupSocketID = [0,4];
  echoSoc: PupSocketID = [0,5];
  bspTestSoc: PupSocketID = [0,6];
  mailSoc: PupSocketID = [0,7];
  eftpReceiveSoc: PupSocketID = [0,20B];
  spruceStatusSoc: PupSocketID = [0,21B];
  statSoc: PupSocketID = [0,22B];
  oldCopyDiskSoc: PupSocketID = [0,24B];
  copyDiskSoc: PupSocketID = [0,25B];
  eventReportSoc: PupSocketID = [0,30B];
  printerReportSoc: PupSocketID = [0,31B];
  juniperpackConversionSoc: PupSocketID = [0,34B];
  juniperEventSoc: PupSocketID = [0,35B];
  rpcpSoc: PupSocketID = [0,36B];
  clearinghouseSoc: PupSocketID = [0,37B];
  librarianSoc: PupSocketID = [0,41B];
  pineSoc: PupSocketID = [0,100B];

PupErrorCode: TYPE = MACHINE DEPENDENT {
  -- Pup got to the destination machine, but wasn't processed
  badChecksumPupErrorCode(1B),
  noProcessPupErrorCode(2B),
  resourceLimitsPupErrorCode(3B),
  -- Pup didn't get to the destination machine
  inconsistentPupErrorCode(1001B),
  cantGetTherePupErrorCode(1002B),
  hostDownPupErrorCode(1003B),
  eightHopsPupErrorCode(1004B),
  tooBigPupErrorCode(1005B),
  iAmNotAGatewayPupErrorCode(518),
  gatewayResourceLimitsPupErrorCode(519),

  -- used by ForwardBuffer for various hacks
  noErrorPupErrorCode(10000),
  connectionLimitPupErrorCode(10001),

  filler(LAST[WORD]) };
```

```
-- This is an attempt to get the TYPE checker to help us.
-- Unfortunately, the values are a bit sparce.
-- Be sure to have enough values to make it an 8 bit field.

PupType: TYPE = MACHINE DEPENDENT {

-- 000-077 OCTAL!       Registered Pup Types
echoMe(1B), iAmEcho, badEcho,
error(4B),
rfc(10B), abort, end, endRep,
data(20B), aData, ack, mark, int, intRep, aMark,
eData(30B), eAck, eEnd, eAbort,
rpp(40B),

-- 200+         Unregistered Pup Types
-- Socket 2 - Gateway info
gatewayRequest(200B), gatewayInfo(201B),

-- Socket 4 - Misc services
dateTenexRequest(202B), dateTenexIs,
dateAltoRequest(206B), dateAltoIs,

mailCheck(210B), mailIsNew, mailNotNew, mailError, mailCheckLaurel,
nameLookup(220B), nameIs, nameError, addressLookup, addressIs,
whereIsUser(230B), userIs, userError,
netDirVersion(240B), sendNetDir,
bootFileSend(244B), kissOfDeath(247B),

-- Pine  (save a few spares too)
request(250B), result, unsolicited, custodian, sync, pineAck, noop,

bootDirReq(257B), bootDirReply(260B),

last(LAST[Byte]) };


-- see <PUP>Servers.EARS (Taft)

-- Socket 4 Misc Services - these overlap with Gateway info
dateTextRequest: PupType = gatewayRequest;
dateTextIs: PupType = gatewayInfo;

-- Socket 4 Misc Services - these overlap with Pine
userAuthReq: PupType = request;  -- 250
userAuthOk: PupType = result;  -- 251
userAuthBad: PupType = unsolicited;  -- 252

-- Socket 22 Statistics - more overlap
statisticsRequest: PupType = gatewayRequest;
statisticsAre: PupType = gatewayInfo;

-- Socket 30 Event Report - more overlap
eventReport: PupType = netDirVersion;
eventReportReply: PupType = sendNetDir;

END.
```

```
-- RDC.mesa (last edited by: McJones on: July 18, 1980  5:25 PM)

DIRECTORY
  Environment USING [Base],
  File USING [Type],
  PilotDisk USING [nullAddress],
  SA4000Face USING [DiskAddress, Operation];

RDC: DEFINITIONS =
BEGIN

-- RDC I/O Command Block
IOCB: TYPE = MACHINE DEPENDENT RECORD [ -- 16-word aligned
  operation (0): SA4000Face.Operation,
  serviceLateRetryCount (15B): CARDINAL, -- microcode retry count
  rdcCommand (16B): WORD,  -- command to be executed by the RDC
  next (17B): IOCBshortPtr, -- points to next IOCB in the queue, set by Initiate
  clientHeader (20B): SA4000Face.DiskAddress,
  dontcare (22B): RECORD [a, b: UNSPECIFIED], -- clobbered by disk microcode
  clientLabel (24B): PackedLabel, -- input to verify and write label operations
  diskLabel (34B): PackedLabel, -- output from verify and read label operations
  rateErrorRetryCount (44B): CARDINAL]; -- microcode retry count

IOCBshortPtr: TYPE = Environment.Base RELATIVE POINTER TO IOCB;
IOCBlongPtr: TYPE = LONG POINTER TO IOCB;

-- Restore the explicit bit positions in the following declaration as soon as Mesa AR 4670 is fixed:
PackedLabel: TYPE = MACHINE DEPENDENT RECORD [
  sequence0Bit2 --(0:0..0)--: CARDINAL[0..2),
  processorID0 --(0:1..15)--: CARDINAL[0..32768),
  processorID1 --(1)--: CARDINAL,
  processorID2 --(2)--: CARDINAL,
  sequence1 --(3)--: CARDINAL,
  sequence0Bits3Thru15 --(4:0..12)--: CARDINAL[0..8192),
  immutable --(4:13..13)--: BOOLEAN, -- clear after first (or every) page in run
  temporary --(4:14..14)--: BOOLEAN, -- clear after first (or every) page in run
  zeroSize --(4:15..15)--: BOOLEAN, -- clear after first (or every) page in run
  filePageLo --(5)--: CARDINAL, -- increment after every page in run
  type --(6)--: File.Type,
  vp --(7)--: SELECT OVERLAID * FROM
    volumeFile => [
      fill: CARDINAL[0..512) ← 0, -- delete when Mesa AR 4670 is fixed
      filePageHi --(7)--: CARDINAL[0..128)],
    normalFile => [bootChainLink --(7)--: PackedDiskAddress],
    ENDCASE];

PackedDiskAddress: TYPE = MACHINE DEPENDENT RECORD [
  cylinder (0:0..7): [0..256), -- =fixedHeadCylinder => next field is a fixed head
  head (0:8..10): [0..8),
  sector (0:11..15): [0..32)];

eof: PackedDiskAddress = LOOPHOLE[-1]; -- cylinder=255;
fixedHeadCylinder: CARDINAL = 254;
firstFixedHead: CARDINAL = 8;

PackDiskAddress: PROCEDURE [da: SA4000Face.DiskAddress] RETURNS [PackedDiskAddress] =
  INLINE BEGIN
  IF da=LOOPHOLE[PilotDisk.nullAddress] THEN
    RETURN[eof]
  ELSE IF da.head>=firstFixedHead THEN
    RETURN[[cylinder: fixedHeadCylinder, head: da.head-firstFixedHead, sector: da.sector]]
  ELSE
    RETURN[[cylinder: da.cylinder, head: da.head, sector: da.sector]]
  END;

UnpackDiskAddress: PROCEDURE [pda: PackedDiskAddress] RETURNS [SA4000Face.DiskAddress] =
  INLINE BEGIN
  IF pda=eof THEN
    RETURN[LOOPHOLE[PilotDisk.nullAddress]]
  ELSE IF pda.cylinder=fixedHeadCylinder THEN
    RETURN[[cylinder: 0, head: pda.head+firstFixedHead, sector: pda.sector]]
  ELSE
    RETURN[[cylinder: pda.cylinder, head: pda.head, sector: pda.sector]]
  END;

DeviceStatus: TYPE = MACHINE DEPENDENT RECORD [
  outcome (0: 0..3): Outcome,
  rdcBits (0: 4..15): RDCStatusBits];
```

```
Outcome: TYPE = MACHINE DEPENDENT {inProgress (0), goodCompletion, dataError, labelError,
  labelCheck, seekTimeout, sectorTimeout, notReady, headerError};

RDCStatusBits: TYPE = MACHINE DEPENDENT RECORD [
  unused (0:0..0),
  serviceLate (0:1..1),
  one (0:2..2), -- should always be 1
  rateError (0:3..3),
  sector00S (0:4..4),
  track00S (0:5..5),
  seekCompS (0:6..6),
  devSelOK (0:7..7),
  bufErr (0:8..8),
  rdErr (0:9..9),
  writeFault (0:10..10),
  oFault (0:11..11): BOOLEAN];

END.

LOG
Time: November 19, 1979  6:32 PM        By: Frandeen    Action: Create file
Time: June 3, 1980  10:22 AM    By: McJones     Action: 48-bit processor ids
```

```
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
    Environment: FROM "Environment" USING [Base,first64K],
    Zone: FROM "Zone" USING [Alignment, BlockSize, Handle, Status];

ResidentHeap: DEFINITIONS =
BEGIN

    -- This interface provides a small object allocator for resident (i.e., non-swappable) storage in the first 64K of VM. Objects can
        be accessed via 16-bit relative pointers.

    -- Alignment of the objects allocated is specified by the following type
Alignment: TYPE = Zone.Alignment;    -- = {a1 (word aligned), a2 (double word aligned), a4 (quad word aligned), a8 (eight-word aligned), a16 (16-
        word aligned)}

    -- The following is the base pointer to the first 64K of virtual memory
first64K: Environment.Base = Environment.first64K; -- for upward compatibility

    -- The following provides access to the zone (not normally needed)
residentZone: READONLY Zone.Handle;

    -- The following procedures manage the resident heap
MakeNode: PROCEDURE [n: Zone.BlockSize, alignment: Alignment ← a1] RETURNS [node: Environment.Base RELATIVE POINTER, s:
        Zone.Status];
FreeNode: PROCEDURE [p: Environment.Base RELATIVE POINTER] RETURNS [s: Zone.Status];
SplitNode: PROCEDURE [p: Environment.Base RELATIVE POINTER, n: Zone.BlockSize] RETURNS [s: Zone.Status];
NodeSize: PROCEDURE [p: Environment.Base RELATIVE POINTER] RETURNS [n: Zone.BlockSize];

END.
```

LOG

Time: May 31, 1979 2:56 PM By: Lauer Action: Created file
Time: July 18, 1979 2:54 PM By: Knutsen Action: Promoted Base *and* first64K *into* Environment.
Time: timeStamp By: yourName Action: shortDescription

```
-- Some ResidentMemory procedures are involved in recovering from the frame heap becoming exhausted.  Because of this, invoking
   these procedures must not cause any frame allocations.  This means that these procedures (and any that they call) must be INLINEs or
   coroutines.  Since there is no such thing as an ENTRY coroutine, it must be simulated by an ENTRY INLINE procedure (which acquires
   the monitor lock) which itself calls the coroutine.  To have an ENTRY INLINE PROCEDURE, the monitor lock must be available in the
   DEFINITIONS module.  To allow a coroutine to be called as a public procedure, the procedure descriptor must be bundled into a record
   to force it to be a procedure variable.

DIRECTORY
    Environment: FROM "Environment" USING [PageCount];

ResidentMemory: DEFINITIONS
    LOCKS residentMemoryLock =

BEGIN OPEN Environment;

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


--  Allocation of resident memory for I/O and miscellaneous use:

Allocate: ENTRY PROCEDURE [where: Location, pages: PageCount] RETURNS [lp: LONG POINTER TO UNSPECIFIED] = INLINE
    -- Guaranteed not to do an ALLOC from the frame heap.
    BEGIN RETURN[allocateInternal[where, pages]] END;

Free: PROCEDURE [where: Location, pages: PageCount, lp: LONG POINTER TO UNSPECIFIED];


--  Allocation of resident memory within the MDS:

AllocateMDS: ENTRY PROCEDURE [pages: PageCount] RETURNS [p: POINTER TO UNSPECIFIED] = INLINE
    -- Guaranteed not to do an ALLOC from the frame heap.
    BEGIN RETURN[allocateMDSInternal[pages]] END;

FreeMDS: PROCEDURE [base: POINTER TO UNSPECIFIED, pages: PageCount] = INLINE
    BEGIN Free[mds, pages, LONG[base]] END;


residentMemoryLock: PRIVATE MONITORLOCK;

AllocateInternal: PRIVATE TYPE = RECORD[
    proc: PROCEDURE [where: Location, pages: PageCount] RETURNS [lp: LONG POINTER TO UNSPECIFIED]];
allocateInternal: PRIVATE AllocateInternal;

AllocateMDSInternal: PRIVATE TYPE = RECORD[
    proc: PROCEDURE [pages: PageCount] RETURNS [p: POINTER TO UNSPECIFIED]];
allocateMDSInternal: PRIVATE AllocateMDSInternal;

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"*

Time: January 25, 1980 2:33 PM — *By: Knutsen* — *Action: Named return parameters.*

Time: April 16, 1980 8:47 AM — *By: Knutsen* — *Action: Expanded Location to include MDS. Added Free. Renamed to AllocateMDS, FreeMDS. Added* residentMemoryLock, *made Allocate, AllocateMDS ENTRY INLINEs.*

```
-- Function: The internal definitions module for the Pilot OISCP Router.

DIRECTORY
 DriverDefs USING [DriverXmitStatus, Network],
 OISCPDefs USING [OisAddress, OisBuffer, OisHostID, OisNetID],
 SocketInternal USING [SocketHandle],
 SpecialCommunication USING [RoutersFunction],
 SpecialSystem USING [NetworkNumber];

Router: DEFINITIONS =
BEGIN OPEN OISCPDefs;

-- TYPEs and definitions

SocketTable: TYPE = RECORD [length: CARDINAL, first: SocketInternal.SocketHandle];
Nibble: TYPE = [0..15B];
RoutingTableEntry: TYPE = LONG POINTER TO RoutingTableObject;
RoutingTableObject: TYPE = RECORD [
 next: RoutingTableEntry,
 destNetwork: SpecialSystem.NetworkNumber, -- ultimate destination network num.
 delay: Nibble,
 timeUnits: Nibble, -- if zero then this entry is discarded
 route: OisHostID, -- if = unknownHostID then the ultimate destination network is local.
 network: DriverDefs.Network]; -- use this network driver to reach route.
-- format of routing tables in routing information packets from OIS internetwork routers
RoutingInfoTuple: TYPE = MACHINE DEPENDENT RECORD [
 objectNetID: OisNetID, interrouterDelay: CARDINAL];
routingInfoRequest: CARDINAL = 1;
routingInfoResponse: CARDINAL = 2;
XmitStatus: TYPE = DriverDefs.DriverXmitStatus;

-- variables
socketRouterLock: MONITORLOCK; -- lock shared by SocketImpl and RouterImpl.
primaryMDS: BOOLEAN; -- whether the router is in the primary MDS or not.
routersFunction: SpecialCommunication.RoutersFunction; -- whether the router is an internet router or not.
checkIt: BOOLEAN; -- whether checksums are on or not.

-- interface
-- Procedures from RouterImpl
OisRouterOn: PROCEDURE;
OisRouterOff: PROCEDURE;
AssignOisAddress: PROCEDURE RETURNS [OisAddress];
AssignDestinationRelativeOisAddress: PROCEDURE [OisNetID] RETURNS [OisAddress];
AddSocket: PROCEDURE [SocketInternal.SocketHandle];
RemoveSocket: PROCEDURE [SocketInternal.SocketHandle];
SendPacket: PROCEDURE [OisBuffer];
ReceivePacket: PROCEDURE [OisBuffer];
SendBroadcastPacket: PROCEDURE [OisBuffer];
FindMyHostID: PROCEDURE RETURNS [OisHostID];
-- debugging
SetOisCheckit: PROCEDURE [BOOLEAN];
SetOisStormy: PROCEDURE [BOOLEAN];
SetOisDriverLoopback: PROCEDURE [BOOLEAN];
SetOisShowit: PROCEDURE [BOOLEAN];
InspectOutgoingOiss: PROCEDURE[PROCEDURE[OisBuffer]];
InspectIncommingOiss: PROCEDURE[PROCEDURE[OisBuffer]];
CaptureErrors: PROCEDURE [PROCEDURE[UNSPECIFIED]];
-- Procedures from RoutingTableImpl
RoutingTableOn: PROCEDURE;
RoutingTableOff: PROCEDURE;
FindDestinationRelativeNetID: PROCEDURE [OisNetID] RETURNS [OisNetID];
AddNetwork: PROCEDURE [DriverDefs.Network];
RemoveNetwork: PROCEDURE [DriverDefs.Network];
StateChanged: PROCEDURE [DriverDefs.Network];
FindNetworkAndTransmit: PROCEDURE [OisBuffer] RETURNS [XmitStatus];
ForwardPacket: PROCEDURE [OisBuffer];
RoutingInformationPacket: PROCEDURE [OisBuffer];

END. -- Router

LOG
```

Time: January 19, 1980 4:01 PM By: Dalal Action: Created interface.
Time: August 1, 1980 6:51 PM By: BLyon Action: replaced internetRouter by routersFunction and moved checksums stuff to Checksums Interface.
Time: August 4, 1980 6:51 PM By: BLyon Action: added destNetwork to RoutingTableEntry and made the routingTable a linked list.
Time: September 13, 1980 6:09 PM By: HGM Action: added StateChanged.
Time: September 18, 1980 3:31 PM By: BLyon Action: deleted FindPrimaryNetID.

-- LastEdited: August 5, 1980 5:20 PM By: VSS

```
DIRECTORY
    Environment USING [Byte], RS232CEnvironment USING
        [AutoRecognitionOutcome,CharLength,CompletionHandle,Correspondent,LineSpeed,LineType,Parity,PhysicalRecord,Phy
        sicalRecordHandle,StopBits, SyncChar,SyncCount];

RS232C: DEFINITIONS =
BEGIN

-- Interface Definitions
Create: PROCEDURE [lineNumber: CARDINAL, lineType: LineType] RETURNS [ChannelHandle];
Get: PROCEDURE [channel: ChannelHandle, rec: PhysicalRecordHandle] RETURNS [CompletionHandle];
Put: PROCEDURE [channel: ChannelHandle, rec: PhysicalRecordHandle] RETURNS [CompletionHandle];
TransferWait: PROCEDURE [channel: ChannelHandle, event: CompletionHandle] RETURNS [byteCount: CARDINAL, status:
        TransferStatus];
Delete: PROCEDURE [channel: ChannelHandle];
Suspend: PROCEDURE [channel: ChannelHandle, class: OperationClass];
Restart: PROCEDURE [channel: ChannelHandle, class: OperationClass];
GetStatus: PROCEDURE [channel: ChannelHandle] RETURNS [stat: DeviceStatus];
StatusWait: PROCEDURE [channel: ChannelHandle, stat: DeviceStatus] RETURNS [newstat: DeviceStatus];

-- RS232C-specific procedures --
AutoRecognitionWait: PROCEDURE [channel: ChannelHandle] RETURNS [outcome: AutoRecognitionOutcome];
TransmitNow: PROCEDURE [channel: ChannelHandle, event: CompletionHandle] RETURNS [byteCount: CARDINAL, status:
        TransferStatus];
SendBreak: PROCEDURE [channel: ChannelHandle];
SetParameter: PROCEDURE [channel: ChannelHandle, parameter: Parameter];
SetLineType: PROCEDURE [channel: ChannelHandle, lineType: LineType];

-- SIGNALs and ERRORs
ChannelAlreadyExists, ChannelSuspended, InvalidLineNumber, InvalidParameter, NoRS232CHardware,
        SendBreakIllegal, UnimplementedFeature: ERROR;

-- Interface type definitions
LineType: TYPE = RS232CEnvironment.LineType;
ChannelHandle: TYPE[2];
AutoRecognitionOutcome: TYPE = RS232CEnvironment.AutoRecognitionOutcome;
CompletionHandle: TYPE = RS232CEnvironment.CompletionHandle;
OperationClass: TYPE = {input, output, other, all};

PhysicalRecordHandle: TYPE = RS232CEnvironment.PhysicalRecordHandle;
PhysicalRecord: TYPE = RS232CEnvironment.PhysicalRecord;
TransferStatus: TYPE = {success, dataLost-- (caused by input buffer overrun --, deviceError, frameTimeout, checksumError, parityError,
        asynchFramingError-- (i.e. stop bit(s) missing) --, invalidChar, invalidFrame, aborted, disaster};
DeviceStatus: TYPE = RECORD[
    statusAborted: BOOLEAN,
    dataLost: BOOLEAN, --latched: caused by arrival of data with no input buffer allocated
    breakDetected: BOOLEAN, --latched
    clearToSend, dataSetReady, carrierDetect: BOOLEAN, -- correspond to signals in EIA RS-232-C Spec
    ringHeard: BOOLEAN, --latched version of EIA RS-232-C Ring Indicator
    ringIndicator: BOOLEAN];
Parameter: TYPE = RECORD[
    SELECT type: ParameterType FROM
        charLength => [charLength: CharLength],
        correspondent => [correspondent: Correspondent],
        dataTerminalReady => [dataTerminalReady: BOOLEAN],
        frameTimeout => [frameTimeout: CARDINAL],
        latchBitClear => [latchBitClearMask: LatchBitClearMask],
        lineSpeed => [lineSpeed: LineSpeed],
        parity => [parity: Parity],
        requestToSend => [requestToSend: BOOLEAN],
        stopBits => [stopBits: StopBits],
        syncChar => [syncChar: SyncChar],
        syncCount => [syncCount: SyncCount],
        ENDCASE];
ParameterType: TYPE = { charLength, correspondent, dataTerminalReady, frameTimeout, latchBitClear, lineSpeed, parity,
        requestToSend, stopBits, syncChar, syncCount};
CharLength: TYPE = RS232CEnvironment.CharLength;
Correspondent: TYPE = RS232CEnvironment.Correspondent;
LatchBitClearMask: TYPE = DeviceStatus;
LineSpeed: TYPE = RS232CEnvironment.LineSpeed;
```

```
Parity: TYPE = RS232CEnvironment.Parity;
StopBits: TYPE = RS232CEnvironment.StopBits;
SyncCount: TYPE = RS232CEnvironment.SyncCount;
SyncChar: TYPE = RS232CEnvironment.SyncChar;

END. -- RS232C
```

LOG

Time: 1978 By: Victor Schwartz Action: Created file

Time: October 1979 By: Victor Schwartz Action: Pre-Teak LOG entries removed
Time: June 20, 1980 2:51 PM By: Victor Schwartz Action: Pre-Amargosa LOG entries removed.
Time: June 20, 1980 2:51 PM By: Victor Schwartz Action: Remove Abort and ChannelDoesNotExist. Change semantics of Suspend/Restart.

-- LastEdited: August 4, 1980 12:52 PM By: Victor Schwartz
-- This DEFINITIONS contains numerical values of RS232CEnvironment.Correspondent and RS232CEnvironment.AutoRecognitionOutcome. This allows support for additional correspondents and/or additional auto-recognition hardware, without requiring the recompilation of any existing code.

```
DIRECTORY
    RS232CEnvironment USING [AutoRecognitionOutcome,Correspondent];

RS232CCorrespondents: DEFINITIONS =
BEGIN
xerox800: RS232CEnvironment.Correspondent = [0];
xerox850: RS232CEnvironment.Correspondent = [1];
system6: RS232CEnvironment.Correspondent = [2];
cmcII: RS232CEnvironment.Correspondent = [3];
ttyHost: RS232CEnvironment.Correspondent = [4];
oisSystemElement: RS232CEnvironment.Correspondent = [5];
ibm3270Host: RS232CEnvironment.Correspondent = [6];
ibm2770Host: RS232CEnvironment.Correspondent = [7];
ibm6670Host: RS232CEnvironment.Correspondent = [8];
ibm6670: RS232CEnvironment.Correspondent = [9];
xerox860: RS232CEnvironment.Correspondent = [10];
oisSystemElementBSC: RS232CEnvironment.Correspondent = [11];
illegal: RS232CEnvironment.AutoRecognitionOutcome = [0];
failure: RS232CEnvironment.AutoRecognitionOutcome = [1];
asciiByteSync: RS232CEnvironment.AutoRecognitionOutcome = [2];
ebcdicByteSync: RS232CEnvironment.AutoRecognitionOutcome = [3];
bitSync: RS232CEnvironment.AutoRecognitionOutcome = [4];
oisProtocol: RS232CEnvironment.AutoRecognitionOutcome = [5];

END. -- RS232CCorrespondents
```

LOG

Time: January 22, 1980 11:00 AM By: Victor Schwartz Action: Created file
Time: May 30, 1980 2:46 PM By: Victor Schwartz Action: Added ibm3270Host *and* ibm2770Host.
Time: June 26, 1980 2:01 PM By: Victor Schwartz Action: Added correspondents ibm6670Host *and* ibm6670, and autoRecognitionOutcome of oisSystemElement.
Time: August 4, 1980 12:54 PM By: Victor Schwartz Action: Added correspondents xerox860 *and* oisSystemElementBSC.

```
-- LastEdited: August 4, 1980 3:38 PM By: Victor Schwartz
-- This DEFINITIONS module defines basic, (hopefully) unchanging TYPEs required both by the RS232C channel (RS232C) and the
        RS232C device face (RS232CFace). There location here prevents interdependencies.

DIRECTORY
    Environment USING [Byte,Block];

RS232CEnvironment: DEFINITIONS =
BEGIN
AutoRecognitionOutcome: TYPE = RECORD[[0..15]];
CharLength: TYPE = [5..8];
CompletionHandle: TYPE [2];
Correspondent: TYPE = RECORD[[0..255]];
LineSpeed: TYPE = {bps50, bps75, bps110, bps134p5, bps150, bps300, bps600, bps1200, bps2400, bps3600, bps4800, bps7200,
        bps9600};
LineType: TYPE = {bitSynchronous, byteSynchronous, asynchronous, autoRecognition};
Parity: TYPE = {none, odd, even, one, zero};
PhysicalRecordHandle: TYPE = POINTER TO PhysicalRecord;
PhysicalRecord: TYPE = RECORD[
    header, body, trailer: Environment.Block];
StopBits: TYPE = [1..2];
SyncCount: TYPE = [0..7];
SyncChar: TYPE = Environment.Byte;

END. -- RS232CEnvironment
```

LOG

Time: January 22, 1980 10:37 AM By: Victor Schwartz Action: Created file
Time: August 4, 1980 3:36 PM By: Victor Schwartz Action: Change CompletionHandle *to an EXPORTed type, opaque to the client.*

```
-- LastEdited: August 4, 1980  12:57 PM  By: VSS
-- This DEFINITIONS module attempts to capture the lowest level interface to an RS-232-C controller which is still independent of the
        particular controller implementation, as well as the processor on which the controller resides.

DIRECTORY
    Environment USING [Byte], RS232CEnvironment USING
        [AutoRecognitionOutcome,CharLength,CompletionHandle,Correspondent,LineSpeed,LineType,Parity,PhysicalRecord,Ph
        ysicalRecordHandle,StopBits,
        SyncChar,SyncCount];

RS232CFace: DEFINITIONS =
BEGIN

-- Interface Procedures
On: PROCEDURE [lineNumber: CARDINAL];
Off: PROCEDURE [lineNumber: CARDINAL];
GetLineCount: PROCEDURE RETURNS [lineCount: CARDINAL];
ResetLine: PROCEDURE [lineNumber: CARDINAL, paramHandle: ParamHandle] RETURNS [outcome: ParameterOutcome];
SetParameters: PROCEDURE [lineNumber: CARDINAL, paramHandle: ParamHandle] RETURNS [outcome: ParameterOutcome];
AutoRecognitionWait: PROCEDURE [lineNumber: CARDINAL] RETURNS [outcome: RS232CEnvironment.AutoRecognitionOutcome];
Put: PROCEDURE [lineNumber: CARDINAL, rec: RS232CEnvironment.PhysicalRecordHandle] RETURNS
        [RS232CEnvironment.CompletionHandle];
Get: PROCEDURE [lineNumber: CARDINAL, rec: RS232CEnvironment.PhysicalRecordHandle] RETURNS
        [RS232CEnvironment.CompletionHandle];
TransferWait: PROCEDURE [lineNumber: CARDINAL, event: RS232CEnvironment.CompletionHandle] RETURNS [byteCount: CARDINAL,
        status: TransferStatus];
SendBreak: PROCEDURE [lineNumber: CARDINAL];
AbortInput: PROCEDURE [lineNumber: CARDINAL];
AbortOutput: PROCEDURE [lineNumber: CARDINAL];
GetStatus: PROCEDURE [lineNumber: CARDINAL] RETURNS [stat: DeviceStatus];
StatusWait: PROCEDURE [lineNumber: CARDINAL, stat: DeviceStatus] RETURNS [newstat: DeviceStatus];
AbortStatus: PROCEDURE [lineNumber: CARDINAL];
TransmitNow: PROCEDURE [lineNumber: CARDINAL, event: RS232CEnvironment.CompletionHandle] RETURNS [byteCount: CARDINAL,
        status: TransferStatus];

-- TYPEs
ParamHandle: TYPE = LONG POINTER TO ParameterRecord;
ParameterRecord: TYPE = RECORD [
    lineType: RS232CEnvironment.LineType,
    correspondent: RS232CEnvironment.Correspondent,
    lineSpeed: RS232CEnvironment.LineSpeed,
    stopBits: RS232CEnvironment.StopBits,
    parity: RS232CEnvironment.Parity,
    charLength: RS232CEnvironment.CharLength,
    syncCount: RS232CEnvironment.SyncCount,
    syncChar: RS232CEnvironment.SyncChar,
    frameTimeout: CARDINAL,
    requestToSend,dataTerminalReady,resetRingHeard,resetBreakDetected,resetDataLost: BOOLEAN];
ParameterOutcome: TYPE = {success, unimplemented};
TransferStatus: TYPE = {success, dataLost-- (caused by input buffer overrun --, deviceError, frameTimeout, checksumError, parityError,
        asynchFramingError-- (i.e. stop bit(s) missing) --, invalidChar, invalidFrame, aborted, disaster};
DeviceStatus: TYPE = RECORD[
    dataLost: BOOLEAN, --latched: caused by arrival of data with no input buffer allocated
    breakDetected: BOOLEAN, -- latched
    clearToSend,dataSetReady,carrierDetect: BOOLEAN,
    ringHeard: BOOLEAN, -- latched
    ringIndicator: BOOLEAN];

END. -- RS232CFace
```

LOG

Time: May 8, 1979 8:45 AM By: Victor Schwartz Action: Created file
Time: May 30, 1980 2:39 PM By: Victor Schwartz Action: Remove pre-Amargosa log entries, and add lineNumber parameter to On and Off, to allow resource allocation on a line-by-line basis.
Time: August 4, 1980 12:58 PM By: Victor Schwartz Action: Remove PROCEDUREs SuspendInput, SuspendOutput, RestartInput, RestartOutput. Change ParamHandle to a LONG pointer.

-- LastEdited: August 5, 1980 11:26 AM By: VSS

```
DIRECTORY
    Environment USING [Byte], RS232C USING [ChannelHandle], RS232CFace USING [ParameterRecord], RS232CCorrespondents USING
        [oisSystemElement,system6,ttyHost];

RS232CInternal: DEFINITIONS =
BEGIN

-- Interface Definitions

DeleteChannel: PROCEDURE [channel: RS232C.ChannelHandle];

-- Non-interface definitions

ChannelStatusHandle: TYPE = LONG POINTER TO ChannelStatus;

ChannelStatus: TYPE = MONITORED RECORD [
    -- The monitor lock (LOCK) goes here. Make certain it is initialized correctly by Mesa or a runtime call.
    parameterRecord: RS232CFace.ParameterRecord,
    inputSuspended,outputSuspended,otherSuspended: BOOLEAN,
    lineNumber: CARDINAL,
    statusWaitCount: CARDINAL];

bitSynchronousDefaults: RS232CFace.ParameterRecord = [lineType: bitSynchronous, correspondent:
        RS232CCorrespondents.oisSystemElement, lineSpeed: bps1200, stopBits: 1, parity: none, charLength: 8, syncCount: 0,
        syncChar:0, frameTimeout: 0, requestToSend: FALSE, dataTerminalReady: FALSE, resetRingHeard: FALSE,
        resetBreakDetected: FALSE, resetDataLost: FALSE];

byteSynchronousDefaults: RS232CFace.ParameterRecord = [lineType: byteSynchronous, correspondent:
        RS232CCorrespondents.system6, lineSpeed: bps1200, stopBits: 1, parity: none, charLength: 8, syncCount: 2, syncChar:62B--
        EBCDIC SYN--, frameTimeout: 0, requestToSend: FALSE, dataTerminalReady: FALSE, resetRingHeard: FALSE,
        resetBreakDetected: FALSE, resetDataLost: FALSE];

asynchronousDefaults: RS232CFace.ParameterRecord = [lineType: asynchronous, correspondent: RS232CCorrespondents.ttyHost,
        lineSpeed: bps1200, stopBits: 1, parity: none, charLength: 8, syncCount: 0, syncChar:0, frameTimeout: 0, requestToSend:
        FALSE, dataTerminalReady: FALSE, resetRingHeard: FALSE, resetBreakDetected: FALSE, resetDataLost: FALSE];

autoRecognitionDefaults: RS232CFace.ParameterRecord = [lineType: autoRecognition, correspondent: RS232CCorrespondents.system6,
        lineSpeed: bps1200, stopBits: 1, parity: none, charLength: 8, syncCount: 2, syncChar:62B--EBCDIC SYN--, frameTimeout: 0,
        requestToSend: FALSE, dataTerminalReady: FALSE, resetRingHeard: FALSE, resetBreakDetected: FALSE, resetDataLost:
        FALSE]; -- Note that all parameters except requestToSend, dataTerminalReady, resetRingHeard, resetBreakDetected and
        resetDataLost are ignored for lineType = autoRecognition, as they must be overwritten by channel client after the results of auto-recognition
        are reported.
channelIsCreated: READONLY PACKED ARRAY [0..15] OF BOOLEAN;

END. -- RS232CInternal
```

LOG

Time: October 4, 1978 8:28 AM *By: Victor Schwartz* *Action: Created file*
Time: May 23, 1980 12:01 PM By: Victor Schwartz Action: Pre-Amargosa log entries deleted.
Time: May 23, 1980 2:26 PM By: Victor Schwartz Action: Redefined ChannelStatus to allow support of multiple RS232C Channels.
Time: July 15, 1980 7:51 AM By: Victor Schwartz Action: Remove statusChange from ChannelStatus record (RS232C Head does the WAIT).
Time: July 24, 1980 9:48 AM By: Victor Schwartz Action: Change default correspondent for lineType = asynchronous from xerox800 to ttyHost.

```
DIRECTORY
    Dialup USING [RetryCount],
    RS232C USING [ChannelHandle, LineType, LineSpeed, AutoRecognitionOutcome];

RS232CManager: DEFINITIONS =
BEGIN

-- Procedures

    ReserveChannel: PROCEDURE [portID: CARDINAL, useType: ChannelUseType, preemptOthers, preemptMe: ReserveType,
        commParamHandle: CommParamHandle] RETURNS [RS232C.ChannelHandle];

    AwaitAutoRecognition: PROCEDURE [RS232C.ChannelHandle] RETURNS [RS232C.AutoRecognitionOutcome];

    RedefineChannelUse: PROCEDURE [channel: RS232C.ChannelHandle, useType: ChannelUseType, preemptOthers, preemptMe:
        ReserveType];

    ReleaseChannel: PROCEDURE [RS232C.ChannelHandle];


    DescribeCommEquipment: PROCEDURE [portID: CARDINAL, commParamHandle: CommParamHandle];

    GetCommEquipmentDescription: PROCEDURE [portID: CARDINAL] RETURNS [CommParamObject];


    DefineIdleChannelUse: PROCEDURE [channel: RS232C.ChannelHandle, doOisCommInBackground: BOOLEAN] ;

-- Types

    ChannelUseType: TYPE = {oisCommunication, foreignDeviceAccess};

    ReserveType: TYPE = {preemptNever, preemptAlways, preemptInactive};

    CommParamHandle: TYPE = POINTER TO CommParamObject;

    -- The following types help describe the communication equipment (modems) being used.  They are too restrictive for multi-speed and multi-mode
        modems.

    CommParamObject: TYPE = RECORD [
        duplex:  CommDuplex,
        lineType:  RS232C.LineType,
        lineSpeed:  RS232C.LineSpeed,
        accessDetail: SELECT netAccess: NetAccess FROM
            directConn => NULL,
            dialConn  => [
                dialMode: DialMode,
                retryCount:  Dialup.RetryCount],
            ENDCASE
        ];

    CommDuplex: TYPE = {full, half};
    NetAccess: TYPE = {directConn, dialConn};
    DialMode: TYPE = {manual, auto};
    ReserveFailedReason: TYPE = {noRS232CHardware, unimplementedFeature, inUse, inconsistentParams};

-- Signals and Errors

    ReserveFailed: ERROR [reason: ReserveFailedReason];
    NonrecoverableSoftwareFailure: ERROR;
END.
```

LOG

Time: August 7, 1978 11:52 AM By: Garlick Action: Created file
Time: June 26, 1980 2:11 PM By: Schwartz Action: Removed pre-Amargosa log entries.
Time: June 26, 1980 2:12 PM By: Schwartz Action: Added ChannelHandle parameter to DefineIdleChannelUse, and changed ReserveFailedReason "noAutoRecognition" to "unimplementedFeature".
Time: August 1, 1980 5:33 PM By: BLyon Action: changed netNumber in from CARDINAL to SpecialSystem.NetworkNumber.
Time: August 4, 1980 4:37 PM By: Schwartz Action: Removed references to netNumber.

```
-- LastEdited: July 10, 1980  10:16 AM  By: BRD
-- This DEFINITIONS module attempts to capture the lowest level interface to an RS-366 port which is still independent of the particular
        implementation, as well as the processor on which the port resides.

RS366Face: DEFINITIONS =
BEGIN

-- Interface Definitions
GetDialerCount: PROCEDURE RETURNS [dialerCount: CARDINAL];
SetStatus: PROCEDURE [dialerNumber: CARDINAL, setStatusBits: SetStatusBits];
GetStatus: PROCEDURE [dialerNumber: CARDINAL] RETURNS [getStatusBits: GetStatusBits];

-- Non-interface definitions
SetStatusBits: TYPE = RECORD [ -- Settable RS366 bits --
    callRequest, digitPresent: BOOLEAN,
    digit: [0..15]];
GetStatusBits: TYPE = RECORD [
    powerIndication, dataLineOccupied, callOriginationStatus, presentNextDigit, abandonCallAndRetry: BOOLEAN];

END. -- RS366Face
```

LOG

Time: September 21, 1979 9:01 AM By: Victor Schwartz Action: Created file
Time: January 15, 1980 5:29 PM *By: Victor Schwartz Action: added GetDialerCount*
Time: January 15, 1980 5:29 PM *By: Bill Danielson Action: separated status bits into two records; SetStatusBits for bits the client can set, and GetStatusBits for bits the client can read.*

```
-- Runtime.mesa (last edited by: Johnsson on: September 4, 1980  4:37 PM)

DIRECTORY
  File USING [Capability, PageCount],
  Mopcodes USING [zKFCB],
  SDDefs USING [sCallDebugger, sInterrupt, sUnNew],
  System USING [GreenwichMeanTime];

Runtime: DEFINITIONS =
  BEGIN -- Global Frame management
  GlobalFrame: PROC [link: UNSPECIFIED] RETURNS [PROGRAM];
  LoadConfig: PROC [
    file: File.Capability, offset: File.PageCount, codeLinks: BOOLEAN ← FALSE]
    RETURNS [PROGRAM];
  NewConfig: PROC [
    file: File.Capability, offset: File.PageCount, codeLinks: BOOLEAN ← FALSE];
  RunConfig: PROC [
    file: File.Capability, offset: File.PageCount, codeLinks: BOOLEAN ← FALSE];
  UnNewConfig: PROC [link: UNSPECIFIED];
  UnNew: PROC [frame: PROGRAM] = MACHINE CODE
    BEGIN Mopcodes.zKFCB, SDDefs.sUnNew END;

  SelfDestruct: PROC;
  NullProgram: PROGRAM = LOOPHOLE[0]; -- Frame validation
  ValidateGlobalFrame: PROC [frame: UNSPECIFIED];
  InvalidGlobalFrame: ERROR [frame: UNSPECIFIED];
  ValidateFrame: PROC [frame: UNSPECIFIED];
  InvalidFrame: ERROR [frame: UNSPECIFIED]; -- Debugger interface
  CallDebugger: PROC [STRING] = MACHINE CODE
    BEGIN Mopcodes.zKFCB, SDDefs.sCallDebugger END;

  Interrupt: PROC = MACHINE CODE BEGIN Mopcodes.zKFCB, SDDefs.sInterrupt END;
  -- Version information

  GetBcdTime: PROC RETURNS [System.GreenwichMeanTime];
  GetBuildTime: PROC RETURNS [System.GreenwichMeanTime]; -- Debugging information
  GetCaller: PROC RETURNS [PROGRAM]; -- Access to compiled tables, etc
  GetTableBase: PROC [frame: PROGRAM] RETURNS [LONG POINTER];
  -- Conditional loading information
  IsBound: PROC [link: UNSPECIFIED] RETURNS [BOOLEAN];
  -- Signals and errors generated by the Pilot Mesa System and of interest to Mesa programmers
  BoundsFault: SIGNAL;
  ConfigError: ERROR [type: ConfigErrorType];
  ConfigErrorType: TYPE = {
    badCode, invalidConfig, missingCode, versionMismatch, unknown};
  ControlFault: SIGNAL [source: UNSPECIFIED --ControlLink--]
    RETURNS [UNSPECIFIED --ControlLink--];
  DivideCheck: SIGNAL;
  LinkageFault: ERROR;
  PointerFault: SIGNAL;
  PortFault: ERROR;
  StartFault: SIGNAL [dest: PROGRAM];
  StackError: ERROR;
  UnboundProcedure: SIGNAL [dest: UNSPECIFIED --ControlLink--]
    RETURNS [UNSPECIFIED --ControlLink--];
  ZeroDivisor: SIGNAL;
  END.
LOG
(For earlier log entries see Pilot 4.0 archive version.)
Time: April 17, 1980  10:20 AM  By: Luniewski   Action: Added NullProgram.
Time: July 18, 1980  6:22 PM    By: Forrest     Action: Added LoadConfig.
Time: August 5, 1980  10:41 AM  By: McJones     Action: Added GetBcdTime, Interrupt.
Time: September 4, 1980  4:37 PM        By: Johnsson    Action: Added codeLinks parameters for loading.
**
```

```
DIRECTORY
    Environment: FROM "Environment" USING [PageNumber],
    Mopcodes: FROM "Mopcodes" USING [zKFCB],
    PrincOps: FROM "PrincOps" USING [GFTIndex, GlobalFrameHandle],
    SDDefs: FROM "SDDefs" USING [sWorryCallDebugger];

RuntimeInternal: DEFINITIONS =
BEGIN

    -- Frame management
GetNextGlobalFrame: PROCEDURE [frame: PrincOps.GlobalFrameHandle]
        RETURNS [nextFrame: PrincOps.GlobalFrameHandle];
EnterGlobalFrame: PROCEDURE [frame: PrincOps.GlobalFrameHandle, nslots: CARDINAL]
        RETURNS [PrincOps.GFTIndex];
RemoveGlobalFrame: PROCEDURE [frame: PrincOps.GlobalFrameHandle];
DeletedFrame: PROCEDURE [gfi: PrincOps.GFTIndex] RETURNS [BOOLEAN];
Codebase: PROCEDURE [frame: PROGRAM] RETURNS [LONG POINTER];
MakeFsi: PROCEDURE [words: CARDINAL] RETURNS [fsi: CARDINAL];
FrameSize: PROCEDURE [fsi: CARDINAL] RETURNS [CARDINAL];
FlushLargeFrames: PROCEDURE;
NoGlobalFrameSlots: SIGNAL [CARDINAL];

    -- Xfer Tracing
StartTrace: PROCEDURE [loc: POINTER, val: UNSPECIFIED, mask: WORD, equal: BOOLEAN];
StopTrace: PROCEDURE;

    -- Calling the debugger to process the map log
CleanMapLog: PROCEDURE;

    -- Calling the debugger from resident modules
WorryCallDebugger: PROCEDURE [STRING] =
        MACHINE CODE BEGIN Mopcodes.zKFCB, SDDefs.sWorryCallDebugger END;

    -- Signals and errors generated by the Signals module
SendMsgSignal: SIGNAL RETURNS [UNSPECIFIED, UNSPECIFIED];
ResumeError: SIGNAL;

    -- Signals and errors generated by the PilotNub module
CantSwap: SIGNAL;
CAbort: SIGNAL;

    -- "pointer" to the load information
loadStatePage: Environment.PageNumber;

END.
```

LOG
Time: April 24, 1980 5:02 PM By: Forrest Action: Trimmed log to Amargosa; ControlDefs = > PrincOps

```
DIRECTORY
    Environment USING [PageCount, PageNumber],
    PrincOps USING [ControlModule, GlobalFrameHandle];

RuntimePrograms: DEFINITIONS =

BEGIN

InitializeFrames: PROCEDURE;

InitializeInstructions: PROCEDURE;

InitializeInterrupt: PROCEDURE;

InitializePilotLoadState: PROCEDURE [
        pageInitialLoadState: Environment.PageNumber, countInitialLoadState: Environment.PageCount];

InitializePilotNub: PROCEDURE [
        pageLoadState: Environment.PageNumber,
        countLoadState: Environment.PageCount,
        pVMMapLog: LONG POINTER --TO VMMMapLog.Descriptor--];
    -- Arguments are page and count of the initial loadState.

InitializeProcesses: PROCEDURE [pagePDA: Environment.PageNumber, countPDA: Environment.PageCount];

InitializeSignals: PROCEDURE;

InitializeSnapshot: PROCEDURE;

InitializeTraps: PROCEDURE;

-- Exported by Traps:

Start: PROCEDURE [PrincOps.ControlModule];

Restart: PROCEDURE [PrincOps.GlobalFrameHandle];

-- Delete the following once everyone is converted to InitializeMumble:

Instructions: PROGRAM;

Processes: PROGRAM [pagePDA: Environment.PageNumber, countPDA: Environment.PageCount];

Signals: PROGRAM;

Traps: PROGRAM;

END.
```

Time: January 27, 1980 8:58 PM	*By: Forrest*	*Action: Moved LoadStatePage to RuntimeInternal*
Time: February 22, 1980 4:45 PM	*By: McJones*	*Action: Delete debugger, debuggee parameters from PilotNub*
Time: April 14, 1980 10:41 AM	*By: Knutsen*	*Action: Frames, SnapshotImpl, PilotLoadState now STARTed by Initialize*.*
Time: April 25, 1980 8:33 AM	*By: Forrest*	*Action: FrameOps = > PrincOps*
Time: May 17, 1980 6:46 PM	*By: Forrest*	*Action: Start Control Modules.*
Time: July 2, 1980 5:32 PM	*By: Knutsen*	*Action: Added parameters to InitializePilotLoadState.*
Time: August 25, 1980 3:16 PM	*By: McJones*	*Action: Add Initialize{Instructions, Interrupt, PilotNub, Processes, Signals, Traps}*

```
DIRECTORY
    Environment USING [Base],
    PilotDisk USING [Handle, Label];

SA4000Face: DEFINITIONS =

BEGIN

-- Processor-independent access to a disk drive of the Shugart SA1000/SA4000 family.


-- PROCEDURES

GetDeviceAttributes: PROCEDURE [DeviceHandle] RETURNS [cylinders, movingHeads, fixedHeads, sectorsPerTrack: CARDINAL];
    -- Return the "shape" of a particular drive in the family.

GetNextDevice: PROCEDURE [DeviceHandle] RETURNS [DeviceHandle];
    -- Return handle of next drive in physical order (starts and ends with nullDeviceHandle).

Initialize: PROCEDURE [t: WORD, globalState: GlobalStatePtr];
    -- Initialize. t is a mask for transmission wakeups. globalState is a client supplied block of globalStateSize words, 16-word aligned
        in first 64K of address space. Lifetime: permanently allocated and passed in on call to Initialize.

InitializeCleanup: PROCEDURE;
    -- Install cleanup coroutine used to quiesce SA4000 devices before a world swap (see DeviceCleanup.mesa).

Initiate: PROCEDURE [OperationPtr];
    -- Queue an operation using the OperationState supplied by the user.

Poll: PROCEDURE [OperationPtr] RETURNS [Status];
    -- Return the status of an operation previously queued.

Recalibrate: PROCEDURE [DeviceHandle];
    -- Cause the disk to be recalibrated before another command is executed.

Reset: PROCEDURE [DeviceHandle];
    -- To be defined.


-- TYPES

DiskAddress: TYPE = MACHINE DEPENDENT RECORD [
    cylinder (0): CARDINAL,
    head (1:0..7): [0..256),
    sector (1:8..15): [0..256)];
    -- The assignment of DiskAddress's to sectors on disks numbers moving heads before fixed heads (if any).

DeviceHandle: TYPE = PilotDisk.Handle;

GlobalStatePtr: TYPE = Environment.Base RELATIVE POINTER;

Command: TYPE = {
        -- Command to be executed by Initiate. If the header operation is read, the label and data field operations will be executed
        even if the header or label does not verify.
        -- In the following commands, v is for verify, r is for read, and w is for write. Field actions are always specified in the order header, label,
        data. For commands with less than three actions, the trailing fields are passed.
        vv, vvr, vvw, vvv, vw, vww, vr, vrr, vrw, vrv, rv, rvr, rvw, rvv, rw, rww, rr, rrr, rrw, rrv, w};

Operation: TYPE = MACHINE DEPENDENT RECORD [
    -- Operation: client supplied block of operationSize words, 16-word aligned in first 64K of address space. The actual size depends
        on the implementation because the trailing portion is used by the microcode for workspace. The portion shown below is used for
        communication between the client and the Face and for communication between between the Face and the microcode.
    -- Lifetime: duration of I/O operation - i.e. from original call on Initiate until either the operation has completed (a call on Poll has
        returned a status for it other than inProgress) or some previous operation has incurred an error (a call on Poll has returned a
        status for it other than inProgress or goodCompletion).
    -- A call on Poll causes fields in the Operation to be updated. The following fields have special significance for runs of pages:
        pageCount, clientHeader, label, and dataPtr. These fields are updated by the Face implementation after each page successfully
        transferred. dataPtr is an exception: if incrementDataPtr is not TRUE, dataPtr will not be updated after each page transferred.
        This option may be used for example to write successive pages of zeros or read don't care data while checking labels.
        -- If the Status returned by Poll is inProgress, care must be taken when accessing these fields because they may be continually
        updated by the Face implementation. For example, if Poll returns inProgess, the pageCount may be zero by the time it is
        accessed by the client.
    clientHeader (0): DiskAddress,  -- disk address to be accessed, initially supplied by client. After Poll, this contains the next sequential disk address
                                    if the Status is inProgress or goodCompletion; otherwise it contains the address that caused the error.
    labelPtr (2): LONG POINTER TO PilotDisk.Label,  -- label initially supplied by client for label verify or read from the disk after a label read operation.
                                    This field is treated by the Face implementation as a composite field with two parts: the first eight words and the
                                    last two words.
                                    -- Whenever the command involves label verify, PilotDisk.MatchLabels is used to compare clientLabel with the
```

*label on the disk. After each successfully transferred page, PilotDisk.NextLabel is used to increment the "filePageHi*2^32 + filePageLo" field and to clear the immutable, temporary, and zeroSize fields (since they are set only the first page of a file). Thus if* ***Poll*** *returns goodCompletion, label will reflect pageCount applications of PilotDisk.NextLabel.*
*-- Once an operation has completed (****Poll*** *returns status other than inProgress), label.bootChainLink will have set by the face implementation to the corresponding field of the disk label of the last successfully transferred page.*

dataPtr (4): LONG POINTER, *-- points to the first 256 word data area in the run, initially supplied by client. After each page successfully transferred, this is updated to be the next data address if incrementDataPtr is TRUE.*
incrementDataPtr (6:0..0): BOOLEAN, *-- if TRUE, dataPtr will be incremented to the next data address after every page successfully transferred; if FALSE, the same dataPtr will be used for each command.*
unused (6:1..10): [0..1777B] ← 0,
command (6:11..15): Command, *-- command to be executed.*
pageCount (7): CARDINAL, *-- number of pages to access, supplied by client. This must always be at least one. After* ***Poll****, this is updated to be the number of pages left to access.*
deviceStatus (10B): RECORD [a, b: UNSPECIFIED] ← NULL, *-- device dependent status set by call on* ***Poll***
diskHeader (12B): DiskAddress ← NULL, *-- buffer to read header from the disk. The contents are only valid after a headerRead operation has completed.*
device (14B): DeviceHandle]; *-- drive to access*

OperationPtr: TYPE = LONG POINTER TO Operation;

Status: TYPE = MACHINE DEPENDENT {inProgress (0), goodCompletion, dataError, labelError, labelCheck, seekTimeout, sectorTimeout, notReady, wrongSector, wrongCylinder, wrongHead, hardwareError};

-- EXPORTED VARIABLES

globalStateSize: READONLY CARDINAL;
nullDeviceHandle: READONLY DeviceHandle;
operationSize: READONLY CARDINAL;

-- CONSTANTS

dataSize: CARDINAL = 256; *-- size of data field in words*
headerSize: CARDINAL = 2; *-- size of header field in words*
labelSize: CARDINAL = 10; *-- size of label field in words*

END.

LOG

Time: August 8, 1979 5:19 PM By: Redell Action: Add log to file from Jarvis
Time: August 22, 1979 12:40 PM By: Gobbel Action: General cleanup
Time: August 30, 1979 1:24 PM By: Gobbel Action: Remove references to RDC
Time: September 26, 1979 3:55 PM By: Redell Action: General cleanup, esp for SA4000 germ
Time: October 12, 1979 11:05 AM By: McJones Action: Set fixedHeads to 0
Time: November 16, 1979 10:23 PM By: Frandeen Action: Change interface to Initiate and Poll for runs of pages and diagnostics; delete sector mask (now unused) from StartB
Time: December 13, 1979 11:40 AM By: Gobbel Action: Add default for unused field of Operation
Time: January 18, 1980 4:18 PM By: McJones Action: Replace OISProcessorFace.DiskHandle with OISDisk.Handle (and add GetNextDevice operation); replace FilePageLabel.Label with OISDisk.Label; replace constants with GetDeviceAttributes; delete hardwareState and device parameter to InitializeCleanup
Time: June 2, 1980 6:15 PM By: McJones Action: 48-bit processor ids

```
DIRECTORY PilotDisk USING [Handle];

SA800Face: DEFINITIONS =
 BEGIN

 -- Processor independent access to a floppy disk drive of the SA800 family. To date this includes the SA800/801 and the SA850/851.
Each controller may have one or more drives, each drive may have one or more recording surfaces (read/write heads and recording
surfaces are synonymous). Currently the SA800/801 is known to be single sided while the SA850/851 may be single or double sided.

 -- EXPORTED VARIABLES

 initialAllocationLength: READONLY CARDINAL; --Length in words of a block of resident memory allocated for internal use by the
device face implementation. Alignment: Quadword. Lifetime: Permanent.

 operationBlockLength: READONLY CARDINAL; --Length in words of the Operation block that must be allocated from resident memory
to create any of the asynchronous functions defined by Initiate and Format. Alignment: Quadword. Lifetime: Duration of the operation.

 nullDeviceHandle: READONLY DeviceHandle; --Null device handle used in device enumeration.


 -- TYPES AND PROCS

 DeviceHandle: TYPE = PilotDisk.Handle;

 Context: TYPE = RECORD [
  protect: BOOLEAN,          --Software write protect available to the client. The actual write protect status is a logical OR of this
                               variable and the physical signal being returned from the drive.
  format: {IBM, Troy},       --Client's selection of the format type; either an IBM compatible format (which includes STAR), or the
                               Xerox 850 (Troy) format.
  density: {single, double}, --Selection of either FM (single density) or MFM (double density) recording. This is an available
                               selection when formatting a new disketted. When accessng a previously recorded diskette, the
                               client must provide the correct setting. (Note that track00 on IBM format diskettes and all tracks of
                               Troy format diskettes will be single density.)
  sectorLength: CARDINAL];   --Length in words of the sectors on the current track. The value must come from a valid set defined
                               as {64, 128, 256, 512} for IBM format and {1022} for Troy format.

 SetContext: PROC [device: DeviceHandle, context: Context] RETURNS [ok: BOOLEAN];
  --Sets context as defined above and returns a BOOLEAN indicating whether the setting where accepted.

 GetContext: PROC [device: DeviceHandle] RETURNS [context: Context];
  --Returns the current settings of the context.

 Attributes: TYPE = RECORD [
  deviceType: {SA800, SA850},  --Indicates what type of drive is connected to the controller.
  numberOfCylinders: [0..256), --Number of cylinders available for recording on the drive connected to the controller.
  numberOfHeads: [0..256),     --Number of read/write heads available for recording on the drive connected to the controller.
  trackLength: CARDINAL];      --Length in words of the buffer that the client must supply in order to format the diskette.

 GetDeviceAttributes: PROC [device: DeviceHandle] RETURNS [attributes: Attributes];
  --Returns attributes as defined above.

 -- An operation is required for all IO.
 -- If function IN [nop..recalibrate], only the device and function fields are of use. All other fields are considered to be UNDEFINED and
may be modified by implementations as they see fit.
 -- if function IN [readSector..readID]) the count is taken to be the number of sectors in the IO operation. if function = formatTrack,
diskAddress.sector is ignored and count is taken to be a track count.

 OperationPtr: TYPE = LONG POINTER TO Operation;
 Operation: TYPE = MACHINE DEPENDENT RECORD [
  device (0): DeviceHandle,
  function (1: 0..2): Function,                  -- ignored if submitted by FormatTrack
  unused (1: 3..14): UNSPECIFIED [0..4096) ← 0,  -- always ignored
  incrementDataPointer (1: 15..15): BOOLEAN,     -- ignored if nop, recal, recovery
  address (2): DiskAddress,                      -- ignored if nop, recal, recovery. Sector is ignored in FormatTrack operations.
  buffer (4): Buffer,                            -- ignored if nop, recal, recovery
  count (7): CARDINAL];                          -- ignored if nop, recal, recovery. Number of tracks to format if FormatTrack; number
                                                    of sectors to transfer otherwise.


 Function: TYPE = {
  nop,                --Does not transfer any data but does create an asychronous operation and returns a valid endng status.
```

```
  recalibrate,          --Request the read/write heads step out until track00 is sensed.
  recovery,             --Performs extended error recovery procedure
  readSector,           --Reads one sector from the diskette at the specified disk address.
  writeSector,          --Writes one sector with a data addres mark at the specified disk address.
  writeDeletedSector,   --Writes one sector with a deleted data addres mark at the specified disk address.
  readID,               --Reads to first encountered record ID from the specified disk address (the value of sector in the disk address is
                          ignored).
  formatTrack};         --format the track specified by DiskAddress

Buffer: TYPE = RECORD [
  address: LONG POINTER, --quadword aligned.
  length: CARDINAL];     --Length in words.

DiskAddress: TYPE = MACHINE DEPENDENT RECORD [
  cylinder: CARDINAL,    --must be IN [0..numberOfCylinders]
  head: [0..256),        --must be IN [0..numberOfHeads]
  sector: [0..256)];     --must be IN [1..{value depends on sectorLength}]

Status: TYPE = MACHINE DEPENDENT RECORD [
  diskChange: BOOLEAN,          -- Disk drive has apparently gone to a not ready (door open) state since the last operation was
                                   performed. Only DiskChangeClear will reset this status.
  na1: BOOLEAN,
  twoSided: BOOLEAN,            --The diskette currently installed is two-sided.
  na3: BOOLEAN,
  error: BOOLEAN,               --This status record indicates an error
  inProgress: BOOLEAN,          --The operation is not yet complete
  recalibrateError: BOOLEAN,    --recalibrate subfunction failed to sense track00.
  dataLost: BOOLEAN,            --The buffer supplied by the client was not as large as the data transfer requested.
  notReady: BOOLEAN,            --The drive is not ready.
  writeProtect: BOOLEAN,        --Logical OR of the context setting of protect and the physical signal being returned from the drive.
  deletedData: BOOLEAN,         --The sector contained a deleted data address mark.
  recordNotFound: BOOLEAN,      --The record defined by the disk address could not be found.
  crcError: BOOLEAN,            --A CRC error was encountered on either the record ID or data field.
  track00: BOOLEAN,             --Read/Write heads are currently positioned over track00, the outermost track.
  index: BOOLEAN,               --An index pulse signal was detected concurrent with the posting of this status.
  busy: BOOLEAN];               --The drive is not idle.

Initiate: PUBLIC PROC [operationPtr: OperationPtr]
  RETURNS [status: Status, maxSecondsThisOperation: CARDINAL];
--Used to create asynchronous operations from the Function set. The status reflects any errors preventing the operatin from being
preformed. MaxSecondsPerOperation can be used by drivers for timeouts. It reflects the seconds the operation will take AFTER
pervious operations have been completed.

Poll: PUBLIC PROC [operationPtr: OperationPtr] RETURNS [status: Status];
--Checks status of an asynchronous operation.

DiskChangeClear: PUBLIC PROC [device: DeviceHandle];
--Clears the disk change status.

Reset: PUBLIC PROC [device: DeviceHandle];
--Stops all processing being done by the Head, causing it to "forget" all known operations, and sets the hardware to a startup state.

Initialize: PUBLIC PROC [notify: WORD, initialAllocation: LONG POINTER];
--Provides information to the device face implementation that will be used internally. (Must be called before any other procedures.)

GetNextDevice: PUBLIC PROC [previous: DeviceHandle]
  RETURNS [next: SA800Face.DeviceHandle];
--Enumerates all devices and controllers for the SA800Face.

InitializeCleanup: PUBLIC PROC [device: DeviceHandle];
--Initializes procedures used during device startup and shutdown.

END...
LOG
Edited: June 17, 1980 9:53 AM by AOF: Added log to file.
Edited: June 18, 1980 9:11 AM by AOF: Added additional enumerated element (recovery) to functions allowed in Initiate.
Edited: June 24, 1980 10:32 AM by AOF: Made recalibrate a subfunction of Initiate.
Edited: June 27, 1980 10:34 AM by AOF: Changed defintion of Reset.
Edited: July 29, 1980 9:41 AM by AOF: Changed ref to OISDisk to PilotDisk.
Edited: August 23, 1980 10:33 AM by AOF: Add multiple sector per operation.
Edited: September 15, 1980 3:16 PM by Forrest: Add don't increment field; delete SA800HeadD0: PROGRAM.
```

```
-- SA800NeckD0.mesa (last edited by: Forrest on: September 18, 1980  3:24 PM)--

DIRECTORY
   Mopcodes USING [zMISC],
   SA800Face USING [Context, Operation, OperationPtr, Status];

SA800NeckD0: DEFINITIONS =
BEGIN

-- CONSTANTS

-- The following are canned status that may be returned to the client, usually at Initate time.
clearResult: Result = LOOPHOLE [-1];
notReady: SA800Face.Status =  [
    diskChange: FALSE, na1: FALSE, twoSided: FALSE, na3: FALSE, error: TRUE,
    inProgress: FALSE, recalibrateError: FALSE, dataLost: FALSE, notReady: TRUE,
    writeProtect: FALSE, deletedData: FALSE, recordNotFound: FALSE, crcError: FALSE,
    track00: FALSE, index: FALSE, busy: FALSE];
diskChange: SA800Face.Status =  [
    diskChange: TRUE, na1: FALSE, twoSided: FALSE, na3: FALSE, error: TRUE,
    inProgress: FALSE, recalibrateError: FALSE, dataLost: FALSE, notReady: FALSE,
    writeProtect: FALSE, deletedData: FALSE, recordNotFound: FALSE, crcError: FALSE,
    track00: FALSE, index: FALSE, busy: FALSE];
inProgress: SA800Face.Status = [
    diskChange: FALSE, na1: FALSE, twoSided: FALSE, na3: FALSE, error: FALSE,
    inProgress: TRUE, recalibrateError: FALSE, dataLost: FALSE, notReady: FALSE,
    writeProtect: FALSE, deletedData: FALSE, recordNotFound: FALSE, crcError: FALSE,
    track00: FALSE, index: FALSE, busy: TRUE];
writeProtect: SA800Face.Status = [
    diskChange: FALSE, na1: FALSE, twoSided: FALSE, na3: FALSE, error: TRUE,
    inProgress: FALSE, recalibrateError: FALSE, dataLost: FALSE, notReady: FALSE,
    writeProtect: TRUE, deletedData: FALSE, recordNotFound: FALSE, crcError: FALSE,
    track00: FALSE, index: FALSE, busy: FALSE];
infoMask: SA800Face.Status = [
    diskChange: FALSE, na1: FALSE, twoSided: TRUE, na3: FALSE, error: FALSE,
    inProgress: TRUE, recalibrateError: FALSE, dataLost: FALSE, notReady: FALSE,
    writeProtect: TRUE, deletedData: FALSE, recordNotFound: FALSE, crcError: FALSE,
    track00: TRUE, index: FALSE, busy: FALSE];
statusError: SA800Face.Status = [
    diskChange: FALSE, na1: FALSE, twoSided: FALSE, na3: FALSE, error: TRUE,
    inProgress: FALSE, recalibrateError: TRUE, dataLost: TRUE, notReady: TRUE,
    writeProtect: FALSE, deletedData: FALSE, recordNotFound: TRUE, crcError: TRUE,
    track00: FALSE, index: FALSE, busy: FALSE];

-- major function words for supported commands
ChipLoad: TYPE = MACHINE DEPENDENT RECORD [
   addr: ChipAddr,        --FDC addressing lines
   displ: [0..17B],       --displacement to next field of IOCB
   data: [0..377B]];      --command data byte
ChipAddr: TYPE = MACHINE DEPENDENT RECORD [
   notRd: BOOLEAN,       --FDCwrite|FDCread'
   notCs: BOOLEAN,       --FDCdack|FDCcs'
   a1: BOOLEAN,          --A1
   a0: BOOLEAN];         --A0
WakeLoad: TYPE = MACHINE DEPENDENT RECORD [
   cmd: [0..17B],        --command dispatch type
   displ: [0..17B],      --displacement to next IOCB field
   wake: [0..377B]];     --WakeAllow funtion

command:   ChipAddr = [notRd: TRUE, notCs: FALSE, a1: FALSE, a0: FALSE];
parameter: ChipAddr = [notRd: TRUE, notCs: FALSE, a1: FALSE, a0: TRUE];

parm0: CARDINAL = 12B;  --offset to parameter #0
parm1: CARDINAL = 13B;  --offset to parameter #1
parm2: CARDINAL = 14B;  --offset to parameter #2
parm3: CARDINAL = 15B;  --offset to parameter #3
parm4: CARDINAL = 16B;  --offset to parameter #4

sel0: UNSPECIFIED = 100B;  --select drive 0--

nop:              ChipLoad = [addr: command, displ: 00B, data: 0B];
readData:         ChipLoad = [addr: command, displ: parm0, data: sel0+27B];
writeData:        ChipLoad = [addr: command, displ: parm0, data: sel0+13B];
writeDeletedData: ChipLoad = [addr: command, displ: parm0, data: sel0+17B];
readID:           ChipLoad = [addr: command, displ: parm0, data: sel0+33B];
specify:          ChipLoad = [addr: command, displ: parm0, data: 65B];
seekTrack:        ChipLoad = [addr: command, displ: parm0, data: sel0+51B];
formatTrack:      ChipLoad = [addr: command, displ: parm0, data: sel0+43B];
```

```
-- Specification of the wake allow after the command phase
nopWakeAllow:   WakeLoad = [cmd: 0B, displ: 0B, wake: 5B];
seekWakeAllow:  WakeLoad = [cmd: 6B, displ: 11B, wake: 1B];
readWakeAllow:  WakeLoad = [cmd: 10B, displ: 11B, wake: 11B];
writeWakeAllow: WakeLoad = [cmd: 14B, displ: 11B, wake: 11B];
initWakeAllow:  WakeLoad = [cmd: 0B, displ: 11B, wake: 5B];

CommandMatrix: TYPE = RECORD [
   function: ChipLoad, wake: WakeLoad,
   stuffer: PROCEDURE [iocb: IOCBlongPtr, operation: SA800Face.OperationPtr]];

SectorIndex: TYPE = [0..4);
SectorMatrix: TYPE = RECORD [
   sectorsPerTrack: CARDINAL,  --number of sectors per track given sector length
   gap3: CARDINAL];            -- length of gap3 (formatting) given sector length

StateHandle: TYPE = LONG POINTER TO State;
State: TYPE = RECORD [
   --Alignment: none.
   --Location: global frame.
   --Lifetime: life of device face.
   user: SA800Face.Context,    --portion of context available to client
   head: HeadContext];         --not available to client

HeadContext: TYPE = RECORD [
   status: SA800Face.Status,   --most currently processed status
   sectorIndex: SectorIndex];  --index into sector length table

-- definition of the CSB for the floppy disk
CSBPtr: TYPE = LONG POINTER TO CSB;
CSB: TYPE = MACHINE DEPENDENT RECORD [state: CSBState];
CSBState: TYPE = MACHINE DEPENDENT RECORD [
   --Alignment: hexword.
   --Location: assumed to be last 16K of first64K.
   --Lifetime: forever.
   next(0): IOCBlongPtr,       -- assumption:  MSW is always zero (first64K)
   abortMicrocode(2): UNSPECIFIED,
   notify(3): UNSPECIFIED,     -- interrupt mask word used by firmware
   tail(4): IOCBlongPtr];      -- (software) pointer to end of IOCB chain

-- definition of the IOCB for the floppy disk
-- The queue of IOCB's the hardware knows about can be broken if there are operations
-- with runs of pages
IOCBlongPtr: TYPE = LONG POINTER TO IOCB;
IOCB: TYPE = MACHINE DEPENDENT RECORD [
   --Alignment: quadword.
   --Location: first64K of memory.
   --Lifetime: Duration of the operation that it represents.
   next(0B): IOCBlongPtr,          --assumption:  MSW is always zero (first64K)
   softwareNext(2B): IOCBlongPtr,  --actual unbroken queue of IOCB's
   address(4B): LONG POINTER,      --pointer to client (quadword aligned) buffer
   length(6B): CARDINAL,           --length of client buffer in words
   result(7B): Result,             --result status collected from controller/device
   wake(10B): WakeLoad,            --word to be loaded in cmdWakeAllow register
   function(11B): ChipLoad,        --word to be loaded in cmd register
   p0(12B): ChipLoad,              --parameter number 0
   p1(13B): ChipLoad,              --parameter number 1
   p2(14B): ChipLoad,              --parameter number 2
   p3(15B): ChipLoad,              --parameter number 3
   p4(16B): ChipLoad,              --parameter number 4
   unused(17B): UNSPECIFIED];      --not currenly assigned

Result: TYPE = MACHINE DEPENDENT RECORD [
   standardResult: StandardResult,       --Standard result from chip [0:5]
   driveStatus: DriveStatus,             --Drive status read from chip [5:7]
   controllerStatus: ControllerStatus];  --Controller status at function end [12:4]

StandardResult: TYPE = MACHINE DEPENDENT RECORD [
   deleted: BOOLEAN,                     --Deleted data found if TRUE
   completion: Completion];              --4 bits of completion type and code

DriveStatus: TYPE = MACHINE DEPENDENT RECORD [
   ready1: BOOLEAN,                      --Drive1 is ready
   writeFault: BOOLEAN,                  --?
   indexPulse: BOOLEAN,                  --coincident with operation complete
   writeProtect: BOOLEAN,                --physically protected
   ready0: BOOLEAN,                      --Drive0 is ready
   track00: BOOLEAN,                     --head currently position over track00
```

```
   customerOption: BOOLEAN];              --?

ControllerStatus: TYPE = MACHINE DEPENDENT RECORD [
   dmaWakeReq: BOOLEAN,                    --DMA wakeup required
   oDataParity: BOOLEAN,                   --parity error on output data
   oFault: BOOLEAN,                        --output fault
   fdcIntF: BOOLEAN];                     --FDC interrupt

Completion: TYPE = MACHINE DEPENDENT RECORD [
   type: [0..3B],                         --completion type (only zero is good)
   code: [0..3B]];                        --completion code

-- PROCEDURES

OUTPUT: PUBLIC PROCEDURE [value, address: UNSPECIFIED] =
   MACHINE CODE BEGIN Mopcodes.zMISC, 6; END;

END.....-- SA800NeckD0
```

LOG

June 25, 1980 9:22 AM by AOF; Added log to file. Changes to make recalibrate a subfuncion of Initiate.
June 27, 1980 4:13 PM by AOF; Added check for controller status and method of setting error in status when one or more fatal errors has occurred..
September 15, 1980 4:22 PM by Forrest; Add pending field to IOCB; add explict bit positions to some Machine Dependent RECORDS..

```
DIRECTORY
    File USING [Capability, ID, PageCount, PageNumber, Type],
    Space USING [PageNumber],
    System USING [GreenwichMeanTime],
    Volume USING [ID];

Scavenger: DEFINITIONS =
BEGIN

Error: ERROR [error: ErrorType];

ErrorType: TYPE = {cannotWriteLog, noSuchPage, orphanNotFound, volumeOpen};

Header: TYPE = MACHINE DEPENDENT RECORD [
    volume (0): Volume.ID,
    date (5): System.GreenwichMeanTime,
    incomplete (7: 0..14): BOOLEAN,
    repaired (7: 15..15): BOOLEAN,
    numberOfFiles (10B): LONG CARDINAL];

LogFormat: TYPE = MACHINE DEPENDENT RECORD [
    header (0): Header,
    files (SIZE [Header]): ARRAY [0..0) OF FileEntry];

Problem: TYPE = MACHINE DEPENDENT RECORD [
    trouble (0): SELECT entryType (0: 0..15):* FROM -- "trouble" added due to compiler bug.
    unreadable, missing => [
        first (1): File.PageNumber,
        count (3): File.PageCount],
    duplicate, orphan => [
        id (1): OrphanHandle]
    ENDCASE];

FileEntry: TYPE = MACHINE DEPENDENT RECORD [
    file (0): File.ID,
    numberOfProblems (5): CARDINAL,
    problems (6): ARRAY [0..0) OF Problem];

OrphanHandle: TYPE [2];

DeleteLog: PROCEDURE [volume: Volume.ID];

DeleteOrphanPage: PROCEDURE [volume: Volume.ID, id: OrphanHandle];

GetLog: PROCEDURE [volume: Volume.ID] RETURNS [logFile: File.Capability];

ReadBadPage: PROCEDURE [file: File.ID, page: File.PageNumber, destination: Space.PageNumber];

ReadOrphanPage: PROCEDURE [volume: Volume.ID, id: OrphanHandle, destination: Space.PageNumber]
    RETURNS [file: File.ID, type: File.Type, pageNumber: File.PageNumber, readErrors: BOOLEAN];

RewritePage: PROCEDURE [file: File.ID, page: File.PageNumber, source: Space.PageNumber];

Scavenge: PROCEDURE [volume, logDestination: Volume.ID, repair: BOOLEAN] RETURNS [logFile: File.Capability];

END.
```

LOG

Time: June 17, 1980 10:06 AM *By: Luniewski Action: Created file.*
Time: July 13, 1980 9:08 PM *By: Forrest Action: Add error: to ERROR declaration.*
Time: September 2, 1980 4:58 PM *By: Luniewski Action: Make all records be Machine Dependent and use explicit word numbers within those records.*
Time: October 10, 1980 6:30 PM *By: Luniewski Action: SIZE[] = > #.*

```
-- SDDefs.Mesa  Edited by Sandman on June 30, 1980  2:57 PM
-- Copyright  Xerox Corporation 1979, 1980

SDDefs: DEFINITIONS =
  BEGIN

  -- indices in system data vector (including trap codes)

  SD: POINTER TO ARRAY [0..0) OF UNSPECIFIED = LOOPHOLE[1100B];

  -- [0..37B] are known by microcode

  sBreak: CARDINAL = 0;
  sStackError: CARDINAL = 2;
  sWakeupError: CARDINAL = 3;
  sXferTrap: CARDINAL = 4;
  sUnimplemented: CARDINAL = 5;
  sAllocTrap: CARDINAL = 6;
  sControlFault: CARDINAL = 7;
  sSwapTrap: CARDINAL = 10B;
  sPageFault: CARDINAL = 11B;
  sWriteProtect: CARDINAL = 12B;
  sUnbound: CARDINAL = 13B;
  sZeroDivisor: CARDINAL = 14B;
  sDivideCheck: CARDINAL = 15B;
  sHardwareError: CARDINAL = 16B;
  sProcessTrap: CARDINAL = 17B; -- known by BCPL code

  sBoundsFault: CARDINAL = 20B;
  sPointerFault: CARDINAL = 21B;

  -- Signals

  sSignalList: CARDINAL = 40B;
  sSignal: CARDINAL = 41B;
  sErrorList: CARDINAL = 42B;
  sError: CARDINAL = 43B;
  sReturnErrorList: CARDINAL = 44B;
  sReturnError: CARDINAL = 45B;
  sUnnamedError: CARDINAL = 46B;
  sUncaughtSignal: CARDINAL = 47B;

  -- Instructions

  sBLTE: CARDINAL = 52B;
  sBYTBLTE: CARDINAL = 53B;
  sBLTEC: CARDINAL = 54B;
  sBYTBLTEC: CARDINAL = 55B;
  sBLTEL: CARDINAL = 56B;
  sBYTBLTEL: CARDINAL = 57B;
  sBLTECL: CARDINAL = 60B;
  sBYTBLTECL: CARDINAL = 61B;
  sStringInit: CARDINAL = 62B;
  sSignedDiv: CARDINAL = 63B;
  sLongMul: CARDINAL = 64B;
  sLongDivMod: CARDINAL = 65B;
  sLongDiv: CARDINAL = 66B;
  sLongMod: CARDINAL = 67B;
  sULongDivMod: CARDINAL = 70B;
  sULongDiv: CARDINAL = 71B;
  sULongMod: CARDINAL = 72B;
  sLongStringCheck: CARDINAL = 73B;

  -- Frames

  sCopy: CARDINAL = 75B;
  sUnNew: CARDINAL = 76B;
  sStart: CARDINAL = 77B;
  sRestart: CARDINAL = 100B;
  sGFTLength: CARDINAL = 101B;

  -- Debugger

  sAlternateBreak: CARDINAL = 103B;
  sCoreSwap: CARDINAL = 104B;
  sProcessBreakpoint: CARDINAL = 105B;
  sCallDebugger: CARDINAL = 106B;
  sWorryCallDebugger: CARDINAL = 107B;
```

```
  sInterrupt: CARDINAL = 110B;
  sGoingAway: CARDINAL = 111B; -- known by BCPL code

  sAddFileRequest: CARDINAL = 112B;
  sIOResetBits: CARDINAL = 113B;
  sBreakBlock: CARDINAL = 114B;
  sBreakBlockSize: CARDINAL = 115B;
  sPerfMonitor: CARDINAL = 116B;
  sLogging: CARDINAL = 117B;
  sXferTrapMonitor: CARDINAL = 120B;
  sCrossMDSLow: CARDINAL = 121B;
  sCrossMDSHigh: CARDINAL = 122B;

  -- Processes

  sFork: CARDINAL = 124B;
  sJoin: CARDINAL = 125B;

  -- Floating Point

  sFADD: CARDINAL = 130B;
  sFSUB: CARDINAL = 131B;
  sFMUL: CARDINAL = 132B;
  sFDIV: CARDINAL = 133B;
  sFCOMP: CARDINAL = 134B;
  sFIX: CARDINAL = 135B;
  sFLOAT: CARDINAL = 136B;

  sFirstCedar: CARDINAL = 150B;
  sLastCedar: CARDINAL = 207B;

  sFirstPilot: CARDINAL = 230B;
  sLastPilot: CARDINAL = 277B;

  sLastSD: CARDINAL = 277B;

  END...
```

```
-- This interface supplies I/O operations for those not in a postion to use the facilities of the VMMgr.
-- Note that if Map/Unmap are invoked from a process that holds the LogicalVolume monitor lock, the required File Descriptors and
   Page Group Descriptors must be pinned in the FileCache (since the File Helper is locked out).

DIRECTORY
    CachedSpace USING [Handle, SizeSwapUnit],
    ResidentMemory USING [Location],
    Space USING [Handle, PageCount, PageNumber, WindowOrigin],
    Utilities USING [LongPointerFromPage];

SimpleSpace: DEFINITIONS
    IMPORTS Utilities =
BEGIN

Handle: TYPE = Space.Handle;
SizeSwapUnit: TYPE = CachedSpace.SizeSwapUnit;
WindowOrigin: TYPE = Space.WindowOrigin;

--~~~~~~~ Operations usable only during Pilot initialization: ~~~~~~

Location: TYPE = ResidentMemory.Location;

AllocateVM: PROCEDURE [count: Space.PageCount, location: Location] RETURNS [page: Space.PageNumber];
    -- Allocates VM only. You must do a StoragePrograms.DescribeSpace[...] yourself.
Create: PROCEDURE [count: Space.PageCount, location: Location, sizeSwapUnit: SizeSwapUnit ← noSwapUnits]
            RETURNS [handle: Space.Handle];
    -- Creates a simpleSpace.
noSwapUnits: SizeSwapUnit = 0;

DisableInitialization: PROCEDURE;
    -- Must be executed after the last simple space has been created but before the contents of the Swapper space and region caches
       are copied up into the VMMgr databases.

--~~~~~~~ Operations usable at all times: ~~~~~~

CopyIn: PROCEDURE [handle: Space.Handle, window: Space.WindowOrigin, countMapped: Space.PageCount ← defaultCount];
CopyOut: PROCEDURE [handle: Space.Handle, window: Space.WindowOrigin, countMapped: Space.PageCount ← defaultCount];
    -- Analogous to Space.CopyIn / CopyOut, 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.

ForceOut: PROCEDURE [Space.Handle];
    -- Analogous to Space.ForceOut. This is a no-op if the space is pinned.

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

LongPointer: PROCEDURE [handle: Space.Handle] RETURNS [LONG POINTER] = INLINE
        { RETURN[ Utilities.LongPointerFromPage[ LOOPHOLE[handle, CachedSpace.Handle].page ] ] };
    -- Returns pointer to starting page number of argument simple space.

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.
defaultCount: Space.PageCount = LAST[Space.PageCount];

Page: PROCEDURE [handle: Space.Handle] RETURNS [page: Space.PageNumber] = INLINE
        { RETURN[ LOOPHOLE[handle, CachedSpace.Handle].page ] };
    -- Returns starting page number of argument simple space.

Unmap: PROCEDURE [Space.Handle];
```

```
    -- Analogous to Space.Unmap.

END.
```

LOG

June 19, 1978 5:50 PM	*McJones*	*Created file.*
August 7, 1978 11:44 PM	*Purcell*	*Added DisableCreate, parameters type to Create and andPin to Map.*
August 10, 1978 3:56 PM	*Purcell*	*Moved generalized create to VFSPrograms.DescribeSpace.*
August 29, 1978 1:17 PM	*McJones*	*Added ForceOut.*
March 5, 1979 9:48 AM	*McJones*	*Changed definition of handle.*
August 15, 1979 9:17 AM	*McJones*	*Added countMapped to Map.*
August 21, 1979 10:18 AM	*Knutsen*	*Added comment to Map (not recompiled).*
April 11, 1980 12:17 PM	*Knutsen*	*Added AllocateVM.* Location *gotten from ResidentMemory. Added swap unit parameter to Create.*
August 26, 1980 10:16 AM	*Knutsen*	*Added Kill, LongPointer, CopyIn, CopyOut.*

```
DIRECTORY
    Environment: FROM "Environment" USING [PageCount],
    File: FROM "File" USING [Capability, PageNumber],
    TemporaryBooting: FROM "TemporaryBooting" USING [defaultSwitches, Switches];

Snapshot: DEFINITIONS =

BEGIN

Switches: TYPE = TemporaryBooting.Switches;

defaultSwitches: Switches = TemporaryBooting.defaultSwitches;

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

OutLoad: PROCEDURE [file: File.Capability, firstPage: File.PageNumber ← 0] RETURNS [inLoaded: BOOLEAN];
    -- Save machine state on file, then return FALSE; later returns TRUE when machine state is inLoaded

InLoad: PROCEDURE [pMicrocode, pGerm: LONG POINTER, countGerm: Environment.PageCount,
        file: File.Capability, firstPage: File.PageNumber ← 0, switches: Switches ← defaultSwitches];
    -- Load new microcode and germ (if pMicrocode~ = NIL and pGerm~ = NIL, respectively), put switches in communication area, and
            restore machine state from file
    -- Note: if pMicrocode~ = NIL, it must point to resident memory (e.g. use SpecialSpace.MakeResident)
    -- Errors (e.g. wrong file type, MakeBootable not performed, or contents wrong) will result in crash and possibly a MP code

END.
```

LOG

Time: August 9, 1979 8:51 AM By: McJones Action: Create file
Time: September 18, 1979 4:30 PM By: McJones Action: Add firstPage to InLoad
Time: January 24, 1980 5:33 PM By: McJones Action: Add switches to InLoad

```
DIRECTORY
 Environment USING [Byte],
 System USING [NetworkAddress];

Socket: DEFINITIONS =
BEGIN

-- definitions

-- various types used by socket channels.
ChannelHandle: TYPE [2];

PhysicalRecordHandle: TYPE = POINTER TO PhysicalRecord;
PhysicalRecord: TYPE = MACHINE DEPENDENT RECORD [
 header: OISPktHeaderRecord,
 body: PACKED ARRAY [0 .. 0) OF Byte];

OISPktHeaderRecord: TYPE = MACHINE DEPENDENT RECORD [
 checksum: PRIVATE CARDINAL,
 pktLength: CARDINAL, -- in bytes, includes header
 transportControl: PRIVATE Byte,
 packetType: Byte, -- see OISCPTypes.OisPacketType
 destination, source: System.NetworkAddress];

GetHandle: TYPE [2]; -- returned by Get, used by TransferWait; will become exported type

Byte: TYPE = Environment.Byte;

TransferStatus: TYPE = {
 pending, goodCompletion, aborted, noRouteToNetwork, hardwareProblem,
 invalidDestAddr,
 -- the following apply to circuit-like media only and mostly to the first packet sent
  noAnswerOrBusy, -- auto-dial case only
  noTranslationForDestination, -- no phone number for this destination
  circuitInUse, -- being used to talk to another destination
  circuitNotReady, -- dial the phone or connect modems (non-auto-dial case)
  noDialingHardware,
  dialerHardwareProblem
 };
SocketStatus: TYPE = RECORD [
 localAddr: System.NetworkAddress,
 state: State,
 incompleteGets: CARDINAL];
State: TYPE = {active, aborted};

WaitTime: TYPE = LONG CARDINAL; -- msecs

-- constants used by the client.
uniqueNetworkAddr: READONLY System.NetworkAddress;
defaultWaitTime: WaitTime = 60000; -- msecs
maxInternetOISPktLength: CARDINAL = 576; -- in bytes and includes header

-- interface

-- exported by SocketImpl.
-- procedures
AssignNetworkAddress: PROCEDURE RETURNS [System.NetworkAddress];
Create: PROCEDURE [local: System.NetworkAddress] RETURNS [ChannelHandle];
Delete: PROCEDURE [cH: ChannelHandle];
Put: PROCEDURE [cH: ChannelHandle, recH: PhysicalRecordHandle] RETURNS [TransferStatus];
Get: PROCEDURE [cH: ChannelHandle, recH: PhysicalRecordHandle] RETURNS [GetHandle];
TransferWait: PROCEDURE [cH: ChannelHandle, getH: GetHandle] RETURNS [recH: PhysicalRecordHandle, status: TransferStatus];
TransferWaitAny: PROCEDURE [cH: ChannelHandle] RETURNS [recH: PhysicalRecordHandle, status: TransferStatus];
Abort: PROCEDURE [cH: ChannelHandle];
Reset: PROCEDURE [cH: ChannelHandle];
GetStatus: PROCEDURE [cH: ChannelHandle] RETURNS [SocketStatus];
SetWaitTime: PROCEDURE [cH: ChannelHandle, time: WaitTime];
-- errors
TimeOut: ERROR; -- error when timeout ocurrs on blocked call like TransferWait
ChannelAborted: ERROR; -- error when state is aborted
ChannelError: ERROR; -- error when bad software structuring, or bad parameter to calls
```

END.

LOG

(trimmed to Teak)
Time: January 22, 1980 8:57 PM By: Dalal Action: modified SocketStatus.
Time: January 26, 1980 10:12 AM By: Dalal Action: made ChannelHandle LONG.
Time: March 11, 1980 3:21 PM By: BLyon Action: Put, Get use PhysicalRecordHandles instead of CrateHandles. Added Reset.
Time: June 18, 1980 2:23 PM By: BLyon Action: use System instead of SpecialSystem;
uniqueNetworkAddr from CONSTANT to READONLY, ChannelHandle & GetHandle are opaque types.
Time: August 6, 1980 10:17 AM By: Garlick Action: Added several TransferStatus's for circuit-oriented network errors.
Time: October 10, 1980 4:50 PM By: Garlick Action: Added TransferStatus noAnswerOrBusy

```
-- Function: The internal definitions module for Pilot Socket Channels.

DIRECTORY
  BufferDefs USING [BufferPool, BufferPoolObject],
  OISCPDefs USING [OisAddress, OisBuffer, QueueObject],
  Socket USING [ChannelHandle, State, WaitTime];

SocketInternal: DEFINITIONS =
BEGIN OPEN Socket;

-- definitions

SocketHandle: TYPE = LONG POINTER TO SocketObject;
SocketObject: TYPE = RECORD [
  next: SocketHandle, -- used only by the router
  -- every socket has a buffer pool, which is protected outside the socketRouterLock lock
  poolObject: BufferDefs.BufferPoolObject, -- the buffer pool object if it is a private one
  -- socketRouterLock monitor protected data and condition variables
  bufferPoolType: BufferPoolType,
  localAddr: OISCPDefs.OisAddress,
  channelState: Socket.State,
  waitTime: Socket.WaitTime,
  completedReceiveQueue: OISCPDefs.QueueObject, -- queues allocated via GetFreeReceiveOisBufferFromPool by the dispatcher,
dequeued by GetPacket.
  pendingUserGetQueue: OISCPDefs.QueueObject, -- queues allocated via Socket.Get; dequeued by the dispatcher.
  completedUserGetQueue: OISCPDefs.QueueObject, -- queues gotten from pendingUserGetQueue; dequeued by TransferWait or
TransferWaitAny.
  newReceiveInput: CONDITION,
  newUserInput: CONDITION];

BufferPoolType: TYPE = {normal, crate};

-- interface

-- exported by SocketImpl.
-- procedures
SocketHandleToChannelHandle: PROCEDURE [sH: SocketHandle]
    RETURNS [ChannelHandle] =
  INLINE BEGIN
  RETURN[LOOPHOLE[sH, ChannelHandle]];
  END;
ChannelHandleToSocketHandle: PROCEDURE [cH: ChannelHandle]
    RETURNS [SocketHandle] =
  INLINE BEGIN
  RETURN[LOOPHOLE[cH, SocketHandle]];
  END;
CreateInternal: PROCEDURE [local: OISCPDefs.OisAddress, bufferPoolType: BufferPoolType, send, receive: CARDINAL] RETURNS
[SocketHandle];
GetBufferPool: PROCEDURE [cH: SocketHandle] RETURNS [BufferDefs.BufferPool];
PutPacket: PROCEDURE [cH: SocketHandle, b: OISCPDefs.OisBuffer];
GetPacket: PROCEDURE [cH: SocketHandle] RETURNS [b: OISCPDefs.OisBuffer];
SocketOn: PROCEDURE;

END.

LOG

Time: January 3, 1980 5:58 PM By: Dalal Action: modified interface completely.
Time: March 11, 1980 3:04 PM By: BLyon Action: pendingGetQueue modified to pendingUserGetQueue. send, receive (buffer counts)
added to create internal. Added GetPacket and PutPacket.
Time: March 18, 1980 1:56 PM By: BLyon Action: Added to/ deleted from socket object.
```

```
DIRECTORY
    TextBlt: FROM "TextBlt";

SoftwareTextBlt: DEFINITIONS =
BEGIN OPEN TextBlt;


-- This is a temporary kludge to allow TextBlt to be implemented in Mesa. This procedure is called when a trap on
    unimplemented instruction is encountered and the instruction is TextBlt.

TextBlt: PROCEDURE
    [index: CARDINAL, bitPos: CARDINAL, micaPos: CARDINAL, count: INTEGER, ptr: POINTER]
    RETURNS [newIndex: CARDINAL, newBitPos: CARDINAL, newMicaPos: CARDINAL, newCount: INTEGER, result:
    Result];


END.    -- of SoftwareTextBlt

LOG
July 29, 1980 9:17 AM -- Jim Frandeen -- Created.
August 29, 1980 3:31 PM -- Jim Frandeen -- Reorder parameters.
```

```
DIRECTORY
    Environment USING [PageCount, PageNumber, PageOffset, wordsPerPage],
    File USING [Capability, nullCapability, PageNumber],
    Transaction USING [Handle, nullHandle];

Space: DEFINITIONS IMPORTS Transaction =

BEGIN

    -- Spaces and space handles
wordsPerPage: CARDINAL = Environment.wordsPerPage;
PageCount: TYPE = Environment.PageCount;
PageNumber: TYPE = Environment.PageNumber;
PageOffset: TYPE = Environment.PageOffset;
Handle: TYPE [2];
nullHandle: READONLY Handle;
mds: READONLY Handle;
virtualMemory: READONLY Handle;

    -- Creating and deleting spaces
defaultBase: PageOffset = 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];
CopyIn: PROCEDURE [space: Handle, window: WindowOrigin, transaction: Transaction.Handle ← Transaction.nullHandle];
CopyOut: PROCEDURE [space: Handle, window: WindowOrigin, transaction: Transaction.Handle ← Transaction.nullHandle];
MakeReadOnly: PROCEDURE [space: Handle, transaction: Transaction.Handle ← Transaction.nullHandle];
MakeWritable: PROCEDURE [space: Handle, file: File.Capability, transaction: Transaction.Handle ← Transaction.nullHandle];
Map: PROCEDURE [space: Handle, window: WindowOrigin ← defaultWindow, transaction: Transaction.Handle ← Transaction.nullHandle];
Remap: PROCEDURE [space: Handle, window: WindowOrigin ← defaultWindow, transaction: Transaction.Handle ←
        Transaction.nullHandle];
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,
        spaceNotUnitary, spaceTreeTooDeep, writeProtected};
InsufficientSpace: ERROR [available: PageCount];

END.
```

LOG

Time: March 15, 1978 10:06 AM 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*
Time: January 25, 1980 1:27 PM	*By: Knutsen*	*Action: Added ErrorType[spaceNotUnitary, writeProtected]*
Time: March 31, 1980 4:25 PM	*By: Gobbel*	*Action: Added transaction machinery*
Time: August 5, 1980 11:41 AM	*By: McJones*	*Action: Modified transaction machinery*

```
-- This is the interface between SpaceImplA and SpaceImplB.

DIRECTORY
    CachedRegion USING [Desc, Operation],
    CachedSpace USING [Desc, Handle, Level],
    Space USING [PageNumber],
    VM USING [Interval];

SpaceImplInternal: DEFINITIONS =

BEGIN

    -- Commonly Used TYPEs:

Interval: TYPE = VM.Interval;

Level: TYPE = CachedSpace.Level;

SpaceD: TYPE = CachedSpace.Desc;

RegionD: TYPE = CachedRegion.Desc;

    -- Shared Data:

spaceLock: MONITORLOCK;

    -- Monitor EXTERNAL Procedures:

InitializeSpaceImplB: --EXTERNAL-- PROCEDURE [];

    -- Monitor INTERNAL Procedures:

ApplyToInterval: --INTERNAL-- PROCEDURE [interval: Interval, operation: CachedRegion.Operation];
    -- Performs, in ascending order, operation (successfully) for each swap unit of each region of interval (exactly as required by
       CachedRegion.Apply).
    -- May raise NotePinned (only if operation.action = unmap).
    -- for operation = writeProtect, flush, remap, and unmap, interval must start and end on a region boundary! (restriction of
       CachedRegion.Apply)
NotePinned: SIGNAL [levelMax: Level, page: Space.PageNumber];
    -- The interval being operated on (being unmapped) contains a pinned space. The space contains page and is pinned at some level
       <= levelMax. The operationwas done on the current region. The catcher of the signal should make the Hierarchy reflect the
       fact that this space(s) are no longer pinned, then RESUME the signal.

ApplyToSpace: --INTERNAL-- PROCEDURE [handle: CachedSpace.Handle, operation: CachedRegion.Operation]
                    RETURNS [validHandle: BOOLEAN];
    -- Validates handle, then ApplyToInterval[space.interval, operation].

ForAllRegions: --INTERNAL-- PROCEDURE [interval: Interval, Predicate: RegionPredicate] RETURNS [trueOfAll: BOOLEAN];
    -- Returns TRUE iff predicate is TRUE for every region of interval.
RegionPredicate: TYPE = PROCEDURE [region: RegionD] RETURNS [trueOfRegion: BOOLEAN];

UnmapInternal: --INTERNAL-- PROCEDURE [pSpaceD: POINTER TO READONLY SpaceD];
    -- Unmaps the space. Caller must have done error checking previously.

END.
```

LOG

Time: June 19, 1980 2:34 PM	*By: Gobbel*	*Action: Created file.*
Time: August 4, 1980 1:45 PM	*By: Knutsen*	*Action: Add NotePinned. Deleted unused items. Commented the procedures.*

-- **SpecialCommunication**.mesa. *Last editted by: BLyon on:* August 1, 1980 3:14 PM

```
DIRECTORY
 DriverTypes USING [DeviceType],
 SpecialSystem USING [NetworkNumber];

SpecialCommunication: DEFINITIONS =
 BEGIN OPEN SpecialSystem;

 -- The router is either a plain router or an InterNetworkRouter.
 RoutersFunction: TYPE = {vanillaRouting, interNetworkRouting};

 -- A network is identified by its PhysicalMedium and its physical location on the
 -- network device chain.
 PhysicalMedium: TYPE = DriverTypes.DeviceType;

 -- The router must be told which function to perform. Function is initially vanilla.
 SetRouterFunction: PROCEDURE [newFunction: RoutersFunction] RETURNS [oldFunction: RoutersFunction];

 -- This procedure returns what the router's current function is.
 GetRouterFunction: PROCEDURE RETURNS [RoutersFunction];

 -- The router/drivers may have to be told what its network numbers are.
 -- This is certainly true if this is the first machine running on a network.
 -- Physical order is the location of the network on the network chain (starting at 1).
 -- I.E. GetNetworkID[2, ethernet] returns the OisNetID of the second ethernet network
 -- driver. Getting or setting a NetworkID before the network exists is an error.
 SetNetworkID: PROCEDURE [physicalOrder: CARDINAL, medium: PhysicalMedium, newNetID: NetworkNumber]
  RETURNS [oldNetID: NetworkNumber];

 GetNetworkID: PROCEDURE [physicalOrder: CARDINAL, medium: PhysicalMedium] RETURNS [NetworkNumber];

 NetworkNonExistent: ERROR;

 END.
```

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

SpecialFile: DEFINITIONS =

BEGIN

MakeBootable: PROCEDURE [file: File.Capability, firstPage: File.PageNumber ← 0, count: File.PageCount, lastLink: Link ← eofLink]
    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];

eofLink: Link = [LAST[LONG CARDINAL]]; -- all one's

MakeUnbootable: PROCEDURE [file: File.Capability, firstPage: File.PageNumber ← 0, count: File.PageCount];
    -- 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: Create file
Time: January 24, 1980 6:40 PM By: McJones Action: MakeUnBootable = > MakeUnbootable, add firstPage and count; add eofLink

```
SpecialHeap: DEFINITIONS =
BEGIN
MakeResident: PROCEDURE [z: UNCOUNTED ZONE];
MakeResidentMDS: PROCEDURE [z: MDSZone];
MakeSwappable: PROCEDURE [z: UNCOUNTED ZONE];
MakeSwappableMDS: PROCEDURE [z: MDSZone];
END.
```

LOG

Time: May 23, 1980 11:56 AM	*By: McJones*	*Action: Created file*
Time: July 8, 1980 5:21 PM	*By: McJones*	*Action: Adapted for new-style uncounted zones*

```
DIRECTORY
    Space USING [defaultBase, Handle, PageCount, PageNumber, PageOffset];

SpecialSpace: DEFINITIONS =

BEGIN

CreateForCode: PROCEDURE [size: Space.PageCount, parent: Space.Handle, base: Space.PageOffset ← Space.defaultBase]
        RETURNS [newSpace: Space.Handle];
    -- Create a space guaranteed not to cross a 64K boundary.

CreateAligned: PROCEDURE [size: Space.PageCount, parent: Space.Handle] RETURNS [newSpace: Space.Handle];
    -- Create a space which begins at a page within virtual memory which is an integral multiple of the smallest power of two not less
    than the given size. (Thus if size is already a power of two, the beginning page number will have at least as many low-order zero
    bits as does size.)

MakeResident, MakeSwappable: PROCEDURE [space: Space.Handle];
    -- Make the space resident/swappable.

MakeCodeResident, MakeCodeSwappable: PROCEDURE [frame: PROGRAM];
    -- Make the swap unit containing the first word of the code segment of the program resident/swappable.

MakeGlobalFrameResident, MakeGlobalFrameSwappable: PROCEDURE [frame: PROGRAM];
    -- Make the swap unit containing the global frame of the program resident/swappable.

MakeProcedureResident, MakeProcedureSwappable: PROCEDURE [proc: --procedure PrincOps.ControlLink-- UNSPECIFIED];
    -- Make the swap unit containing the first word of the code of the procedure resident/swappable.

DonateDedicatedRealMemory: PROCEDURE [page: Space.PageNumber, size: Space.PageCount];
    -- Take any real memory mapped (in the PrincOps sense) to the specified virtual pages and move it to Pilot's swapping pool. The
    virtual pages should lie within a space which is not mapped (in the Pilot sense); on return they will no longer be mapped (in the
    PrincOps sense). This operation is useful for hardware configurations containing dedicated real memory not normally accessed
    through the virtual memory (e.g. special I/O buffers) when an application doesn't use the function for which the real memory is
    dedicated and could use the extra swapping memory.

realMemorySize: READONLY Space.PageCount;
    -- Maximum amount of real memory which might be present in virtual memory, including dedicated real memory (e.g. Dandelion
    display bitmap, but not Dandelion page map).

END.
```

LOG

Time: August 28, 1978 10:07 AM	*By: Lauer*	*Action: Created file*
Time: July 10, 1979 12:06 PM	*By: Knutsen*	*Action: Made compatible with Teak interfaces*
Time: August 9, 1979 6:41 PM	*By: McJones*	*Action: Added realMemorySize*
Time: January 25, 1980 7:49 PM	*By: Forrest*	*Action: Added default base for CreateForCode*
Time: June 5, 1980 2:00 PM	*By: Knutsen*	*Action: Added MakeGlobalFrame*, MakeProcedure*, etc*
Time: June 17, 1980 1:15 PM	*By: McJones*	*Action: Added CreateAligned*
Time: September 2, 1980 3:35 PM	*By: McJones*	*Action: Added DonateDedicatedRealMemory*

```
SpecialSystem: DEFINITIONS =

BEGIN

    -- Network addresses
NetworkAddress: TYPE = MACHINE DEPENDENT RECORD [
    net: NetworkNumber,
    host: HostNumber,
    socket: SocketNumber];
nullNetworkAddress: NetworkAddress = [net: nullNetworkNumber, host: nullHostNumber, socket: [0]];

NetworkNumber: TYPE = MACHINE DEPENDENT RECORD [a, b: UNSPECIFIED];
nullNetworkNumber: NetworkNumber = [0, 0];

HostNumber: TYPE = MACHINE DEPENDENT RECORD [SELECT type: * FROM
    physical => [a: CARDINAL[0..32768), b: CARDINAL, c: CARDINAL],
    multicast => [a: CARDINAL[0..32768), b: CARDINAL, c: CARDINAL],
    ENDCASE];
    -- A physical host number is assigned to an actual system element.  A multicast host number names a logical group all wishing to
            receive a particular class of multicast packets.
nullHostNumber: physical HostNumber = [physical[0, 0, 0]];
    -- The number of no host; must be kept consistent with the communication interfaces!

MulticastID: TYPE = multicast HostNumber ← NULL;
broadcastHostNumber: HostNumber = [multicast[LAST[CARDINAL[0..32768)], LAST[CARDINAL], LAST[CARDINAL]]];
    -- A special host number used for broadcasting messages; must be kept consistent with the communication interfaces!

SocketNumber: TYPE = MACHINE DEPENDENT RECORD [UNSPECIFIED];
nullSocketNumber: SocketNumber = [0];

    -- Processor identifiers
ProcessorID: TYPE = physical HostNumber ← NULL;
nullProcessorID: ProcessorID = nullHostNumber;

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

END.
```

LOG

Time: January 25, 1980 3:12 PM By: Dalal Action: Created file
Time: January 27, 1980 7:12 PM By: Forrest Action: Corrected file creation date
Time: May 9, 1980 1:36 PM By: McJones Action: Changed to 48-bit processor id, added network address types
Time: September 2, 1980 4:56 PM By: McJones Action: Add NULL default to ProcessorID, MulticastID so they can be used within implementations of exported types

-- *The items in this interface are intended to assist in debugging both transaction clients, and the transaction facility itself. They are* ***not*** *for use other than debugging.*

-- *How this is supposed to work:*
-- *The transaction facility calls* optionalCrash *(if non-NIL) at various key places in its code.* optionalCrash *may then crash or not, as it sees fit. The parameter* unimportance *is supplied as a hint indicating the relative "interestingness" of the point from which* optionalCrash *was called.* ***Warning:*** *Since* optionalCrash *is called from deep within the transaction machinery, it should* ***not*** *attempt to call any Pilot operations in making its decision, as this may result in deadlock.*

```
SpecialTransaction: DEFINITIONS =

BEGIN

ClientStart: PROCEDURE;
    -- To be exported by transaction client: If this procedure is bound, starts client before transaction crash recovery, so he can
        execute SetCrashProcedure.

SetCrashProcedure: PROCEDURE [optionalCrash: OptionalCrash ← NullProc];
    -- optionalCrash will be called at critical points in the transaction code. The OptionlCrash takes an Unimportance as argument,
        which it may use as a hint in deciding whether or not to crash at that point.

NullProc: OptionalCrash;
    -- does nothing, causes a call to SetCrashProcedure with no argument to effectively disable the bomb facility.

OptionalCrash: TYPE = PROCEDURE [unimportance: Unimportance];

Unimportance: TYPE = CARDINAL [1..LAST[CARDINAL]];

END.
```

LOG

September 25, 1980 11:56 AM Gobbel Create file.

```
DIRECTORY
    Boot USING [LVBootFiles, PVBootFiles],
    PhysicalVolume USING [ID, PageNumber],
    Volume USING [ID, nullID, PageCount];

SpecialVolume: DEFINITIONS =
BEGIN

LogicalVolumePageNumber: TYPE = LONG CARDINAL;
SubVolume: TYPE = RECORD[
    lvID: Volume.ID,
    subVolumeSize: Volume.PageCount,
    firstLVPageNumber: LogicalVolumePageNumber,
    firstPVPageNumber: PhysicalVolume.PageNumber];
nullSubVolume: SubVolume = [Volume.nullID, LAST[Volume.PageCount], LAST[LogicalVolumePageNumber],
        LAST[PhysicalVolume.PageNumber]];

SubVolumeUnknown: ERROR;

GetNextSubVolume: PROCEDURE [pvID: PhysicalVolume.ID, this: SubVolume] RETURNS [next: SubVolume];

-- Get/set boot file pointers
GetLogicalVolumeBootFiles: PROCEDURE [lvID: Volume.ID, pBootFiles: LONG POINTER TO Boot.LVBootFiles];
GetPhysicalVolumeBootFiles: PROCEDURE [pvID: PhysicalVolume.ID, pBootFiles: LONG POINTER TO Boot.PVBootFiles];
SetLogicalVolumeBootFiles: PROCEDURE [lvID: Volume.ID, pBootFiles: LONG POINTER TO Boot.LVBootFiles];
SetPhysicalVolumeBootFiles: PROCEDURE [pvID: PhysicalVolume.ID, pBootFiles: LONG POINTER TO Boot.PVBootFiles];

END.
```

LOG
Time: May 15, 1980 4:40 PM By: McJones Action: Equate PhysicalVolume.ID *to same type in System*
Time: June 28, 1980 5:23 PM By: Forrest Action: Trim long. Eliminate procedures now defined in other interfaces, and ReGroup adding some comments about moving Procedures. Move [Get|Set][Logical|Physiccal]VolumeBootFiles to KernelFile, GetContaining PhysicalVolume to PhysicalVolume.
Time: July 13, 1980 9:15 PM By: Forrest Action: Move most procedures elsewhere.
Time: July 28, 1980 2:33 PM By: Luniewski Action: Added nullSubVolume, SubVolumeUnknown.

```
-- File: StatsDefs.mesa,  Last Edit: HGM September 14, 1980  12:38 AM

StatsDefs: DEFINITIONS =
BEGIN

WindowHandle: TYPE = POINTER; -- Beware of compilation order problems

-- these routines bump the actual counters
StatIncr: PROCEDURE [StatCounterIndex];
StatBump: PROCEDURE [StatCounterIndex, CARDINAL];
StatLog: PROCEDURE [WindowHandle, StatCounterIndex, POINTER, CARDINAL];
StatGetCounter: PROCEDURE [StatCounterIndex] RETURNS [LONG CARDINAL];

StatNew: PROCEDURE;  -- call before MakeImage to remember Date+Time
StatStart: PROCEDURE [WindowHandle, STRING];  -- prints header line, resets counters
StatPrintCurrent: PROCEDURE [WindowHandle];  -- prints totals so far
StatFinish: PROCEDURE [WindowHandle];  -- prints grand totals

-- there is also a second parallel set of counters that can be used
-- for local timings without klobbering the totals for the whole run
StatReady: PROCEDURE;  -- resets extra set of counters
StatSince: PROCEDURE [WindowHandle];  -- print, then reset extra set of counters


-- extra hooks for do it yourself types

-- Allocates one of the spares if the STRING is not already known.
-- NB: The STRING is NOT copied over.  Be sure it does not go away.
StatsStringToIndex: PROCEDURE [STRING] RETURNS [StatCounterIndex];
  StatsStringsFull: ERROR;

StatUpdate: PROCEDURE;
StatsGetCounters: PROCEDURE RETURNS [
  POINTER TO ARRAY StatCounterIndex OF LONG CARDINAL];
StatsGetText: PROCEDURE RETURNS [
  POINTER TO ARRAY StatCounterIndex OF STRING];
```

```
--  ******** keep Text in StatsStrings up to date too!!

StatCounterIndex: TYPE = {

-- General
statMouseTrap,  -- for anybody to use
statTime,               -- Time in MiliSeconds
statSeconds,    -- Time in Seconds
statHours,      -- Time in Hours

-- Queue Package
statEnqueue, statDequeue, statXqueue, statDequeueNIL, statXqueueNIL,
statNoBuffer, statBufferWaits,


-- Router
statZappedP,  -- Lightning
statJunkBroadcastPups,  statJunkBroadcastOis,
statJunkPupsForUsNoLocalSocket,  statJunkOisForUsNoLocalSocket,
statPacketsDiscarded,  statPupsDiscarded,  statOisDiscarded,
statPupReceived,   statOisReceived,
statPupForwarded,   statOisForwarded,
statPupSent,   statOisSent,
statPupBroadcast,   statOisBroadcast,
statPupGatewayPacketsRecv,   statOisGatewayPacketsRecv,
statErrorPupsSent,  statOisErrorPacketsSent,  -- not via stream
statPupsSentNowhere,  statOisSentNowhere,
statPupNotForwarded,  statOisNotForwarded,
statPupInputQueueOverflow,  statOisInputQueueOverflow,
statReceivedBadPupChecksum,  statReceivedBadOisChecksum,

-- Low level debugging
pupsEchoed, pupsBadEchoed, pupBytesEchoed,
packetsEchoed, packetsBadEchoed, bytesEchoed,

-- EtherNet Hardware Interface
statEtherPacketsSent, statEtherWordsSent,
statEtherPacketsReceived, statEtherWordsReceived,
statEtherPacketsLocal, statEtherWordsLocal,
statEtherInUnderOut,

-- Collisions
statEtherSendsCollision1, statEtherSendsCollision2, statEtherSendsCollision3,
statEtherSendsCollision4, statEtherSendsCollision5, statEtherSendsCollision6,
statEtherSendsCollision7, statEtherSendsCollision8, statEtherSendsCollision9,
statEtherSendsCollision10, statEtherSendsCollision11, statEtherSendsCollision12,
statEtherSendsCollision13, statEtherSendsCollision14, statEtherSendsCollision15,
statEtherSendsCollisionLoadOverflow,

-- Ethernet driver Glitches
statEtherLostInterrupts,
statEtherMissingStatus,
statEtherInterruptDuringInterrupt,
statResetDidntPost,
statInterfaceReset,
statPacketsStuckInOutput,
statInputIdle,

-- receiver funnys
statEtherReceivedNot16,
statEtherReceivedNot16BadCRC,
statEtherReceivedBadCRC,
statEtherReceivedOverrun,
statEtherReceivedBadStatus,
statEtherReceivedTooLong,
statEtherReceivedKlobberedByReset,
statEtherEmptyFreeQueue,
statEtherEmptyNoBuffer,
statEtherEmptyInputChain,

-- transmitter funnys
statEtherSendWhileReceiving,
statEtherSendOverrun,
statEtherSendBadStatus,
statEtherSendFromOutputQueue,
```

```
-- 8/32/48 translation things
cacheFault, unsuccessfulTranslation, translationRetries, requestsForMe, cacheDepth,
```

```
-- Streams
statDataPacketsSent, statDataBytesSent,
statDataPacketsReceived, statDataBytesReceived,
statMarksSent, statMarksReceived,
statAckRequestsSent, statAckRequestsReceived,
statAcksSent, statAcksReceived,
statSystemPacketsSent, statSystemPacketsReceived,
statAttentionsSent, statAttentionsReceived,

statDataPacketsRetransmitted,
statErrorPacketsSent, statErrorPacketsReceived,

statDataPacketsReceivedAgain,
statDataPacketsReceivedEarly,
statDataPacketsReceivedVeryLate,

statProbesSent, statProbesReceived, statEmptyFunnys,
statEmptyAlloc, statDuplicateAcks,

statPacketsRejectedBadID,
statPacketsRejectedBadSource,
statPacketsRejectedBadType,


-- spares for other use
statSpares0, statSpares1, statSpares2, statSpares3, statSpares4,
statSpares5, statSpares6, statSpares7, statSpares8, statSpares9,
statSpares10, statSpares11, statSpares12, statSpares13, statSpares14,
statSpares15, statSpares16, statSpares17, statSpares18, statSpares19,
statSpares20, statSpares21, statSpares22, statSpares23, statSpares24,
statSpares25, statSpares26, statSpares27, statSpares28, statSpares29,
statSpares30, statSpares31, statSpares32, statSpares33, statSpares34,
statSpares35, statSpares36, statSpares37, statSpares38, statSpares39,
statSpares40, statSpares41, statSpares42, statSpares43, statSpares44,
statSpares45, statSpares46, statSpares47, statSpares48, statSpares49,
statSpares50, statSpares51, statSpares52, statSpares53, statSpares54,
statSpares55, statSpares56, statSpares57, statSpares58, statSpares59,
statSpares60, statSpares61, statSpares62, statSpares63, statSpares64,
statSpares65, statSpares66, statSpares67, statSpares68, statSpares69,
statSpares70, statSpares71, statSpares72, statSpares73, statSpares74,
statSpares75, statSpares76, statSpares77, statSpares78, statSpares79,
statSpares80, statSpares81, statSpares82, statSpares83, statSpares84,
statSpares85, statSpares86, statSpares87, statSpares88, statSpares89,
statSpares90, statSpares91, statSpares92, statSpares93, statSpares94,
statSpares95, statSpares96, statSpares97, statSpares98, statSpares99

};

END.
```

```
-- File: StatsOps.mesa,  Last Edit: Fay October 13, 1980  12:20 PM

DIRECTORY
  StatsDefs USING [StatCounterIndex];

StatsOps: DEFINITIONS =
BEGIN OPEN StatsDefs;

statLock: MONITORLOCK;
statGrand: ARRAY StatCounterIndex OF LONG CARDINAL;
statText: ARRAY StatCounterIndex OF STRING;

StatPrintCounters: PROCEDURE [
    UNSPECIFIED --Window.Handle--, POINTER TO ARRAY StatCounterIndex OF LONG CARDINAL];

StatsHot, StatsCold, StatsStrings, StatsPrint: PROGRAM;

END.
```

```
-- This interface defines types for the Heirarchy and Projection B-trees, and routines for creating, etc., leaves of the B-trees.

-- Note that the elements of a projection desc are a subset of the elements of a CachedRegion.Desc.

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

STLeaf: DEFINITIONS =
BEGIN

ILeaf: TYPE = RECORD [CARDINAL]; -- index of a Leaf

PLeaf: TYPE = LONG POINTER TO Leaf; -- pointer to a Leaf

sizeLeaf: CARDINAL = Environment.wordsPerPage;

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 => -- the elements of projection desc are a subset of the elements of CachedRegion.Desc.
        [page: VM.PageNumber, -- key
        level: CachedSpace.Level,
        levelMapped: CachedSpace.Level,
        hasSwapUnits: BOOLEAN,
        state: ProjectionState,
        writeProtected: BOOLEAN,
        needsLogging: BOOLEAN ],
    ENDCASE ];

alive: CachedRegion.State = CachedRegion.State[outAlive]; -- in the Projection, we only know that the region is some unknown flavor of alive (out, in,
            pinned).

ProjectionState: TYPE = CachedRegion.State[CachedRegion.State[unmapped]..alive];

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:* McJones	*Action: Created file*
Time: June 24, 1978 5:38 PM	*By:* McJones	*Action: PageNumber moved to VM*
Time: August 4, 1978 2:14 PM	*By:* McJones	*Action: Added beingRemapped to projection Desc*
Time: September 26, 1979 1:53 PM	*By: Knutsen*	*Action: Added* hasSwapUnits, writeProtected *to, deleted* beingRemapped *from projection Desc. Added* alive.
Time: July 7, 1980 4:10 PM	*By: Gobbel*	*Action: Added* needsToBeLogged *to projection Desc.*
Time: July 31, 1980 10:50 AM	*By: Knutsen*	*Action: Made compatible with new CachedRegion.Desc.*

```
DIRECTORY
    Boot USING [DiskFileID, Location, LVBootFiles],
    Device USING [Type],
    Environment USING [Long, PageCount, PageNumber , wordsPerPage],
    Inline USING [LongMult],
    Space: FROM "Space" USING [Handle, WindowOrigin],
    StartList: FROM "StartList" USING [Index, Base];

StoragePrograms: DEFINITIONS
    IMPORTS Inline =

BEGIN

-- Subcomponents of the Store configuration:

InitializeFileMgr: PROCEDURE [bootFile: LONG POINTER TO disk Boot.Location, pLVBootFiles: POINTER TO Boot.LVBootFiles]
    RETURNS [debuggerDeviceType: Device.Type, debuggerDeviceOrdinal: CARDINAL];
    -- If bootFile.deviceType~ = Device.nullType then set system volume to logical volume containing bootFile.id.da.  If no installed
        debugger can be found then return debuggerDeviceType = nullDevice, else return type, handle, and boot file id's from
        our debugger's system volume.

InitializeFiler: PROCEDURE;

InitializeMStore: PROCEDURE; -- real memory management.

InitializeRegionCacheA: PROCEDURE; -- creates region cache, initializes AllocateRuthlessly.
InitializeRegionCacheB: PROCEDURE; -- describes region cache space to Swapper.

InitializeResidentMemoryA: PROCEDURE; -- (On return, ResidentMemory is fully functional.)
InitializeResidentMemoryB: PROCEDURE; -- Describes its space(s) to the Swapper.

InitializeSimpleSpace: PROCEDURE;

InitializeSwapper: PROCEDURE [pMapLogDesc: LONG POINTER];

InitializeTransactionData: PROCEDURE;

InitializeVMMgr: PROCEDURE [countVM: Environment.PageCount, pMapLogDesc: LONG POINTER];

RecoverTransactions: PROCEDURE;

ReplacementProcess: PROCEDURE [threshold: Environment.PageCount];


-- Facilities used during initialization of the components of the Store configuration:


-- Exported by PilotControl:

countVM: Environment.PageCount; -- amount of virtual memory implemented by the current processor.

pageMDS: Environment.PageNumber; -- first page of Pilot's (only) MDS.

lpMDS: LONG POINTER; -- to Pilot's (only) MDS.

pageHyperspace: Environment.PageNumber; -- first page following Pilot's MDS.

countHyperspace: Environment.PageCount;

pageMDSGerm: Environment.PageNumber; -- first page of Germ's MDS.

tableBase: StartList.Base; -- base of the Start List.

StartListProc: TYPE = PROCEDURE [index: StartList.Index];
EnumerateStartList: PROCEDURE [Proc: StartListProc];
    -- Calls Proc once for each entry in the StartList.

PointerFromPage: PROCEDURE [page: Environment.PageNumber] RETURNS [p: POINTER];


-- Exported by SimpleSpaceImpl:

nullSpaceHandle: Space.Handle; -- for use before Space.nullHandle is initialized.

SpaceOptions: TYPE = RECORD [
    createSpace, pinSpaceD, -- options for space creation.
    mapped, -- options for space and region creation.
    createRegion, pinRegionD, subspace, initiallyResident, pinned, -- options for region creation.
    mStoreDeallocate, -- do an MStore.Deallocate on this interval.
```

```
    emptyInterval : -- an unused VM.Interval useable for creation of simpleSpaces.
            BOOLEAN ← FALSE];
    -- Note: subspace = TRUE indicates that the primary space (family space) is divided into smaller swap unit spaces/regions.

createSimpleSpace: SpaceOptions =
    [createSpace: TRUE, pinSpaceD: TRUE, mapped: FALSE, createRegion: TRUE, pinRegionD: TRUE, subspace: FALSE];

empty: SpaceOptions = [emptyInterval: TRUE]; -- an unused VM.Interval useable for creation of simpleSpaces.

free: SpaceOptions = [mStoreDeallocate: TRUE]; -- do an MStore.Deallocate on this interval.

outlaw: SpaceOptions = -- a space whose contents will not be managed by the Swapper.
    [createSpace: TRUE, pinSpaceD: FALSE, mapped: FALSE, createRegion: TRUE, pinRegionD: FALSE, subspace: FALSE];

-- Creates entries in VM databases (space and region caches) describing initially resident VM spaces allocated by StartPilot.

DescribeSpace: PROCEDURE
    [options: SpaceOptions, page: Environment.PageNumber, count: Environment.PageCount, window: Space.WindowOrigin];
    -- Note: a space or region is made readOnly iff the window's permissions only allow reading.
    -- The initially resident memory allocated and filled by StartPilot is organized into spaces, possibly tiled with smaller swap unit
       subspaces. Information describing this initially resident memory must be inserted into the space and region caches by
       DescribeSpace, which only supports this organization of spaces. In particular, there must be a space descriptor created for
       each mapped space, and a region descriptor created for each unitary mapped space and for each subspace of a mapped space.
       No other space or region descriptors should be created.

HandleFromPage: PROCEDURE [page: Environment.PageNumber] RETURNS [handle: Space.Handle];
    -- Returns handle corresponding to the unitary or family space starting at the given page. (See StartList.mesa.)

SuperFromPage: PROCEDURE [page: Environment.PageNumber] RETURNS [handle: Space.Handle, superPage:
        Environment.PageNumber];
    -- Returns handle and first page number of the parent space of the unitary or family space starting at the given page. (See
       StartList.mesa.)


-- Exported by MStore:

RecoverMStore: PROCEDURE;
    -- Places all free real memory into arbitrary vacant virtual pages. Caller must first disable interrupts.
    -- Allocation table is not updated since another system is about to be booted.


-- Exported by VMMControl and UtilityVMMControl and DiagnosticPilotImpl:

IsUtilityPilot: PROCEDURE RETURNS [BOOLEAN];

IsDiagnosticPilot: PROCEDURE RETURNS [BOOLEAN];


-- INLINES:

LongPointerFromPage: PROCEDURE [page: Environment.PageNumber] RETURNS [lp: LONG POINTER] = INLINE
    { RETURN[LOOPHOLE[Inline.LongMult[page, Environment.wordsPerPage]]] };

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

END.
```

LOG

Time: July 14, 1978 10:01 AM	*By: McJones*	*Action: Create file*
Time: July 17, 1978 5:30 PM	*By: McJones*	*Action: Add countReal to SwapperControl, VMMControl*
Time: August 10, 1978 5:13 PM	*By: Purcell*	*Action: Add DesribeSpace*
Time: August 16, 1978 8:01 PM	*By: Purcell*	*Action: Add pMapLogDesc*
Time: August 29, 1978 11:33 AM	*By: McJones*	*Action: Add ReplacementProcess; added pMapLogDesc to SwapperControl*
Time: September 29, 1978 5:23 PM	*By: McJones*	*CR20.45: Add UnmapDataSpaces*
Time: July 31, 1979 11:16 AM	*By: McJones*	*Action: Add boot file id's to FileImpl*
Time: August 23, 1979 9:12 AM	*By: Knutsen*	*Action: Improve readability of SpaceOptions table.*
Time: August 27, 1979 10:58 AM	*By: Knutsen*	*Action: Change VMMControl: PROGRAM to InitVMMgr: PROCEDURE*
Time: September 18, 1979 3:18 PM	*By: McJones*	*Action: Change FileImpl parameters to use types from Boot*
Time: January 31, 1980 1:15 PM	*By: Knutsen*	*Action: Pass VM for Region Cache to SwapperControl.*
Time: February 5, 1980 3:23 PM	*By: McJones*	*Action: Replace FileMgr debugger device result with type and ordinal*
Time: April 16, 1980 9:04 AM	*By: Knutsen*	*Action: Added InitializeResidentMemoryA/B, InitializeRegionCacheA/B, InitializeMStore, InitializeSimpleSpace, RecoverMStore, StartListProc, countVM, pageMDS, pageMDSGerm, lpMDS, pageHyperspace, countHyperspace, EnumerateStartList, tableBase. InitializeSwapper takes fewer parameters. Changed IsBoundIfNotUtilityPilot to*

IsUtilityPilot, added IsDiagnosticPilot.
Time: July 1, 1980 4:55 PM By: Knutsen Action: New StartList format. Redo SpaceOptions. Made inlines really inline.
Time: July 7, 1980 5:13 PM By: Gobbel Action: Added InitializeTransactionData, RecoverTransactions.

```
StoreDriverStartChain: DEFINITIONS =

BEGIN

-- NOTE: When AR 3536 is fixed, the modules HeadStartChain, DriverStartChain, and StoreDriverStartChain may be
replace with single module StartChain.

-- Mechanism used to implement starting extensible set of modules (e.g. heads, drivers, test programs)

-- We refer to a member of the set of modules as a "link"; we refer to the module wishing to start the set as the "master link".
-- Each link should export Start with a body consisting simply of a call on Start imported through a second instance of StartChain (e.g.
     IMPORTS RemainingHeads: StartChain).
-- The master link should export Start with a body which just returns and should also contain a call on Start imported through a second
     instance of StartChain named EntireStartChain.
-- The machinery is completed with a configuration which sets up a chain of StartChain imports-exports from EntireStartChain, through
     each of the links, and back to the master link.

Start: PROCEDURE;
     -- Start the (rest of the) chain

END.
```

LOG

Time: January 31, 1980 11:09 AM By: McJones Action: Create file

```
DIRECTORY
    Environment USING [Block, Byte, Word];

Stream: DEFINITIONS =

BEGIN

        -- Types
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};

        -- Operations
GetByte: PROCEDURE [sH: Handle] RETURNS [byte: Byte] = INLINE {RETURN[sH.getByte[sH]]};
    -- Get the next byte from the stream.
GetChar: PROCEDURE [sH: Handle] RETURNS [char: CHARACTER] = INLINE {RETURN[LOOPHOLE[sH.getByte[sH]]]};
    -- Get the next character from the stream.

GetWord: PROCEDURE [sH: Handle] RETURNS [word: Word] = INLINE {RETURN[sH.getWord[sH]]};
    -- Get the next two bytes from the stream, and return them in a word (first in left half).
    GetBlock: PROCEDURE [sH: Handle, block: Block]
        RETURNS [bytesTransferred: CARDINAL, why: CompletionCode, sst: SubSequenceType] = INLINE {[bytesTransferred, why, sst]
        ← sH.get[sH, block, sH.options]};
    -- Get the next zero or more bytes from the stream, moving them to the specified block of memory.

SetInputOptions: PROCEDURE [sH: Handle, options: InputOptions] = INLINE {sH.options ← options};
    -- Set the input options of the stream.

PutByte: PROCEDURE [sH: Handle, byte: Byte] = INLINE {sH.putByte[sH, byte]};
    -- Put a byte at the next available position in the stream.

PutChar: PROCEDURE [sH: Handle, char: CHARACTER] = INLINE
        {sH.putByte[sH, LOOPHOLE[char]]};
    -- Put a character at the next available position in the stream.

PutWord: PROCEDURE [sH: Handle, word: Word] = INLINE {sH.putWord[sH, word]};
    -- Put two bytes into the next available positions in the stream, left half of word first.

PutBlock: PROCEDURE [sH: Handle, block: Block, endPhysicalRecord: BOOLEAN ← FALSE] = INLINE {sH.put[sH, block,
        endPhysicalRecord]};
    -- Put zero or more bytes into the next available positions in the stream, moving them from a block in memory.

SendNow: PROCEDURE [sH: Handle] = INLINE {bl: Block = [NIL, 0, 0]; sH.put[sH, bl, TRUE]};
    -- Terminate the current physical record.

SetSST: PROCEDURE [sH: Handle, sst: SubSequenceType] = INLINE {sH.setSST[sH, sst]};
    -- Change the current SubSequenceType.

SendAttention: PROCEDURE [sH: Handle, byte: Byte] = INLINE {sH.sendAttention[sH, byte]};
    -- Send an attention down the stream.

WaitForAttention: PROCEDURE [sH: Handle] RETURNS [Byte] = INLINE {RETURN[sH.waitAttention[sH]]};
    -- Wait for an attention to arrive.

Delete: PROCEDURE [sH: Handle] = INLINE {sH.delete[sH]};
    -- Delete the stream object.

        -- Signals and errors
EndOfStream: SIGNAL [nextIndex: CARDINAL];
LongBlock: SIGNAL [nextIndex: CARDINAL];
ShortBlock: ERROR;
SSTChange: SIGNAL [sst: SubSequenceType, nextIndex: CARDINAL];
TimeOut: SIGNAL [nextIndex: CARDINAL];

        -- Representation
Object: TYPE = RECORD [
    options: InputOptions,
    getByte: GetByteProcedure,
    putByte: PutByteProcedure,
```

```
    getWord: GetWordProcedure,
    putWord: PutWordProcedure,
    get: GetProcedure,
    put: PutProcedure,
    setSST: SetSSTProcedure,
    sendAttention: SendAttentionProcedure,
    waitAttention: WaitAttentionProcedure,
    delete: DeleteProcedure];
GetByteProcedure: TYPE = PROCEDURE [sH: Handle] RETURNS [byte: Byte];
PutByteProcedure: TYPE = PROCEDURE [sH: Handle, byte: Byte];
GetWordProcedure: TYPE = PROCEDURE [sH: Handle] RETURNS [word: Word];
PutWordProcedure: TYPE = PROCEDURE [sH: Handle, word: Word];
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];

        -- The following are default values for stream object records
defaultObject: READONLY Object;
    -- = [
        -- options: defaultInputOptions, (see above)
        -- getByte: DefaultGetByte, (requires sH.get defined)
        -- putByte: DefaultPutByte, (requires sH.put defined)
        -- getWord: DefaultGetWord, (requires one of sH.getByte or sH.get defined)
        -- putWord: DefaultPutWord, (requires one of sH.putByte or sH.put defined)
        -- get: DefaultGet, (requires sH.getByte defined)
        -- put: DefaultPut, (requires sH.putByte defined)
        -- setSST: DefaultSetSST, (raises Runtime.UnboundProcedure)
        -- sendAttention: DefaultSendAttention, (raises Runtime.UnboundProcedure)
        -- waitAttention: DefaultWaitAttention, (raises Runtime.UnboundProcedure)
        -- delete: DefaultDelete];      (raises Runtime.UnboundProcedure)

END.
```

LOG

Time: March 17, 1978 11:39 AM By: Lauer Action: Created file
Time: April 6, 1978 3:24 PM By: Lauer Action: Updated to conform to revised Functional Specifications
Time: April 18, 1978 9:58 AM By: Lauer 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: 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: Lauer Action: Moved type Block from Stream to Environment; added GetChar and PutChar
Time: August 14, 1978 2:03 PM By: Lauer 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 respectively
Time: February 1, 1980 7:16 PM By: McJones Action: Added object procedures for get/put byte/word; new defaults
Time: August 13, 1980 5:07 PM By: McJones Action: Deleted defaults within definition of Object

```
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

```
-- SubSys.mesa (Last editied by Johnsson,  September 4, 1980  4:36 PM)

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

SubSys: DEFINITIONS =

  BEGIN

  Handle: TYPE = RECORD [UNSPECIFIED];
  nullHandle: Handle = [NIL];

  Load: PROCEDURE [
    file: File.Capability, offset: File.PageCount, codeLinks: BOOLEAN ← FALSE]
    RETURNS [Handle];
  Run: PROCEDURE [Handle];
  UnLoad: PROCEDURE [Handle];

  END....
```

```
DIRECTORY
    DiskChannel USING [Address, CompletionHandle, Handle],
    Environment USING [PageNumber],
    File USING [PageCount, PageNumber],
    FileInternal USING [Descriptor, Operation],
    Volume USING [ID],
    PhysicalVolumeFormat USING [PageNumber, SubVolumeDesc],
    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: PhysicalVolumeFormat.PageNumber,
    nPages: PageCount,
    channel: DiskChannel.Handle];

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

GetNext: PROC [inSubVolume: Handle] RETURNS [outSubVolume: Handle];

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

OnLine: PROC [subVolume: PhysicalVolumeFormat.SubVolumeDesc, channel: DiskChannel.Handle];

OffLine: PROC [vID: Volume.ID, channel: DiskChannel.Handle];

StartIO: PROC [io: IOptr];

IOptr: TYPE = POINTER TO IO;

IO: TYPE = RECORD[ -- argument record to StartIO; lifetime = duration of call only
    op: FileInternal.Operation,
    subVolume: Handle,
    subVolumePage: PageNumber,
    memPage: Environment.PageNumber,
    filePtr: --FileInternal.FilePtr-- LONG POINTER TO READONLY FileInternal.Descriptor,
    filePage: File.PageNumber,
    pageCount: File.PageCount,
    fixedMemPage: BOOLEAN ← FALSE,
    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 Disk.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.
Time: September 19, 1979 8:36 AM By: Forrest Action: Added GetNext; changed Offline to take disk channel.
Time: September 21, 1979 10:36 AM By: Forrest Action: Added SetSubvolumeMarkerID and GetSubvolumeMarkerID.
Time: September 22, 1979 2:29 PM By: Forrest Action: Took out the Subvolume marker stuff.
Time: November 26, 1979 3:00 PM By: Gobbel Action: Added count argument to StartIO.
Time: December 13, 1979 12:00 PM By: Redell Action: Changed StartIO to take arguments in a record, including new fixedMemoryPage.
Time: January 31, 1980 7:54 PM By: Gobbel Action: Changed pageCount field of IO record to be LONG (File.PageCount).
Time: May 20, 1980 2:54 PM By: Luniewski Action: PhysicalVolume = > PhysicalVolumeFormat.
Time: October 11, 1980 8:46 PM By: Forrest Action: Change IO's filePtr field to be pointer to Readonly Descriptor.

```
--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
    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

```
SwapperPrograms: DEFINITIONS =
BEGIN

-- CachedRegionImplA is in StoragePrograms:

CachedRegionImplB: PROGRAM [pMapLogDesc: LONG POINTER]; -- for patch table

CachedSpaceImpl: PROGRAM;

-- MStoreImpl is in StoragePrograms.

PageFaultImpl: PROGRAM;

-- ResidentMemoryImpl is in StoragePrograms.

-- SwapBufferImpl:
InitializeSwapBuffer: PROCEDURE; -- allocates swap buffer, describes its space to Swapper.

SwapperExceptionImpl: PROGRAM;

SwapTaskImpl: PROGRAM;

-- SimpleSpaceImpl is in StoragePrograms.

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*
Time: August 21, 1979 6:10 PM	*By: Knutsen*	*Action: Deleted obsolete parameter of* **SimpleSpaceImpl, MStoreImpl.**
Time: September 19, 1979 2:41 PM	*By: Knutsen*	*Action: Split CachedRegionImpl into CachedRegionImplA, B.*
Time: January 31, 1980 2:17 PM	*By: Knutsen*	*Action: Added intervalRegionCache parameter.*
Time: April 15, 1980 10:13 AM	*By: Knutsen*	*Action: SwapBufferImpl replaced by InitializeSwapBuffer. CachedRegionImplA, MStoreImpl, ResidentMemoryImpl, SimpleSpaceImpl moved to StoragePrograms.*

```
DIRECTORY
    CachedRegion: FROM "CachedRegion" USING [Desc],
    PageMap: FROM "PageMap" USING [Flags],
    VM: FROM "VM" USING [Interval, PageCount, PageNumber];

SwapTask: DEFINITIONS =

BEGIN

State: TYPE = RECORD [
    next: PRIVATE PState,                   -- next in swapTaskList.
    swapInterval: VM.Interval,              -- the interval to be swapped.
    flags: PageMap.Flags,                   -- of region when swap is finished.
    bufferPage: VM.PageNumber,              -- start of i/o buffer.
    countRemaining: VM.PageCount,           -- number of pages not yet transferred.
    region: CachedRegion.Desc];             -- desc to be checked-in upon completion.

PState: PRIVATE TYPE = POINTER TO State;

Fork: PROCEDURE [swapInterval: VM.Interval, flags: PageMap.Flags, bufferPage: VM.PageNumber, swapCount: VM.PageCount, region:
                CachedRegion.Desc]
            RETURNS [forked: BOOLEAN];
    -- Stores a State describing the swapping to be done on a swap unit of a region.  If the operation is vacuous, no State is stored and
            forked = FALSE is returned.

Advance: PROCEDURE [interval: VM.Interval] RETURNS [state: State];
    -- Returns the State which contains interval.page.  If State.countRemaining = 0, the State is also deleted.

END.
```

LOG

Time: During 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.*
Time: September 20, 1979 9:45 AM	*By: Knutsen*	*Action: Added* swapInterval. *Added documentation.*
Time: November 26, 1979 12:17 PM	*By: Gobbel*	*Action: Added count to Advance.*

```
DIRECTORY
    MiscAlpha USING [aRCLK],
    Mopcodes USING [zMISC];

System: DEFINITIONS =

BEGIN

    -- Universal identifiers
UniversalID: TYPE [5];
nullIDRep: RECORD [a, b, c, d, e: WORD] = [0, 0, 0, 0, 0]; -- never returned by GetUniversalID
nullID: UniversalID = LOOPHOLE[nullIDRep];
GetUniversalID: PROCEDURE RETURNS [uid: UniversalID];
FileID: TYPE = RECORD [UniversalID]; -- a useful special case of UniversalID
VolumeID: TYPE = RECORD [UniversalID]; -- another useful special case of UniversalID
PhysicalVolumeID: TYPE = RECORD [UniversalID]; -- yet another useful special case of UniversalID

    -- Network addresses
NetworkAddress: TYPE [6];
nullNetworkAddress: READONLY NetworkAddress;

    -- Timekeeping facilities
GreenwichMeanTime: TYPE = RECORD [LONG CARDINAL];
    -- A greenwich mean time t represents the time which is t-gmtEpoch seconds after midnight, 1 January 1968, the time chosen as
            the epoch or beginning of the Pilot time standard. Within the range in which they overlap, the Alto and Pilot time
            standards assign identical bit patterns, but the Pilot standard runs an additional 67 years before overflowing.
    -- Greenwich mean times should be compared directly only for equality; to find which of two gmt's comes first, apply
            SecondsSinceEpoch to each and compare the result. If t2 is a gmt known to occur after t1, then t2-t1 is the seconds
            between t1 and t2. If t is a gmt, then System.GreenwichMeanTime[t + 60] is the gmt one minute after t.
gmtEpoch: GreenwichMeanTime = [2114294400]; -- = (67 years * 365 days + 16 leap days) * 24 hours * 60 minutes * 60 seconds
GetGreenwichMeanTime: PROCEDURE RETURNS [gmt: GreenwichMeanTime];
SecondsSinceEpoch: PROCEDURE [gmt: GreenwichMeanTime] RETURNS [LONG CARDINAL] =
    INLINE BEGIN RETURN[gmt-gmtEpoch] END;
AdjustGreenwichMeanTime: PROCEDURE [gmt: GreenwichMeanTime, delta: LONG INTEGER]
    RETURNS [GreenwichMeanTime] = INLINE BEGIN RETURN[[gmt + delta]] END;

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

-- Old interval timer mechanism (clients should convert GetClockPulses)
TimerHandle: TYPE = RECORD [Pulses];
CreateIntervalTimer: PROCEDURE RETURNS [t: TimerHandle] = LOOPHOLE[GetClockPulses];
GetIntervalTime: PROCEDURE [t: TimerHandle] RETURNS [m: Microseconds];

    -- 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 NetworkAddress
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
Time: January 25, 1980 4:15 PM By: McJones Action: Redefined old interval timers in terms of Pulses

Time: February 9, 1980 12:41 PM By: McJones Action: Converted to new GreenwichMeanTime representation
Time: May 9, 1980 11:07 AM By: McJones Action: Added PhysicalVolumeID
Time: June 10, 1980 5:52 PM By: McJones Action: Added two more words to NetworkAddress and one more word to UniversalID; converted them both to exported types
Time: August 13, 1980 4:20 PM By: McJones Action: Added AdjustGreenwichMeanTime; converted to MiscAlpha

```
DIRECTORY
    SpecialSystem USING [nullProcessorID, ProcessorID];

SystemInternal: DEFINITIONS =

BEGIN

    -- Universal identifiers
    -- Implementation of the exported type in System
UniversalID: TYPE = MACHINE DEPENDENT RECORD [
    processor (0): SpecialSystem.ProcessorID,
    sequence (3): LONG CARDINAL];
    -- NOTE: The fields of a UniversalID should NOT be used as hints as to the location or age of the associated entity (e.g. file,
            volume).
nullID: UniversalID = [SpecialSystem.nullProcessorID, 0]; -- assert = LOOPHOLE[System.nullID]

    -- Unimplemented features
Unimplemented: SIGNAL;
    -- This signal can't be caught by client code since SystemInternal is not exported by Pilot.  However it can be resumed from the
            debugger when raised via SIGNAL (rather than ERROR) and this may possibly ease clients' debugging.

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 10:03 AM By: Lauer Action: Corrected bugs in OISProcessorSeries, UniversalID; changed TimerHandle; 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: August 10, 1979 5:07 PM By: Forrest Action: Added Unimplemented (formerly in Forgot)
Time: January 25, 1980 3:36 PM By: Dalal Action: Created dependencies on SpecialSystem
Time: January 27, 1980 7:00 PM By: Forrest Action: Reformatted; made records machine dependent
Time: June 10, 1980 5:55 PM By: McJones Action: 48-bit processor ids

```
-- file Table.Mesa
-- last modified by Satterthwaite, July 30, 1980  10:32 AM
-- Copyright  Xerox Corporation 1979, 1980

DIRECTORY
  Bases: TYPE USING [
    Base, BaseDescriptor, Finger, Index, IPointer, Limit, OrderedIndex,
    SizeDescriptor];

Table: DEFINITIONS =
  BEGIN

  Selector: TYPE = CARDINAL;

  Base: TYPE = Bases.Base;
  Finger: TYPE = Bases.Finger;
  Limit: CARDINAL = Bases.Limit;
  Index: TYPE = Bases.Index;
  OrderedIndex: TYPE = Bases.OrderedIndex;

  IPointer: TYPE = Bases.IPointer;

  BaseDescriptor: TYPE = Bases.BaseDescriptor;
  SizeDescriptor: TYPE = Bases.SizeDescriptor;

  -- allocation from the tables as stacks

  Allocate: PROC [table: Table.Selector, size: CARDINAL]
    RETURNS [Table.OrderedIndex];
  Bounds: PROC [table: Table.Selector] RETURNS [base: Table.Base, size: CARDINAL];
  Top: PROC [table: Table.Selector] RETURNS [Table.OrderedIndex] = INLINE {
    RETURN[FIRST[Table.OrderedIndex] + Bounds[table].size]};

  Trim: PROC [table: Table.Selector, size: CARDINAL];

  -- allocation from free list (first table only)

  chunkType: Table.Selector = FIRST[Table.Selector];

  GetChunk: PROC [size: CARDINAL] RETURNS [Table.Index];
  FreeChunk: PROC [index: Table.Index, size: CARDINAL];

  -- inquiries

  WordsUsed: PROC RETURNS [CARDINAL];
  WordsFree: PROC RETURNS [CARDINAL];

  -- notification of repacking

  Notifier: TYPE = PROC [base: Table.BaseDescriptor];

  AddNotify: PROC [proc: Table.Notifier];
  DropNotify: PROC [proc: Table.Notifier];

  -- initialization and termination

  Region: TYPE = RECORD [origin: Table.Base, size: CARDINAL];

  Create: PROC [region: Table.Region, weights: DESCRIPTOR FOR ARRAY OF CARDINAL];
  Destroy: PROC;

  Overflow: SIGNAL RETURNS [Region];
  Failure: ERROR [table: Table.Selector];

  END.
```

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

Teledebug: DEFINITIONS =
BEGIN

Debug: PROCEDURE [
 scratchPage: Environment.PageNumber,
 dFirst64KStorage: LONG DESCRIPTOR FOR ARRAY OF WORD];

END.
```

LOG

Time: March 21, 1980 6:34 PM By: Forrest Action: Create file

```
    DIRECTORY
        Device: FROM "Device" USING [Type],
        PupTypes: FROM "PupTypes" USING [maxDataWordsPerGatewayPup, PupType, PupSocketID];

TeledebugProtocol: DEFINITIONS =
BEGIN

CorePage, DiskPage: TYPE = LONG CARDINAL;
Flag: TYPE = WORD;
Label: TYPE = ARRAY [0..10) OF UNSPECIFIED;
Page: TYPE = ARRAY [0..256) OF WORD;

acknowledgement: PupTypes.PupType = LOOPHOLE[204B];


coreStoreRequest: PupTypes.PupType = LOOPHOLE[300B];
 CoreStoreRequest: TYPE = MACHINE DEPENDENT RECORD [
  page: CorePage, flags: Flag, data: Page];
 CoreStoreAcknowledgement: TYPE = MACHINE DEPENDENT RECORD [
  page: CorePage, flags: Flag]; -- vacant flag if failure (not mapped)

coreFetchRequest: PupTypes.PupType = LOOPHOLE[301B];
 CoreFetchRequest: TYPE = MACHINE DEPENDENT RECORD [page: CorePage];
 -- reply with vacant flag if failure (not mapped)
 CoreFetchAcknowledgement: TYPE = MACHINE DEPENDENT RECORD [
  page: CorePage, flags: Flag, data: Page];

-- server is free to ignore device and ordinal
diskAddressSetRequest: PupTypes.PupType = LOOPHOLE[302B];
 DiskAddressSetRequest: TYPE = MACHINE DEPENDENT RECORD [
  page: DiskPage, device: Device.Type, deviceOrdinal: CARDINAL];
 -- failure not possible
 DiskAddressSetAcknowledgement: TYPE = MACHINE DEPENDENT RECORD [
  page: DiskPage, device: Device.Type, deviceOrdinal: CARDINAL];

diskStoreRequest: PupTypes.PupType = LOOPHOLE[303B];
 DiskStoreRequest: TYPE = MACHINE DEPENDENT RECORD [
  label: Label, data: Page];
 -- reply with noLabel indicates failure.  Garbage can be usefull for debugging
 DiskStoreAcknowledgement: TYPE = MACHINE DEPENDENT RECORD [
  label: Label, garbage: ARRAY [0..0) OF UNSPECIFIED];

diskFetchRequest: PupTypes.PupType = LOOPHOLE[304B];
 -- DiskFetchRequest: TYPE = RECORD [no contents];
 -- If a disk fetch request comes by with contents size = diskAddressSet request, then set the disk address BEFORE doing the Fetch
 DiskFetchAcknowledgement: TYPE = MACHINE DEPENDENT RECORD [
  label: Label, data: Page];

go: PupTypes.PupType = LOOPHOLE[202B];
 -- GoRequest, GoAcknowledgement: TYPE = RECORD [no contents];

goReply: PupTypes.PupType = LOOPHOLE[203B];
 -- GoAcknowledgementRequest: TYPE = RECORD [no contents];
 -- no reply frome server

noLabel: Label = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1];
vacantFlag: Flag = 60000B; -- writeProtected + dirty = vacant
teleSwatSocket: PupTypes.PupSocketID = [0, 60B];

-- Checks
MaxSize: TYPE = CARDINAL [0..PupTypes.maxDataWordsPerGatewayPup];
coreStoreRequestSize: MaxSize = SIZE[CoreStoreRequest];
coreStoreAcknowledgementSize: MaxSize = SIZE[CoreStoreAcknowledgement];
coreFetchRequestSize: MaxSize = SIZE[CoreFetchRequest];
coreFetchAcknowledgementSize: MaxSize = SIZE[CoreFetchAcknowledgement];
diskAddressSetRequestSize: MaxSize = SIZE[DiskAddressSetRequest];
diskAddressSetAcknowledgementSize: MaxSize = SIZE[DiskAddressSetAcknowledgement];
diskStoreRequestSize: MaxSize = SIZE[DiskStoreRequest];
diskStoreAcknowledgementSize: MaxSize = SIZE[DiskStoreAcknowledgement];
diskFetchAcknowledgementSize: MaxSize = SIZE[DiskFetchAcknowledgement];

END.....
```

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

TemporaryBooting: DEFINITIONS =

BEGIN

tBootFile: READONLY File.Type;
    -- Only files of this type may be made bootable.

MakeBootable: PROCEDURE [file: File.Capability, firstPage: File.PageNumber ← 0];
    -- Install the chaining links in the labels of the file pages starting with the specified first page and for the length of the contained
            Pilot boot file.
    -- MakeBootable must be performed after contents are written and before file is passed to BootFrom.
    -- InvalidParameters is raised if file is of wrong type or doesn't contain Pilot boot file starting at firstPage.

MakeUnbootable: PROCEDURE [file: File.Capability, firstPage: File.PageNumber ← 0];
    -- Remove the chaining links.  May speed subsequent bulk access to the file.

InstallVolumeBootFile: PROCEDURE [file: File.Capability, firstPage: File.PageNumber ← 0];
    -- Set up the file as the one gaining control when the containing (logical) volume is booted.

InstallPhysicalVolumeBootFile: PROCEDURE [file: File.Capability, firstPage: File.PageNumber ← 0];
    -- Set up the (logical) volume as the one gaining control when the containing physical volume is booted.

--Switches: TYPE = PACKED ARRAY Switch OF BOOLEAN; (in Mesa 6.0)

Switch: TYPE = {zero, one, two, three, four, five, six, seven, eight, nine, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z};

Switches: TYPE = RECORD [zero, one, two, three, four, five, six, seven, eight, nine, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w,
        x, y, z: UpDown ← up];

UpDown: TYPE = {up, down};

defaultSwitches: Switches = [];

BootFromFile: PROCEDURE [file: File.Capability, firstPage: File.PageNumber ← 0, switches: Switches ← defaultSwitches];
    -- Restart the system from the specified boot file.  Improper arguments will likely result in an MP code and crash.

BootFromVolume: PROCEDURE [volume: Volume.ID, switches: Switches ← defaultSwitches];
    -- Restart the system from the Pilot boot file installed on the specified volume.

BootFromPhysicalVolume: PROCEDURE [volume: Volume.ID, switches: Switches ← defaultSwitches];
    -- Restart the system from the Pilot boot file installed on the physical volume containing the specified (logical) volume.

BootButton: PROCEDURE [switches: Switches ← defaultSwitches];
    -- Restart the system as if its boot button had been pressed (may ignore switches).

InvalidParameters: ERROR;

END.
```

LOG

Time: September 14, 1979 2:45 PM By: McJones Action: Create file
*Time: October 3, 1979 6:17 PM By: McJones Action: Add InstallAsBoot**
Time: January 25, 1980 10:58 AM By: McJones Action: InstallAsBootFile = > InstallVolumeBootFile;
*InstallAsBootVolume[vol] = > InstallPhysicalBootVolume[file]; add BootFromPhysicalVolume, Boot; add Switches to Boot**
Time: April 17, 1980 10:34 AM By: McJones Action: Default for firstPage to BootFrmFile missing

```
TemporarySetGMT: DEFINITIONS =

BEGIN

SetGMT: PROCEDURE;
    -- Perform OISProcessorFace.SetGreenwichMeanTime with the correct time.
    -- Implementation may use EthernetFace directly; only called early in Pilot initialization and coming back from Snapshot.OutLoad
        (the second time), with interrupts disabled.
    -- Note: This is an interim measure necessary until a clock implementation which can run across boots exists.

END.
```

LOG

Time: April 16, 1980 12:14 PM By: McJones Action: Create file

```
TemporaryTransaction: DEFINITIONS =
BEGIN

DisableTransactions: PROCEDURE;

END.
```

LOG
Time: July 16, 1980 9:03 PM By: Gobbel Action: Created file.
Time: July 30, 1980 4:34 PM By: Gobbel Action: Renamed from SpecialTransaction to TemporaryTransaction.

```
DIRECTORY
    MiscAlpha: FROM "MiscAlpha" USING [aTEXTBLT],
    Mopcodes: FROM "Mopcodes" USING [zMISC];

TextBlt: DEFINITIONS =
BEGIN

Function:   TYPE = {display, format, resolve};

TextBltArg: TYPE = MACHINE DEPENDENT RECORD [
    reserved:   [0..37777B] ← 0,
    function:   Function,               -- display, format or resolve
    last:       CARDINAL,               -- index of last character to process
    text:       LONG POINTER TO PACKED ARRAY CARDINAL OF CHARACTER,
    font:       FontHandle,             -- Long Pointer to font information
    dst:        LONG POINTER,           -- destination bitmap (display only)
    dstBpl:     CARDINAL,               -- Bits per line (display only)
    margin:     CARDINAL,               -- mica value of right margin (format only)
    space:      INTEGER,                -- width adjustment to pad characters (display, resolve)
    coord:      POINTER TO ARRAY CARDINAL[0..0) OF CARDINAL -- widths array for resolve
    ];

TextBltArgAlignment: CARDINAL = 16;
TextBltArgSpace: TYPE = ARRAY [1..SIZE[TextBltArg] + TextBltArgAlignment) OF UNSPECIFIED;
AlignedTextBltArg: PROCEDURE [ip: POINTER TO TextBltArgSpace] RETURNS [p: POINTER TO TextBltArg] =
    INLINE BEGIN
    align: TYPE = MACHINE DEPENDENT RECORD[
        s: [0..LAST[WORD]/TextBltArgAlignment),
        z: [0..TextBltArgAlignment)];
    p ← LOOPHOLE[ip + TextBltArgAlignment - 1];
    LOOPHOLE[p, align].z ← 0;
    END;

FontHandle: TYPE = LONG POINTER TO Font;

Font:   TYPE;


Result: TYPE = {normal, margin, stop};


TextBlt: PROCEDURE
    [index: CARDINAL, bitPos: CARDINAL, micaPos: CARDINAL, count: INTEGER, ptr: POINTER TO TextBltArg]
    RETURNS [newIndex: CARDINAL, newBitPos: CARDINAL, newMicaPos: CARDINAL, newCount: INTEGER, result:
    Result] = MACHINE CODE BEGIN Mopcodes.zMISC, MiscAlpha.aTEXTBLT; END;



END.    -- of TextBlt
```

LOG
3-24-80 -- Guyton -- Created
July 30, 1980 10:56 AM -- Jim Frandeen -- Add FontRecord.height
August 29, 1980 3:29 PM -- Jim Frandeen -- Remove Font, rearrange parameters.
October 7, 1980 9:12 AM -- Jim Frandeen -- Add AlignedTextBltArg. Fix TextBlt Inline definition.

```
-- TimeStamp.Mesa  Edited by Sandman on August 12, 1980  10:38 AM
-- Copyright  Xerox Corporation 1980

TimeStamp: DEFINITIONS =
  BEGIN

  Stamp: TYPE = RECORD [net, host: [0..377B], time: LONG CARDINAL];

  Null: Stamp = Stamp[net: 0, host: 0, time: 0];

  END.
```

```
Transaction: DEFINITIONS =
BEGIN
Handle: TYPE [1];
nullHandle: READONLY Handle;
InvalidHandle: ERROR;
Begin: PROCEDURE RETURNS [Handle];
Commit: PROCEDURE [transaction: Handle];
Abort: PROCEDURE [handle: Handle];
END.
```

LOG

Time: March 31, 1980 4:30 PM By: Gobbel Action: Created file
Time: June 12, 1980 11:19 AM By: Gobbel Action: Added InvalidHandle, made Handle be an exported type
Time: June 16, 1980 2:21 PM By: McJones Action: Made nullHandle be READONLY

-- This interface defines types, constants, and global variables which are private to the Transaction implementation software, and supplies the procedural interface between TransactionStateImpl and TransactionLogImpl.

```
DIRECTORY
    Environment USING [PageCount, PageNumber, wordsPerPage],
    Inline USING [LowHalf],
    File USING [delete, grow, ID, PageCount, PageNumber, Permissions, read, shrink, write],
    MiscAlpha USING [aCHKSUM],
    Mopcodes USING [zMISC],
    SimpleSpace USING [Handle, WindowOrigin],
    Space USING [PageCount],
    SpecialTransaction USING [OptionalCrash],
    TransactionState USING [LogEntry, LogEntryPtr, FileOps, SpaceNodePtr];

TransactionInternal: DEFINITIONS
    IMPORTS Inline =

BEGIN

FilePageNumber: TYPE = Environment.PageNumber; -- = LowHalf[File.PageNumber]. We will never use more than 2**16 pages in a log file, and
                                                   would rather not store the extra bits.
FilePageCount: TYPE = Environment.PageCount; -- = LowHalf[File.PageCount]. We will never use more than 2**16 pages in a log file, and would
                                                  rather not store the extra bits.

permissions: File.Permissions = File.read + File.write + File.grow + File.shrink + File.delete; -- to save space in the State Table, we only save the
                                File.ID of a Log File. Since we created all of these files ourselves, we can reasonably supply any permissions we want.


TxIndex: TYPE = CARDINAL [0..77B]; -- (six bits) defines the maximum number of simultaneously active transactions.
nullTxIndex: TxIndex = LAST[TxIndex]; -- Must be LAST[]! This entry in the State Table is never used for a transaction. Instead, it holds checking
    information which is used to verify that the entire State Table has been written on the disk, as follows: a serial number is written in every entry in the
    State Table, before it is written on the disk; When the State Table is read from the disk during Crash REcovery, if the serial number of the last entry
    (i.e. state[nullTxIndex] ) matches that of all previous entries, it indicates that all of the previous entries were sucessfully written on the disk.
ValidTxIndex: TYPE = TxIndex[ FIRST[TxIndex] .. nullTxIndex - 1 ];

TxEdition: TYPE = CARDINAL [0..1777B]; -- (ten bits) identifies current edition of a particular transaction handle.
invalidTxEdition: TxEdition = FIRST[TxEdition]; -- used for nullHandle. Also, zero is the most likely value to be supplied by malfunctioning
                                                   hardware/software.
ValidTxEdition: TYPE = TxEdition[ invalidTxEdition + 1 .. LAST[TxEdition] ];
-- Note: TxIndex and TxEdition must total 16 bits or less, so that a Transaction.Handle will fit in one word.


Handle: TYPE = RECORD [txEdition: TxEdition, index: TxIndex]; -- the concrete type for a Transaction.Handle.
```

```
-- Transaction State Table  (describes state of all current transactions)

StateFormatVersion: TYPE = RECORD [ CARDINAL ]; -- could be less than 16 bits.
stateFormatVersion: StateFormatVersion = [1]; -- change this every time the format of the State Table / State Data changes!
                    (in a non-backward compatible manner) Changing this value invalidates all existing Transaction State files.
                    Clients must be warned to run their old system past crash recovery and shut down cleanly before
                    installing the new system.

StateEdition: TYPE = CARDINAL; -- (could be less than a full word)

State: TYPE = ARRAY TxIndex OF TxDescriptor; -- the State Table. Records the state of all current transactions, active or inactive.

-- A TxDescriptor records the state of one transaction. If you change its format (in a non-backward compatible manner), change
   stateFormatVersion! (unless you want to test crash recovery of the State Table)
TxDescriptor: TYPE = MACHINE DEPENDENT RECORD [ -- MACH DEP so that format is stable over compiler versions.
    stateEdition (0: 0..15): StateEdition, -- a copy of currentStateEdition. First field of TxDesc so that a correct value indicates that all of the previous
                                              TxDesc, and at least the first word of this TxDesc, got written on the disk.
    seal (1: 0..15): StateFormatVersion ← stateFormatVersion, -- identifies version of format of TxDesc and State.
    txEdition (8: 0..9): TxEdition, -- identifies the current edition of this transaction. Not relevant to crash recovery.
    fileInUse (8: 10..10): BOOLEAN, -- tells whether this transaction has actually begun using logFile yet. Not relevant to crash recovery.
    logFile (2: 0..79): File.ID, -- the log file available for or being used by this transaction.
    logFileSize (7: 0..15): FilePageCount, -- current size of logFile. Not relevant to crash recovery.
    fill (8: 11..12): [0..1] ← 0,
    vp (8: 13..79): SELECT status (8: 13..13): Status FROM
        active => [ -- becomes active at Begin[], becomes free at completion of Commit, Abort, or Crash Recovery.
            noMoreOperations (8: 14..14): BOOLEAN, -- no more logging operations allowed. Not relevant to crash recovery.
            committed (8: 15..15): BOOLEAN, -- all changes to client's files have been made on the disk.
            spaceList (9: 0..31): TransactionState.SpaceNodePtr, -- head of list of Spaces involved in transaction. Not relevant to crash recovery.
            pageFree (11: 0..15): FilePageNumber, -- first page of logFile not used yet. Not relevant to crash recovery.
            countPrevLogEntry (12: 0..15): FilePageCount ], -- length of last log entry made (header + any data). Not relevant to crash
                                                               recovery.
        free => [ fill2 (8: 14..15): [0..1] ← 0 ], -- (fill out to a word boundary)
        ENDCASE ];
TxDescPtr: TYPE = LONG POINTER TO TxDescriptor;
ActiveTxDescPtr: TYPE = LONG POINTER TO active TxDescriptor;

Status: TYPE = MACHINE DEPENDENT { active (0), free (1) }; -- status of a transaction Desc.

StateData: TYPE = MACHINE DEPENDENT RECORD [ -- layout of data written to the disk.
    state (0): State, -- must be first field in record! (see InitializeStateB)
    checksumData (832): ChecksumData ]; -- checksum of all previous words in StateData.

stateCount: Space.PageCount = (SIZE[StateData] + Environment.wordsPerPage-1) / Environment.wordsPerPage;


-- Transaction Log File  (contains "undo" data for a single transaction)

LogFileFormatVersion: TYPE = RECORD[ CARDINAL ]; -- could be less than 16 bits.
logFileFormatVersion: LogFileFormatVersion = [123456B]; -- change this every time the format of the Log File changes (in a
                      non-backward compatible manner)! Changing this value invalidates all existing Transaction Log Files.
                      Clients must be warned to run their old system past crash recovery and shut down cleanly before
                      installing the new system.

LogFileEntry: TYPE = MACHINE DEPENDENT RECORD [
    seal (0: 0..15): LogFileFormatVersion ← logFileFormatVersion, -- for Crash Recovery.
    transactionHandle (1: 0..15): Handle, -- for Crash Recovery, just to help weed out obsolete log file contents.
    countPrevEntry (2: 0..15): FilePageCount ← 0, -- size of previous LogFileEntry (header + any data).
    op (3: 0..223): TransactionState.LogEntry, -- describes "undo" action for Abort and Crash Recovery.
    spare (17: 0..31): LONG CARDINAL ← 0, -- (for future expansion)
    transactionHandle2 (19: 0..15): Handle ]; -- a copy of transactionHandle. For Crash Recovery.
LogFileEntryPtr: TYPE = LONG POINTER TO LogFileEntry;

LogFileEntryData: TYPE = MACHINE DEPENDENT RECORD [ -- layout of data written to the disk.
    logFileEntry (0): LogFileEntry,
    checksumData (20): ChecksumData ]; -- checksum of all previous words in this LogFileEntry.

countLogEntry: Environment.PageCount = 1; -- at present, each log entry is written into a separate whole page. ( + any pages containing data)

standardLogFileCount: FilePageCount = 10; -- size of log files when not in use.
```

```
-----------------------------------------------------
-- Procedures and variables
-----------------------------------------------------

-- Inlines:

Checksum: PROCEDURE [cksumOfOtherData: CARDINAL, nWords: CARDINAL, p: LONG POINTER] RETURNS [cksum: ChecksumData] =
    MACHINE CODE BEGIN Mopcodes.zMISC, MiscAlpha.aCHKSUM END; -- should be in Inline!
ChecksumData: TYPE = CARDINAL;

ShortenPageCount: PROCEDURE [File.PageCount] RETURNS [FilePageCount] = Inline.LowHalf; -- (there's a LOOPHOLE here)

ShortenPageNumber: PROCEDURE [File.PageNumber] RETURNS [FilePageNumber] = Inline.LowHalf; -- (there's a LOOPHOLE here)


-- Exported by TransactionStateImpl:

disabled: BOOLEAN; -- TRUE forces all (public) transaction operations to no-op.

state: LONG POINTER TO State; -- the State Table. (LOOPHOLE[state] is also a pointer to stateData)
stateSpace: SimpleSpace.Handle; -- the working copy of the state in VM.

state1Window, state2Window: SimpleSpace.WindowOrigin; -- the two representatives for the stable storage containing the state table.

MaybeCrash: SpecialTransaction.OptionalCrash --[unimportance: SpecialTransaction.Unimportance]--;
    -- For testing only, calls client-supplied procedure to crash Pilot at critical points.

UpdateStateFile: --INTERNAL-- PROCEDURE [];
    -- Writes two copies of the State Table onto the disk.


-- Exported by TransactionLogImpl:

InitializeLogsA: --INTERNAL-- PROCEDURE [];
    -- Creates simple spaces. Must be called early during Pilot initialization.

InitializeLogsB: --INTERNAL-- PROCEDURE [];

LogInternal: --INTERNAL-- PROCEDURE [txi: ValidTxIndex, op: TransactionState.LogEntryPtr, fileOps: TransactionState.FileOps];
    -- Internal version of TransactionState.Log (q.v.).
    -- May raise Volume.InsufficientSpace.

END.
```

LOG

June 10, 1980 9:37 PM	*Gobbel*	*Created file.*
July 19, 1980 9:52 AM	*McJones*	*Adjusted field positions in LogEntry for 80-bit universal ids, made first64K READONLY.*
July 22, 1980 12:05 PM	*Fay*	*Added Handle to get around exported type clash between TransactionImpl and TransactionStateImpl.*
July 22, 1980 3:28 PM	*Gobbel*	*Make it work.*
September 2, 1980 3:09 PM	*Knutsen*	*Added Log File defs, ValidTxIndex, LogInternal, parameters to LogsA/B. Removed ProcessLog.*
September 11, 1980 5:17 PM	*Knutsen*	*Restore* MACHINE DEPENDENT*ness.*
September 12, 1980 1:47 PM	*Gobbel*	*Started stateFormatVersion over at 1, to be incremented for future changes.*
September 25, 1980 12:52 PM	*Gobbel*	*Added MaybeCrash.*

```
-- This interface supplies operations for modifying the transaction data base for use by higher-level transaction-related software
   (FileMgr, TransactionMgr, VMMgr).

DIRECTORY
    File USING [Capability, ID, PageCount, Type],
    Space USING [Handle, WindowOrigin],
    System USING [VolumeID],
    Transaction USING [Handle];

TransactionState: DEFINITIONS =
BEGIN

-- The Log for a transaction is an "undo log". Thus, a LogEntry records actions to be done if the transaction aborts, or if by crash
   recovery for uncommitted transactions. For example, if a file was made permanent in a transaction, a makeTemporary entry would be
   recorded the transaction's Log.
-- Changing the format of a LogEntry (in a non-backward compatible manner) invalidates all existing Transaction Log Files. If
   you do change the format, you must change TransactionInternal.stateFormatVersion.
LogOp: TYPE = MACHINE DEPENDENT
        { create (0), delete (1), makeMutable (2), makeTemporary (3), move (4), setContents (5), shrink (6) };
LogEntry: TYPE = MACHINE DEPENDENT RECORD [ -- MACHINE DEPENDENT so that format endures over compiler changes.
    logEntry (0): SELECT operation (0: 0..2): LogOp FROM
        create => [
            fill (0: 3..15): [0..17777B] ← 0,
            id (1: 0..79): File.ID,
            volume (6: 0..79): System.VolumeID,
            size (11: 0..31): File.PageCount,
            type (13: 0..15): File.Type ],
        delete, makeMutable, makeTemporary => [
            fill (0: 3..15): [0..17777B] ← 0,
            file (1: 0..95): File.Capability ],
        move => [
            fill (0: 3..15): [0..17777B] ← 0,
            file (1: 0..95): File.Capability,
            volume (7: 0..79): System.VolumeID ],
        setContents => [
            makePermanent (0: 3..3), makeImmutable (0: 4..4): BOOLEAN ← FALSE,
            fill (0: 5..15): [0..3777B] ← 0,
            window (1: 0..127): Space.WindowOrigin,
            size (9: 0..31): File.PageCount], -- if necessary, grow file so that file.size >= window.base + size.
        shrink => [
            fill (0: 3..15): [0..17777B] ← 0,
            file (1: 0..95): File.Capability,
            size (7: 0..31): File.PageCount ] -- shrink file to have size pages.
        ENDCASE ];
LogEntryPtr: TYPE = LONG POINTER TO LogEntry;

SpaceNode: TYPE = RECORD [ -- list element for a Space that is part of a transaction.
    handle: Space.Handle,
    window: Space.WindowOrigin, -- (scratch storage for Abort processing)
    nextSpace: SpaceNodePtr];
SpaceNodePtr: TYPE = LONG POINTER TO SpaceNode;

ReleaseTransaction: PROCEDURE [transaction: Transaction.Handle];
    -- Marks this transaction no longer in use (i.e. it has been Committed or Aborted).
    -- All updates to client files must have been done or undone on the disk before this routine is called!

RecordCommit: PROCEDURE [transaction: Transaction.Handle];
    -- Marks the transaction committed (and records the fact on the disk).
    -- All updates to client files must have been forced out to the disk before this routine is called!

AddToTransaction: PROCEDURE [transaction: Transaction.Handle, space: Space.Handle];
    -- Adds this space to the list of spaces involved in the transaction.
    -- May raise Transaction.InvalidHandle.
    -- The client must assure that the space is not already involved in the transaction!

WithdrawFromTransaction: PROCEDURE [transaction: Transaction.Handle, space: Space.Handle];
    -- Withdraws this space from the list of spaces involved in the transaction.
    -- May raise Transaction.InvalidHandle.
    -- The client must assure that the space is involved in the transaction!

InitializeStateA: PROCEDURE [];
    -- Creates SimpleSpaces. Must be called before VMMgr is initialized.
```

```
InitializeStateB: PROCEDURE [];
    -- Initializes databases.  Must be called after VMMgr is initialized.

Log: PROCEDURE [transaction: Transaction.Handle, op: LogEntryPtr, fileOps: FileOps];
    -- Records one "undo" operation in the transaction's Log File.
    -- May raise Transaction.InvalidHandle, Volume.InsufficientSpace.
    -- If a higher-level operation requires several LogEntrys, they must be logged in reverse of the order in which these "undo" actions
       must be done (if the transaction aborts).
    -- Note that the files operated on by fileOps must not be the file being logged.
    -- Since this procedure is called from within the FileImpl monitor, it may not be called from any system component at a level higher
       than the FileMgr.
FileOps: TYPE = RECORD [
        CreateFile: PROCEDURE [size: File.PageCount, type: File.Type] RETURNS [file: File.ID],
                -- Creates file of given size and type on the system volume.
                -- May raise Volume.InsufficientSpace.
        MakeFilePermanent: PROCEDURE [file: File.ID],
                -- Makes the file permanent.
                -- If file is unknown, you will wake up in the debugger.
        SetFileSize: PROCEDURE [file: File.ID, size: File.PageCount] ];
                -- Adjusts file to have size pages.
                -- May raise Volume.InsufficientSpace.  If file is unknown, you will wake up in the debugger.

NoMoreOperations: PROCEDURE [transaction: Transaction.Handle] RETURNS [spaceList: SpaceNodePtr];
    -- Checks out this transaction to Abort or Commit.  Prevents further client operations.
    -- spaceList is the head of a list of spaces involved in transaction.
    -- Raises Transaction.InvalidHandle if the transaction has already been checked out by some other process.

END.
```

LOG

June 10, 1980 9:37 PM	*Gobbel*	*Created file from TransactionInternal.*
August 15, 1980 1:17 PM	*Knutsen*	*Added comments describing clients.*
September 2, 1980 2:56 PM	*Knutsen*	*Renamed module from SpecialTransacton to TransactionState. Added* disabled, ReleaseTransaction, RecordCommit, InitializeStateA/B, NoMoreOperations, ProcessLog *(temorary!). Deleted* GrowLog. *Added* fileOps *to* Log.
September 8, 1980 5:58 PM	*Gobbel*	*Added* WithDrawFromTransaction.

```
-- Trap.Mesa  Edited by Forrest on September 2, 1980  9:03 AM

DIRECTORY
  Mopcodes,
  PrincOps USING [ControlLink];

Trap: DEFINITIONS = BEGIN

  ReadATP, ReadOTP: PROCEDURE RETURNS [PrincOps.ControlLink] =
    MACHINE CODE BEGIN Mopcodes.zLLB, 3B END;

  END...
```

```
-- Last edited: September 30, 1980 10:22 AM By: Mary Artibee

DIRECTORY
    TTYPortEnvironment: FROM "TTYPortEnvironment" USING [CharacterLength, LineSpeed, Parity, StopBits];

TTYPort: DEFINITIONS =

BEGIN

-- Procedures, alphabetically

CharsAvailable: PROCEDURE [channel: ChannelHandle] RETURNS [number: CARDINAL];
Create: PROCEDURE [lineNumber: CARDINAL] RETURNS [ChannelHandle];
Delete: PROCEDURE [channel: ChannelHandle];
Get: PROCEDURE [channel: ChannelHandle] RETURNS [data: CHARACTER, status: TransferStatus];
GetStatus: PROCEDURE [channel: ChannelHandle] RETURNS [stat: DeviceStatus];
Put: PROCEDURE [channel: ChannelHandle, data: CHARACTER] RETURNS [status: TransferStatus];
Quiesce: PROCEDURE [channel: ChannelHandle];
SetParameter: PROCEDURE [channel: ChannelHandle, parameter: Parameter];
StatusWait: PROCEDURE [channel: ChannelHandle, stat: DeviceStatus] RETURNS [newstat: DeviceStatus];

-- SIGNALs and ERRORs

ChannelAlreadyExists, ChannelQuiesced, InvalidLineNumber, NoTTYPortHardware: ERROR;

-- Type Definitions

ChannelHandle: TYPE = PRIVATE LONG POINTER TO UNSPECIFIED;

TransferStatus: TYPE = {success, parityError, asynchFramingError, dataLost, breakDetected, aborted, abortedByDelete};

DeviceStatus: TYPE = RECORD [
    aborted: BOOLEAN,
    breakDetected: BOOLEAN, --latched
    dataTerminalReady: BOOLEAN,
    readyToGet: BOOLEAN,
    readyToPut: BOOLEAN,
    requestToSend: BOOLEAN];

Parameter: TYPE = RECORD [SELECT parameter: * FROM
    breakDetectedClear => [breakDetectedClear: BOOLEAN],
    characterLength => [characterLength: CharacterLength],
    clearToSend => [clearToSend: BOOLEAN],
    dataSetReady => [dataSetReady: BOOLEAN],
    lineSpeed => [lineSpeed: LineSpeed],
    parity => [parity: Parity],
    stopBits => [stopBits: StopBits],
    ENDCASE];

CharacterLength: TYPE = TTYPortEnvironment.CharacterLength;
LineSpeed: TYPE = TTYPortEnvironment.LineSpeed;
Parity: TYPE = TTYPortEnvironment.Parity;
StopBits: TYPE = TTYPortEnvironment.StopBits;

END. -- TTYPort
```

LOG

Time: July 18, 1980 4:12 PM By: Mary Artibee Action: Use TTYPortFace.
Time: July 24, 1980 8:19 PM By: Mary Artibee Action: Changes to DeviceStatus, TransferStatus, Parameter. Removed ParameterType and LatchBitClearMask.
Time: July 28, 1980 1:23 PM By: Mary Artibee Action: Renamed TTYPort from Front; CheckDataFromPrinter replaced by CharsAvailable.
Time: July 28, 1980 3:40 PM By: Mary Artibee Action: ChannelHandle now LONG POINTER.
Time: August 5, 1980 9:34 AM By: Mary Artibee Action: Removed timeout on TransferStatus.
Time: August 28, 1980 10:17 AM By: Mary Artibee Action: Use TTYPortEnvironment.
Time: September 30, 1980 10:23 AM By: Mary Artibee Action: TransferStatus.deviceError now dataLost and breakDetected; removed DeviceStatus.dataLost, parameter clearDataLost.

-- Last edited: August 28, 1980 10:24 AM By: Mary Artibee

-- Defines basic, hopefully unchanging TYPEs required by BOTH the TTY Port channel (TTYPort) and device face (TTYPortFace), thus preventing interdependencies.

```
TTYPortEnvironment: DEFINITIONS =

BEGIN

-- Only Type Definitions

CharacterLength: TYPE = {lengthIs5bits, lengthIs6bits, lengthIs7bits, lengthIs8bits};
LineSpeed: TYPE = {bps50, bps75, bps110, bps134p5, bps150, bps300, bps600, bps1200, bps1800, bps2000, bps2400, bps3600,
        bps4800, bps7200, bps9600, bps19200};
Parity: TYPE = {none, odd, even};
StopBits: TYPE = {none, one, oneAndHalf, two};

END. -- TTYPortEnvironment
```

LOG

Time: August 28, 1980 10:24 AM By: Mary Artibee Action: Created.

-- Last edited: August 28, 1980 10:22 AM By: Mary Artibee

```
DIRECTORY
    TTYPortEnvironment: FROM "TTYPortEnvironment" USING [CharacterLength, LineSpeed, Parity, StopBits];

TTYPortFace: DEFINITIONS =

BEGIN

-- Procedures, alphabetically

GetCommand: PROCEDURE [lineNumber: CARDINAL] RETURNS [data: CHARACTER, stat: TransferStatus];
GetLineCount: PROCEDURE RETURNS [lineCount: CARDINAL];
GetStatus: PROCEDURE [lineNumber: CARDINAL] RETURNS [stat: DeviceStatus];
Off: PROCEDURE [lineNumber: CARDINAL];
On: PROCEDURE [lineNumber: CARDINAL, mask: UNSPECIFIED];
PutCommand: PROCEDURE [lineNumber: CARDINAL, data: CHARACTER] RETURNS [stat: TransferStatus];
SetParameter: PROCEDURE [lineNumber: CARDINAL, parameter: Parameter];

-- Type Definitions

Parameter: TYPE = RECORD [SELECT parameter: * FROM
    characterLength => [characterLength: CharacterLength],
    clearToSend => [clearToSend: BOOLEAN],
    dataSetReady => [dataSetReady: BOOLEAN],
    lineSpeed => [lineSpeed: LineSpeed],
    parity => [parity: Parity],
    stopBits => [stopBits: StopBits],
    ENDCASE];

CharacterLength: TYPE = TTYPortEnvironment.CharacterLength;
LineSpeed: TYPE = TTYPortEnvironment.LineSpeed;
Parity: TYPE = TTYPortEnvironment.Parity;
StopBits: TYPE = TTYPortEnvironment.StopBits;

DeviceStatus: TYPE = RECORD [
    dataTerminalReady: BOOLEAN,
    readyToGet: BOOLEAN,
    readyToPut: BOOLEAN,
    requestToSend: BOOLEAN];

TransferStatus: TYPE = {success, parityError, asynchFramingError, dataLost, breakDetected, notReady};


END. -- TTYPortFace
```

LOG

Time: July 18, 1980 3:26 PM By: Mary Artibee Action: Created file from FrontFace.
Time: July 21, 1980 3:28 PM By: Mary Artibee Action: Many changes.
Time: August 28, 1980 10:21 AM By: Mary Artibee Action: Use TTYPortEnvironment.

```
-- Last edited:  September 30, 1980  10:03 AM  By: Mary Artibee

DIRECTORY
    TTYPort: FROM "TTYPort" USING [CharacterLength, DeviceStatus, LineSpeed, Parity, StopBits, TransferStatus];

TTYPortInternal: DEFINITIONS =

BEGIN

DeleteChannel: PROCEDURE [channel: LONG POINTER TO ChannelStatus];
InitializeInterrupts: PROCEDURE [channel: LONG POINTER TO ChannelStatus];

channelIsCreated: READONLY PACKED ARRAY [0..15] OF BOOLEAN;

ChannelStatusHandle: TYPE = LONG POINTER TO ChannelStatus;

ChannelStatus: TYPE = MONITORED RECORD [
    -- make certain the Monitor LOCK is initialized correctly by Mesa or by a Runtime call
    deleteInProgress: BOOLEAN,
    breakBeingProcessed: BOOLEAN,
    myDataLost: BOOLEAN,
    lineNumber: CARDINAL,
    parameterRecord: ParameterRecord,
    deviceStatus: TTYPort.DeviceStatus,
    bufferInput: CONDITION, -- wakes up when character added to buffer
    breaksInBuffer: CARDINAL,
    charsInBuffer: CARDINAL,
    topInUse: CARDINAL,
    topUnused: CARDINAL,
    buffer: ARRAY [0..maxBufferSize) OF BufferRecord,
    interrupt: PROCESS,
    wakeUpPtr: LONG POINTER TO CONDITION, -- wakes up when microcode does something interesting, TxRdy/RxRdy
    statusChange: CONDITION,
    statusWaitCount: CARDINAL];

maxBufferSize: CARDINAL = 16;

BufferRecord: TYPE = RECORD [
    char: CHARACTER,
    stat: TTYPort.TransferStatus];

ParameterRecord: TYPE = RECORD [
    characterLength: TTYPort.CharacterLength,
    clearToSend: BOOLEAN,
    dataSetReady: BOOLEAN,
    lineSpeed: TTYPort.LineSpeed,
    parity: TTYPort.Parity,
    stopBits: TTYPort.StopBits];

END. -- TTYPortInternal


LOG

Time: July 28, 1980  5:22 PM  By: Mary Artibee     Action: Created.
Time: July 29, 1980  10:51 PM     By: Mary Artibee     Action: Changed ChannelStatus, BufferRecord, ParameterRecord; added
        maxBufferSize, InitializeInterrupts, bufferInput condition.
Time: August 5, 1980  1:36 PM     By: Mary Artibee     Action: Changed ptrWakeUp to be POINTER.
Time: August 6, 1980  4:29 PM     By: Mary Artibee     Action: Moved ttyPortDefaults to TTYPortDriverA.Create.
Time: September 3, 1980  12:09 AM     By: Mary Artibee     Action: wakeUpPtr now LONG POINTER (NakedCondition now available in
        ProcessInternal).
Time: September 30, 1980  10:04 AM    By: Mary Artibee     Action: handle breakDetected, dataLost properly; added
        breakBeingProcessed, myDataLost, breaksInBuffer.
```

```
-- UserTerminal.mesa (last edited by: McJones on: September 2, 1980  2:58 PM)

DIRECTORY
  BitBlt USING [BBTable];

UserTerminal: DEFINITIONS =
  BEGIN

-- Types

Coordinate: TYPE = MACHINE DEPENDENT RECORD [x, y: INTEGER];
CursorArray: TYPE = ARRAY [0..16) OF WORD;

State: TYPE = {on, off, disconnected};
Background: TYPE = {white, black};

-- Errors

BitmapIsDisconnected: ERROR; -- only raised by GetBitBltTable

-- Constants (modulo display hardware)

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

-- Keyboard, mouse and cursor memory locations

keyboard: READONLY LONG POINTER TO READONLY ARRAY OF WORD;
mouse: READONLY LONG POINTER TO READONLY Coordinate;
cursor: READONLY LONG POINTER TO Coordinate;

-- Mouse & cursor functions available only through procedures

SetMousePosition: PROCEDURE [newMousePosition: Coordinate];

SetCursorPattern: PROCEDURE [cursorPattern: CursorArray];
GetCursorPattern: PROCEDURE RETURNS [cursorPattern: CursorArray];

-- Bitmap functions

WaitForScanLine: PROCEDURE [scanLine: INTEGER];

SetBorder: PROCEDURE [oddPairs, evenPairs: [0..377B]];
hasBorder: READONLY BOOLEAN;

BlinkDisplay: PROCEDURE;
GetBackground: PROCEDURE RETURNS [background: Background];
SetBackground: PROCEDURE [new: Background] RETURNS [old: Background];

GetState: PROCEDURE RETURNS [state: State];
SetState: PROCEDURE [new: State] RETURNS [old: State];

GetBitBltTable: PROCEDURE RETURNS [bbt: BitBlt.BBTable];

END.
```

```
DIRECTORY
    Environment USING [Long, PageNumber, wordsPerPage],
    Inline USING [LongMult],
    Mopcodes USING [zLI4, zRFSL, zSHIFT, zWFSL],
    PrincOps USING [FieldDescriptor];

Utilities: DEFINITIONS IMPORTS Inline =
BEGIN

LongMove: PROC [pSource: LONG POINTER, size: CARDINAL, pSink: LONG POINTER];
    -- Moves 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

ShortCARDINAL: PROC [cc: LONG UNSPECIFIED] RETURNS [CARDINAL] =
    -- Assumes cc IN CARDINAL
    INLINE { RETURN[LOOPHOLE[cc, Environment.Long].lowbits] };

-- Inline copies of Page < = > LongPointer for people not in positions to make procedure calls
LongPointerFromPage: PROC [page: Environment.PageNumber] RETURNS [LONG POINTER] =
    INLINE { RETURN[LOOPHOLE[Inline.LongMult[page, Environment.wordsPerPage]]] };

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

--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- Packed Bit Array operations:
--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

-- Note: packed bit arrays are currently limited to a maximum of 4096 bits.

Bit: TYPE = CARDINAL[0..1];

BitIndex: TYPE = CARDINAL[0..4096); -- (restriction of current implementation)

BitArray: TYPE = UNSPECIFIED;  -- TIGHTLY PACKED ARRAY (0..0) OF Bit (no array descriptor)

BitGet: PROC [lp: LONG POINTER TO BitArray, bitIndex: BitIndex] RETURNS [Bit] =
    INLINE { RETURN[BitGetFromBitDesc[lp, BitDescFromBitIndex[bitIndex]]] };

BitPut: PROC [bit: Bit, lp: LONG POINTER TO BitArray, bitIndex: BitIndex] =
    INLINE { BitPutFromBitDesc[bit, lp, BitDescFromBitIndex[bitIndex]] };

-- Private to implementation:

BitDescriptor: PRIVATE TYPE = PrincOps.FieldDescriptor;

BitGetFromBitDesc: PRIVATE PROC [lp: LONG POINTER TO BitArray, bd: BitDescriptor] RETURNS [Bit] =
    MACHINE CODE BEGIN Mopcodes.zRFSL END;

BitPutFromBitDesc: PRIVATE PROC [bit: Bit, lp: LONG POINTER TO BitArray, bd: BitDescriptor] =
    MACHINE CODE BEGIN Mopcodes.zWFSL END;

-- shifts left 4 bits (therefore, bitIndex must be < 4096).
BitDescFromBitIndex: PRIVATE PROC [bitIndex: BitIndex] RETURNS [BitDescriptor] =
    MACHINE CODE BEGIN Mopcodes.zLI4; Mopcodes.zSHIFT END;
END.
```

LOG

Time: April 24, 1980 4:52 PM By: Forrest Action: Trimmed log Amargosa. Converted to use PrincOps.
Time: April 24, 1980 4:52 PM By: Forrest Action: Made Page < = > LongPointer INLINE.

```
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

```
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

```
-- This interface defines operations for allocating/deallocating windows and single pages of backing storage for use by the Virtual
        Memory Manager.

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

VMMgrStore: DEFINITIONS =

BEGIN

AllocateWindow: PROCEDURE [pWindowResult: LONG POINTER TO Space.WindowOrigin, count: VM.PageCount];
    -- allocates a window, returned at pWindowResult↑.

DeallocateWindow: PROCEDURE [pWindow: LONG POINTER TO Space.WindowOrigin, count: VM.PageCount];

AllocateMapLogFile: PROCEDURE [pWindowResult: LONG POINTER TO Space.WindowOrigin] RETURNS [count: VM.PageCount];
    -- allocates a special window for the Map Log, returned at pWindowResult↑.

END.
```

LOG

Time: November 26, 1979 11:46 AM By: Knutsen Action: Created file.

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

VMMPrograms: DEFINITIONS =

BEGIN

HierarchyImpl: PROGRAM;

MapLogImpl: PROGRAM [pMapLogDesc: LONG POINTER TO UNSPECIFIED];

ProjectionImpl: PROGRAM [countVM: VM.PageCount];

InitializeSpace: PROCEDURE [countVM: VM.PageCount, 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*
Time: August 27, 1979 12:27 PM	*By: Knutsen*	*Action: Changed SpaceImpl: PROGRAM to InitSpace: PROCEDURE.*
Time: November 26, 1979 2:59 PM	*By: Knutsen*	*Action: Deleted parameter volumeData of InitSpace.*
Time: January 28, 1980 11:21 AM	*By: Forrest*	*Action: Change InitSpace: Procedure to SpaceImpl: PROGRAM.*
Time: April 9, 1980 9:34 AM	*By: Knutsen*	*Action: Changed SpaceImpl: PROGRAM to InitializeSpace: PROCEDURE AGAIN!*

```
DIRECTORY
    FileInternal: FROM "FileInternal" USING [FilePtr, PageGroup],
    LogicalVolume: FROM "LogicalVolume" USING [Handle, PageNumber];

VolAllocMap: DEFINITIONS =
BEGIN

FilePtr: TYPE = FileInternal.FilePtr;
GroupPtr: TYPE = POINTER TO FileInternal.PageGroup;
Handle: TYPE = LogicalVolume.Handle;

-- stoping operation (called from VolumeImpl, scavengeImpl)
Close: PROCEDURE [final: BOOLEAN];   -- The sense of "final" is TRUE = flush Logical Volume/FALSE = stopping use for now


-- page group interfaces writing labels (called from FileImpl)
AllocPageGroup: PROCEDURE [vol: Handle, filePtr: FilePtr, groupPtr: GroupPtr, createFile: BOOLEAN]; -- allocates a group of pages and
        writes labels (group is a modifiable hint)
FreePageGroup: PROCEDURE [vol: Handle, filePtr: FilePtr, groupPtr: GroupPtr, deleteFile: BOOLEAN]; -- free a page group (deleting file if
        filePage = 0 and deleteFile = TRUE);

-- Called from VolFileMapImpl to get new file map pages
AccessVAM: PROCEDURE [vol: Handle, volumePage: LogicalVolume.PageNumber, set, clear: BOOLEAN] RETURNS [busy: BOOLEAN]; -- set,
        clear or read one bit of vam, always returns the (previous) value;
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*

Time: *August 28, 1979 5:35 PM By: Forrest Action: Changed Procs to use LogicalVolumeOps.Handle; took following stuff out of main body:*

```
--File: FROM "File" USING [ID, Type],
-- page group interfaces writing labels
-- GetPageGroup: PROCEDURE [volume: Volume.ID, hint: FileInternal.PageGroup] RETURNS [FileInternal.PageGroup];
-- 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];
-- 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
```

Time: *October 1, 1979 5:33 PM By: Forrest Action: took out Open:*

```
Open: PROCEDURE [vol: LogicalVolume.Handle, init, rebuild: BOOLEAN];
```

Time: *March 7, 1980 8:43 PM By: Forrest Action: Leaxically redefined types so would compile with new logicalVolume; take Handle out of close*

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

VolFileMap: DEFINITIONS =

BEGIN

FilePtr: TYPE = FileInternal.FilePtr;
GroupPtr: TYPE = POINTER TO FileInternal.PageGroup;
Handle: TYPE = LogicalVolume.Handle;
Interval: TYPE = LogicalVolume.Interval;
Key: TYPE = LogicalVolume.Key;
Level: TYPE = LogicalVolume.Level;


-- public interfaces
Close: PROCEDURE [final: BOOLEAN]; -- The sense of final is true = flush, false = forceout

GetPageGroup: PROCEDURE [vol: Handle, filePtr: 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 [vol: Handle, filePtr: FilePtr, groupPtr: GroupPtr];
    -- inserts a pageGroup into B-tree (unordered inserts are merged for rebuild)
DeletePageGroup: PROCEDURE [vol: Handle, filePtr: FilePtr, groupPtr: GroupPtr];
    -- deletes a pageGroup suffix (leaving zeroGroup unless explicitly asked)

InitMap: PROCEDURE [vol: Handle]; -- called  by scavanger

END.



LOG
Time: timeStamp    By: initials      Action: shortDescription
Time: April 13, 1978  2:10 PM By: SP    Action: Created file
Time: October 1, 1979  5:38 PM    By: Forrest     Action: Added InitMap[]; took out ...
    -- 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];


    -- (previously) 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];
    -- Open: PROCEDURE [vol: LogicalVolume.Handle, init, rebuild: BOOLEAN];
Time: March 7, 1980  8:45 PM      By: Forrest     Action: Eliminated GetNextFileProc; lexically changed type definitions; deleted Handle
        from close
```

```
DIRECTORY
    File USING [Capability],
    System USING [NetworkAddress, nullID, VolumeID];

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: PROC RETURNS [ID] = INLINE { RETURN[systemID] };

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

    -- Attributes
Type: TYPE = MACHINE DEPENDENT {normal(0), debugger(1), debuggerDebugger(2), nonPilot(3)};
maxNameLength: CARDINAL = 40; -- maximun name length of a volume's labelString.
GetAttributes: PROC [volume: ID] RETURNS [volumeSize, freePageCount: PageCount, rootFile: File.Capability];
GetLabelString: PROC [volume: ID, s: STRING];
GetType: PROC [volume: ID] RETURNS [type: Type];
SetRootFile: PROC [volume: ID, file: File.Capability];

    -- Mounting and unmounting volumes
Status: TYPE = {unknown, partiallyOnLine, closedAndInconsistent, closedAndConsistent, open};
BooleanDefaultFalse: TYPE = BOOLEAN ← FALSE;
TypeSet: TYPE = PACKED ARRAY Type OF BooleanDefaultFalse;
onlyEnumerateCurrentType: TypeSet = []; -- all false
Close, Open: PROC [volume: ID];
GetNext: PROC [volume: ID, includeWhichVolumes: TypeSet ← onlyEnumerateCurrentType] RETURNS [nextVolume: ID];
        -- returns ID's of all completely online (but not necessarily open) volumes. If all elements of includeWhichVolumes are FALSE, returns only
        volumes satisfying GetType[nextVolume] = GetType[systemID]; if any elements of includeWhichVolumes are TRUE, returns only volumes
        satisfying includeWhichVolumes[GetType[nextVolume]] = TRUE.
GetStatus: PROC [volume: ID] RETURNS [Status];

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

    -- Signals and errors
InsufficientSpace: ERROR;
NeedsScavenging: ERROR;
NotOpen: ERROR [volume: ID];
Unknown: ERROR [volume: ID];

END.
```

LOG

Time: January 25, 1980 7:53 PM By: Forrest Action: Deleted InvalidLabelString
Time: May 15, 1980 4:28 PM By: McJones Action: Added NeedsScavenging
Time: July 13, 1980 10:12 PM By: Forrest Action: Moved in GetStatus (originally in PhysicalVolume)
Time: July 15, 1980 10:27 PM By: Forrest Action: Move in GetType, expand GetNext
Time: August 15, 1980 2:49 PM By: McJones Action: VolumeSet = > TypeSet

```
DIRECTORY
    Volume USING [ID];

VolumeExtras: DEFINITIONS =
BEGIN
OpenVolume: PROCEDURE [volume: Volume.ID, readOnly: BOOLEAN ← FALSE];

END.
```

LOG

Time: January 25, 1980 7:53 PM By: Luniewski Action: Created file.

```
DIRECTORY
    File USING [ID, PageNumber, Type],
    FileInternal USING [Descriptor, PageGroup],
    LogicalVolume USING [Handle, VolumeAccessProc],
    PhysicalVolume USING [ID, PageNumber],
    PhysicalVolumeFormat USING [Handle, SubVolumeDesc],
    System USING [UniversalID],
    Volume USING [ID, PageCount, Type],
    VolumeInternal USING [PageNumber];

VolumeImplInterface: DEFINITIONS =
BEGIN
-- These interfaces are used by PhysicalVolumeImpl VolumeImpl to get at needed procedures in each of them. These should probably not be called from
    any other module.

BarePvID: TYPE = System.UniversalID;

readOrWrite: TYPE = {read, readWrite};

AccessPhysicalVolumeRootPage: PROCEDURE [id: BarePvID, proc: PROCEDURE [PhysicalVolumeFormat.Handle], access: readOrWrite ←
  read];

CheckLogicalVolume: PROCEDURE [vID: Volume.ID];

LogicalVolumeCreate: PROCEDURE [pvID: PhysicalVolume.ID, size: Volume.PageCount, name: STRING, type: Volume.Type,
        minPVPageNumber: PhysicalVolume.PageNumber ← 1] RETURNS [Volume.ID];

LogicalVolumeErase: PROCEDURE [lvID: Volume.ID];

FindLogicalVolume: PROCEDURE [vID: POINTER TO READONLY Volume.ID] RETURNS [BOOLEAN];

GetLVStatus: PROCEDURE [vID: Volume.ID] RETURNS [onLine, open: BOOLEAN];

OpenInitialVolumes: PROCEDURE;

PinnedFileEnter: PROCEDURE[fd: FileInternal.Descriptor, grp: FileInternal.PageGroup];

PinnedFileFlush: PROCEDURE [fd: File.ID];

RegisterLogicalSubvolume: PROCEDURE [sv: PhysicalVolumeFormat.SubVolumeDesc, pvID: PhysicalVolume.ID];

RegisterVFiles: PROCEDURE [v: LogicalVolume.Handle];

SignalVolumeAccess: PROCEDURE [pVID: POINTER TO READONLY Volume.ID, proc: LogicalVolume.VolumeAccessProc];

SubvolumeOffline: PROCEDURE [lvID: Volume.ID, root: BOOLEAN];

SubvolumeOnline: PROCEDURE [lvID: Volume.ID, root: BOOLEAN];

UnregisterVFiles: PROCEDURE [v: LogicalVolume.Handle];

VFileEnter: PROCEDURE [volume: Volume.ID, file: File.ID, fileAndVolPage: File.PageNumber, nextPage: VolumeInternal.PageNumber, type:
    File.Type];

END.....
```

Time: October 7, 1979 11:05 AM By: Forrest Action: Split out from VolumeImpl.
Time: March 7, 1980 11:46 PM By: Forrest Action: Moved FreeVolumePages to LogicalVolume. Took Out LV[online, find, offline].
Time: May 28, 1980 4:05 PM By: Luniewski Action: PhysicalVolume = > PhysicalVolumeFormat. Added UnregisterVFiles, GetHints. Made PinnedFileFlush be real. Deleted logical and physical volume specific stuff as part of reorganizing the volume implementation into separate logical and physical volume implementation modules.
Time: July 28, 1980 9:18 AM By: Luniewski Action: Addded LogicalVolumeCreate, LogicalVolumeErase and GetContainingPhysicalVolume so VolumeImpl need not export PhysicalVolume which fails due to a compiler "feature" of not permitting the specification the destination of an Export.
Time: September 17, 1980 10:12 AM By: Luniewski Action: Make SignalVolumeAccess take a

LogicalVolume.AccessProc as a second argument.

```
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.
Time: January 30, 1980 1:57 PM By: Gobbel Action: altoVolume commented out.

```
-- XferTrap.mesa  (last edited by: Johnsson on: September 2, 1980  12:37 PM)

DIRECTORY
  Mopcodes USING [zLLB, zRR, zWR],
  PrincOps USING [ControlLink, XTSreg];

XferTrap: DEFINITIONS =
  BEGIN

  -- At the end of each Xfer (before instruction fetch) the status is
  -- shifted right one bit.  If the bit shifted off the right end is a one,
  -- an Xfer trap is started with the original destination of the Xfer
  -- as the parameter and the new L as the source.

  Status: TYPE = MACHINE DEPENDENT {
    off(0), on(1), skip1(2), skip2(4), skip3(8), skip4(16), (65535)};

  ReadXTP: PROCEDURE RETURNS [PrincOps.ControlLink] =
    MACHINE CODE {Mopcodes.zLLB, 3};

  ReadXTS: PROCEDURE RETURNS [Status] =
    MACHINE CODE {Mopcodes.zRR, PrincOps.XTSreg};

  WriteXTS: PROCEDURE [Status] =
    MACHINE CODE {Mopcodes.zWR, PrincOps.XTSreg};

  END...


LOG

Time: August 28, 1980  10:17 AM; By: Johnsson; Action: Created File
```

```
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

```
DIRECTORY
    Zone: FROM "Zone" USING [BlockSize];

ZoneInternal: DEFINITIONS =
BEGIN

    -- This interface defines the basic structure of a zone - the form of its various headers and pointers.

Base: TYPE = LONG ORDERED BASE POINTER;

RPtr: TYPE = Base RELATIVE ORDERED POINTER;

NodePointer: TYPE = Base RELATIVE ORDERED POINTER TO NodeHeader;
    FreeNodePointer: TYPE = Base RELATIVE ORDERED POINTER TO free NodeHeader;
    InuseNodePointer: TYPE = Base RELATIVE ORDERED POINTER TO inuse NodeHeader;

NodeHeader: TYPE = RECORD [
    length: Zone.BlockSize,
    extension: SELECT state: * FROM
        inuse => NULL,
        free => [fwdp, backp: FreeNodePointer],
    ENDCASE];

ZonePointer: TYPE = LONG ORDERED POINTER TO ZoneHeader;

ZoneHeader: TYPE = MONITORED RECORD [
    node: free NodeHeader,
    freeList, rover: FreeNodePointer,
        -- freeList always points to node, the head of the free list
        -- rover is a roving pointer which points into the middle of the (circular) free list to slow down fragmentation (see Knuth, Vol I, p. 597 #6)
    length: Zone.BlockSize,
    nextSegment: SegmentPointer, -- link to additional segments of zone
    zoneBase: Base,
    threshold: Zone.BlockSize,
    checking: BOOLEAN];

SegmentPointer: TYPE = Base RELATIVE ORDERED POINTER TO SegmentHeader;

SegmentHeader: TYPE = RECORD [
    nextSegment: SegmentPointer, -- link to additional segments of zone
    length: Zone.BlockSize];

END.
```

LOG

Time: March 31, 1980 4:44 PM By: Luniewski Action: Created file
Time: timeStamp By: yourName Action: shortDescription