```
-- InternalNub.mesa;
-- Edited by:
--            Sandman on May 22, 1978  11:55 AM
--            Barbara on July 18, 1978  3:22 PM
--            Johnsson on July 18, 1978  2:47 PM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [BYTE, BytesPerPage, PageNumber],
  AltoFileDefs: FROM "altofiledefs" USING [CFP, eofDA, fillinDA, vDA],
  BFSDefs: FROM "bfsdefs" USING [ActOnPages, GetNextDA, MakeCFP],
  BootDefs: FROM "bootdefs" USING [GetSystemTable],
  ControlDefs: FROM "controldefs" USING [
    ControlLink, FieldDescriptor, FrameHandle, GetReturnFrame, GetReturnLink,
    GlobalFrameHandle, Greg, Lreg, NullFrame, SetReturnFrame, SetReturnLink,
    StateVector],
  CoreSwapDefs: FROM "coreswapdefs" USING [
    BBHandle, callDP, DebugParameter, ExternalStateVector, PuntInfo,
    PuntTable, startDP, SVPointer, SwapReason, UBBPointer, UserBreakBlock],
  DebugData: FROM "debugdata" USING [
    DebuggeeFH, DebuggeeFP, DebuggerFP, SelfFP, userwindow],
  DebugMiscDefs: FROM "debugmiscdefs" USING [
    DebugAbort, DebugInit, InitializeDebuggerFiles],
  DiskDefs: FROM "diskdefs" USING [DiskRequest, RealDA],
  FrameDefs: FROM "framedefs" USING [LockCode, UnlockCode, UnNew],
  ImageDefs: FROM "imagedefs" USING [
    AbortMesa, AddFileRequest, FileRequest, StopMesa, UserCleanupProc],
  InternalNubDefs: FROM "internalnubdefs" USING [InternalDebugCommand],
  IODefs: FROM "iodefs" USING [
    CR, NewLine, WriteChar, WriteLine, WriteOctal, WriteString],
  KeyDefs: FROM "keydefs" USING [Keys],
  LoadStateDefs: FROM "loadstatedefs" USING [GetLoadState, SetLoadState],
  MiscDefs: FROM "miscdefs" USING [SetBlock],
  Mopcodes: FROM "mopcodes" USING [zRFS],
  NovaOps: FROM "novaops" USING [NovaJSR],
  NucleusDefs: FROM "nucleusdefs" USING [Wart],
  ProcessDefs: FROM "processdefs" USING [
    Aborted, DisableInterrupts, EnableInterrupts],
  SDDefs: FROM "sddefs" USING [
    sBreakBlock, sBreakBlockSize, sCallDebugger, sCoreSwap, SD, sFirstFree,
    sInterrupt, sProcessBreakpoint, sUncaughtSignal],
  SegmentDefs: FROM "segmentdefs" USING [
    Append, CopyDataToFileSegment, CopyFileToDataSegment, DataSegmentAddress,
    DataSegmentHandle, DefaultBase, DefaultVersion, DeleteDataSegment,
    DeleteFileSegment, EnumerateFileSegments, FileHandle, FileSegmentHandle,
    GetEndOfFile, GetFileSegmentDA, LockFile, NewDataSegment, NewFile,
    NewFileSegment, Read, ReleaseFile, SetEndOfFile, SetFileAccess,
    SetFileSegmentDA, UnlockFile, Write],
  SystemDefs: FROM "systemdefs" USING [PagesForWords],
  TrapDefs: FROM "trapdefs" USING [ParityError, PhantomParityError],
  WindowDefs: FROM "windowdefs" USING [
    GetCurrentDisplayWindow, SetCurrentDisplayWindow, SetFileForWindow,
    WindowHandle];

DEFINITIONS FROM CoreSwapDefs;

InternalNub: PROGRAM [user: PROGRAM]
  IMPORTS BFSDefs, BootDefs, DDptr: DebugData, DebugMiscDefs, DiskDefs,
    FrameDefs, ImageDefs, InternalNubDefs, IODefs, LoadStateDefs, MiscDefs,
    NucleusDefs, ProcessDefs, SegmentDefs, SystemDefs, TrapDefs, WindowDefs
  EXPORTS CoreSwapDefs, InternalNubDefs
  SHARES BootDefs, SegmentDefs, ControlDefs =

  BEGIN

  FileHandle: TYPE = SegmentDefs.FileHandle;

  ProcessBreakpoint: PROCEDURE [s: CoreSwapDefs.SVPointer] =
    BEGIN -- called by BRK trap handler in resident code
    inst: AltoDefs.BYTE;
    swap: BOOLEAN;
    [inst, swap] ← DoBreakpoint[s];
    IF swap THEN
      BEGIN
      FrameDefs.LockCode[s.dest];
      CoreSwap[breakpoint, s];
      FrameDefs.UnlockCode[s.dest];
```

```
    END
  ELSE s.instbyte ← inst;  --replant the instruction and go on
  RETURN
  END;

DoBreakpoint: PROCEDURE [s: CoreSwapDefs.SVPointer]
  RETURNS [AltoDefs.BYTE, BOOLEAN] =
  BEGIN OPEN SDDefs;
  ubb: CoreSwapDefs.UBBPointer;
  bba: BBHandle = SD[sBreakBlock];
  i: CARDINAL;
  l: ControlDefs.FrameHandle ← s.dest;
  FOR i IN [0..bba.length) DO
    ubb ← @bba.blocks[i];
    IF ubb.frame = l.accesslink AND ubb.pc = l.pc THEN
      IF TrueCondition[ubb, s.source] THEN EXIT
      ELSE RETURN[ubb.inst, FALSE];
    ENDLOOP;
  RETURN[0, TRUE];
  END;

TrueCondition: PROCEDURE [ubb: CoreSwapDefs.UBBPointer,
  base: ControlDefs.FrameHandle] RETURNS [BOOLEAN]  =
  BEGIN --decide whether to take the breakpoint
  fd: ControlDefs.FieldDescriptor;
  left, right: UNSPECIFIED;
  IF ubb.counterL THEN
    RETURN[(ubb.ptrL ← ubb.ptrL + 1) = ubb.ptrR];
  fd ← [offset: 0, posn: ubb.posnL, size: ubb.sizeL];
  left ← IF ~ubb.localL THEN ReadField[ubb.ptrL, fd]
  ELSE ReadField[(base + LOOPHOLE[ubb.ptrL, CARDINAL]), fd];
  IF ~ubb.immediateR THEN
    BEGIN
    fd ← [offset: 0, posn: ubb.posnR, size: ubb.sizeR];
    right ← IF ~ubb.localR THEN ReadField[ubb.ptrR, fd]
    ELSE ReadField[(base + LOOPHOLE[ubb.ptrR, CARDINAL]), fd];
    END
  ELSE right ← ubb.ptrR;
  RETURN[SELECT ubb.relation FROM
    lt  => left < right,
    gt  => left > right,
    eq  => left = right,
    ne  => left # right,
    le  => left <= right,
    ge  => left >= right,
    ENDCASE => FALSE]
  END;

ReadField: PROCEDURE [POINTER, ControlDefs.FieldDescriptor] RETURNS [UNSPECIFIED] =
  MACHINE CODE BEGIN Mopcodes.zRFS END;

NumberBlocks: CARDINAL = 5;

InitBreakBlocks: PROCEDURE =
  BEGIN OPEN SDDefs;
  sd: POINTER TO ARRAY [0..0) OF UNSPECIFIED ← SD;
  sd[sBreakBlock] ← @sd[sFirstFree];
  sd[sBreakBlockSize] ← SIZE[UserBreakBlock]*NumberBlocks+1;
  sd[sFirstFree] ← 0;
  RETURN
  END;

SwatBreak: PROCEDURE [s: CoreSwapDefs.SVPointer] =
  BEGIN OPEN ControlDefs, NovaOps;
  break: RECORD[a,b: WORD];
  break ← [77400B, 1400B];
  s.instbyte ← NovaJSR[JSR, @break, 0];
  RETURN
  END;

Interrupt: PROCEDURE =
  BEGIN -- called by BRK trap handler in resident code
  state: ControlDefs.StateVector;
  state ← STATE;
  state.dest ← REGISTER[ControlDefs.Lreg];
  CoreSwap[breakpoint, @state];
```

```
    END;

  Catcher: PROCEDURE [msg, signal: UNSPECIFIED, f: ControlDefs.FrameHandle] =
    BEGIN
    OPEN ControlDefs;
    SignallerGF: GlobalFrameHandle;
    state: StateVector;
    frame: FrameHandle;
    SignallerGF ← GetReturnFrame[].accesslink;
    state.stk[0] ← msg;
    state.stk[1] ← signal;
    state.stkptr ← 0;
    -- the call stack below here is: Signaller, [Signaller,] offender
    state.dest ← frame ← GetReturnFrame[].returnlink.frame;
    IF frame.accesslink = SignallerGF THEN state.dest ← frame.returnlink;
    BEGIN
      CoreSwap[uncaughtsignal, @state
        ! DebugMiscDefs.DebugAbort => GOTO Abort];
    EXITS
      Abort =>
        IF signal = ProcessDefs.Aborted THEN
          BEGIN BackStop[f]; ERROR KillThisTurkey; END
        ELSE SIGNAL ProcessDefs.Aborted;
    END;
    RETURN
    END;

  BackStop: PROCEDURE [root: ControlDefs.FrameHandle] =
    BEGIN OPEN ControlDefs;
    endProcess: ControlLink ← root.returnlink;
    caller: PROCEDURE = LOOPHOLE[GetReturnLink[]];
    root.returnlink ← LOOPHOLE[REGISTER[Lreg]];
    SetReturnFrame[NullFrame];
    caller[ ! KillThisTurkey => CONTINUE];
    SetReturnLink[endProcess];
    RETURN
    END;

  KillThisTurkey: SIGNAL = CODE;

  CallDebugger: PROCEDURE =
    BEGIN
    state: ControlDefs.StateVector;
    state.stkptr ← 0;
    state.dest ← ControlDefs.GetReturnLink[];
    CoreSwap[explicitcall, @state];
    RETURN
    END;

  -- The non-Swapping information handlers

  ShowBreak: PROCEDURE [s: CoreSwapDefs.SVPointer] =
    BEGIN OPEN IODefs;
    IF ~NewLine[] THEN WriteChar[CR];
    WriteString["*** interrupt ***"L];
    WriteChar['[];
    WriteOctal[REGISTER[ControlDefs.Lreg]];
    WriteChar[']];
    InternalNubDefs.InternalDebugCommand[];
    END;

  ShowInterrupt: PROCEDURE =
    BEGIN
    RETURN
    END;

  ShowSignal: PROCEDURE [msg, signal: UNSPECIFIED, f: ControlDefs.FrameHandle] =
    BEGIN OPEN IODefs;
    IF ~NewLine[] THEN WriteChar[CR];
    SELECT signal FROM
      TrapDefs.PhantomParityError, TrapDefs.ParityError =>
        BEGIN
        IF signal = TrapDefs.PhantomParityError THEN WriteString["Phantom "L];
        WriteString["Parity Error"L];
        IF signal = TrapDefs.ParityError THEN
          BEGIN
```

```
        WriteString[" at "L];
        WriteOctal[msg];
        END;
      WriteLine[" ... Proceeding! "L];
      RETURN;
      END;
    ENDCASE;
  WriteString["*** uncaught SIGNAL "L];
  WriteOctal[signal];
  WriteString[", msg = "L];
  WriteOctal[msg];
  WriteChar[' ];
  WriteChar['[];
  WriteOctal[REGISTER[ControlDefs.Lreg]];
  WriteChar[']];
  InternalNubDefs.InternalDebugCommand[];
  END;

GetDebuggerNub: PROCEDURE =
  BEGIN
  InternalNubDefs.InternalDebugCommand[];
  END;

PauseAtDebuggerNub: PROCEDURE =
  BEGIN OPEN SDDefs;
  BEGIN ENABLE UNWIND => SD[sCallDebugger] ← PauseAtDebuggerNub;
  SD[sCallDebugger] ← CallDebugger;
  InternalNubDefs.InternalDebugCommand[];
  SD[sCallDebugger] ← PauseAtDebuggerNub;
  END;
  END;

-- The core swapper

Quit: SIGNAL = CODE;
DoSwap: PORT [POINTER TO ExternalStateVector];

parmstring: STRING ← [40];

CoreSwap: PUBLIC PROCEDURE [why: SwapReason, sp: SVPointer] =
  BEGIN OPEN NovaOps;
    e: ExternalStateVector;
    DP: DebugParameter;
    decode: PROCEDURE RETURNS [BOOLEAN] =
      BEGIN OPEN ControlDefs; -- decode the SwapReason
      f: GlobalFrameHandle;
      lsv: StateVector;
      SELECT e.reason FROM
        proceed, resume => RETURN[TRUE];
        call =>
          BEGIN
          lsv ← LOOPHOLE[e.parameter, callDP].sv;
          lsv.source ← REGISTER[Lreg];
          TRANSFER WITH lsv;
          lsv ← STATE;
          LOOPHOLE[e.parameter, callDP].sv ← lsv;
          why ← return;
          END;
        start =>
          BEGIN
          f ← LOOPHOLE[e.parameter, startDP].frame;
          IF ~f.started THEN START LOOPHOLE[f, PROGRAM] ELSE RESTART f;
          why ← return;
          END;
        quit => SIGNAL Quit;
        kill => ImageDefs.AbortMesa[];
        showscreen =>
          BEGIN
          UNTIL KeyDefs.Keys.Spare3 = down DO NULL ENDLOOP;
          why ← return;
          END;
        ENDCASE =>
          BEGIN
          RETURN [TRUE];
          END;
      RETURN [FALSE]
```

```
      END;

    -- Body of CoreSwap

    BEGIN OPEN LoadState: LOOPHOLE[e.extension.loadstate, SegmentDefs.FileSegmentHandle];
    e.state ← sp;
    e.drumFile ← SelfFH; -- actually filehandle for DDptr.SelfFP

    DP.string ← parmstring;
    e.parameter ← @DP;
    e.tables ← BootDefs.GetSystemTable[];
    e.extension.loadstate ← LoadStateDefs.GetLoadState[];
    e.loadstateCFA.fp ← LoadState.file.fp;
    e.loadstateCFA.fa ← [page: LoadState.base, byte: 0,
      da: SegmentDefs.GetFileSegmentDA[@LoadState]];
    e.lspages ← LoadState.pages;
    e.fill ← [0,0,0];

    DO
      e.reason ← why;
      ImageDefs.UserCleanupProc[OutLd];
      ProcessDefs.DisableInterrupts[];
      DoSwap[@e];
      ProcessDefs.EnableInterrupts[];
      ImageDefs.UserCleanupProc[InLd];
      IF decode[ !
        DebugMiscDefs.DebugAbort =>
          IF e.level>0 THEN BEGIN why ← return; CONTINUE END;
        Quit => GOTO abort] THEN EXIT
      REPEAT abort => SIGNAL DebugMiscDefs.DebugAbort;
      ENDLOOP;
    END;

    RETURN
    END;

  SetSwappableSD: PUBLIC PROCEDURE =
    BEGIN OPEN SDDefs;
    sd: POINTER TO ARRAY [0..0) OF UNSPECIFIED = SD;
    sd[sProcessBreakpoint] ← ProcessBreakpoint;
    sd[sUncaughtSignal] ← Catcher;
    sd[sInterrupt] ← Interrupt;
    sd[sCallDebugger] ← PauseAtDebuggerNub;
    END;

  SetNonSwappableSD: PUBLIC PROCEDURE =
    BEGIN OPEN SDDefs;
    sd: POINTER TO ARRAY [0..0) OF UNSPECIFIED = SD;
    sd[sProcessBreakpoint] ← ShowBreak;
    sd[sUncaughtSignal] ← ShowSignal;
    sd[sInterrupt] ← ShowInterrupt;
    sd[sCallDebugger] ← GetDebuggerNub;
    END;

  -- File Lookup

  FileRequest: TYPE = ImageDefs.FileRequest;

  DebuggerFR: short FileRequest ← FileRequest[file: NIL, link:,
    access: SegmentDefs.Read, body: short[fill:, name:"MesaDebugger."]];

  CoreFR: short FileRequest ← FileRequest[file: NIL, link:,
    access: SegmentDefs.Read, body: short[fill:, name:"MesaCore."]];

  SwatFR: short FileRequest ← FileRequest[file: NIL, link:,
    access: SegmentDefs.Read, body: short[fill:, name:"Swatee."]];

  DebugDebuggerFR: short FileRequest ← FileRequest[file: NIL, link:,
    access: SegmentDefs.Read, body: short[fill:, name:"MesaDebugDebugger."]];

  SelfFH: FileHandle;
  puntData: PuntTable;

  RequestFiles: PROCEDURE =
    BEGIN
    DebuggerFR.file ← NIL;
```

```
    CoreFR.file ← NIL;
    SwatFR.file ← NIL;
    DebugDebuggerFR.file ← NIL;
    ImageDefs.AddFileRequest[@DebuggerFR];
    ImageDefs.AddFileRequest[@CoreFR];
    ImageDefs.AddFileRequest[@SwatFR];
    ImageDefs.AddFileRequest[@DebugDebuggerFR];
    END;

  DebuggerDebuggerInstall: PUBLIC PROCEDURE =
    BEGIN OPEN SegmentDefs;
    debuggee: FileHandle;
    MoveLoadState[@DebugDebuggerFR];
    [SelfFH, DDptr.SelfFP.leaderDA] ← FixFile[@DebugDebuggerFR, @DDptr.SelfFP];
    [debuggee, DDptr.DebuggeeFP.leaderDA] ←
      FixFile[@DebuggerFR, @DDptr.DebuggeeFP];
    DDptr.DebuggerFP ← DDptr.SelfFP;
    DDptr.DebuggeeFH ← debuggee;
    DebugMiscDefs.InitializeDebuggerFiles[debuggee];
    SetupPuntESV[FALSE, NIL];
    SetNonSwappableSD[];
    ChangeTypescript[];
    RESTART DebugMiscDefs.DebugInit;
    END;

  Install: PUBLIC PROCEDURE =
    BEGIN OPEN SegmentDefs;
    swappable: BOOLEAN ← FALSE;
    debuggee: FileHandle;
    [SelfFH, DDptr.SelfFP.leaderDA] ← FixFile[@DebuggerFR, @DDptr.SelfFP];
    IF CoreFR.file # NIL THEN
      BEGIN
      [debuggee, DDptr.DebuggeeFP.leaderDA] ←
        FixFile[@CoreFR, @DDptr.DebuggeeFP];
      ReleaseFile[SwatFR.file];
      END
    ELSE [debuggee, DDptr.DebuggeeFP.leaderDA] ←
      FixFile[@SwatFR, @DDptr.DebuggeeFP];
    IF DebugDebuggerFR.file # NIL THEN
      BEGIN
      swappable ← TRUE;
      [, DDptr.DebuggerFP.leaderDA] ←
        FixFile[@DebugDebuggerFR, @DDptr.DebuggerFP];
      END;
    DDptr.DebuggeeFH ← debuggee;
    DebugMiscDefs.InitializeDebuggerFiles[debuggee];
    SetupPuntESV[swappable, IF swappable THEN SelfFH ELSE NIL];
    IF swappable THEN SetSwappableSD[] ELSE SetNonSwappableSD[];
    RESTART DebugMiscDefs.DebugInit;
    END;

  FixFile: PROCEDURE [
   fr: POINTER TO short FileRequest, p: POINTER TO AltoFileDefs.CFP]
   RETURNS [f: FileHandle, da: AltoFileDefs.vDA] =
    BEGIN OPEN SegmentDefs;
    s: FileSegmentHandle;
    pages: AltoDefs.PageNumber;
    bytes: CARDINAL;
    IF (f ← fr.file) = NIL THEN f ←
      NewFile[fr.name, Read+Write+Append, DefaultVersion]
    ELSE SetFileAccess[f, Read+Write+Append];
    LockFile[f];
    [pages, bytes] ← GetEndOfFile[f];
    IF pages < 255 OR (pages = 255 AND bytes < AltoDefs.BytesPerPage) THEN
      SetEndOfFile[f, 255, AltoDefs.BytesPerPage];
    s ← SegmentDefs.NewFileSegment[f,1,1,SegmentDefs.Read];
    BFSDefs.MakeCFP[p,@f.fp];
    da ← LOOPHOLE[DiskDefs.RealDA[SegmentDefs.GetFileSegmentDA[s]]];
    SegmentDefs.DeleteFileSegment[s];
    UnlockFile[f];
    RETURN[f, da];
    END;

  SetupPuntESV: PROCEDURE [swappable: BOOLEAN, file: FileHandle] =
    BEGIN OPEN LoadState: LOOPHOLE[puntData.puntESV.extension.loadstate, SegmentDefs.FileSegmentHandle]
**, ControlDefs, AltoFileDefs;
```

```
    puntData.puntESV.reason ← punt;
    puntData.puntESV.tables ← BootDefs.GetSystemTable[];
    puntData.puntESV.drumFile ← file;
    puntData.puntESV.extension.loadstate ← LoadStateDefs.GetLoadState[];
    puntData.puntESV.loadstateCFA.fp ← LoadState.file.fp;
    puntData.puntESV.loadstateCFA.fa ← [page: LoadState.base,
      byte: 0, da: SegmentDefs.GetFileSegmentDA[@LoadState]];
    puntData.puntESV.lspages ← LoadState.pages;
    IF swappable THEN
      BEGIN
      puntData.pDebuggerFP ← @DDptr.DebuggerFP;
      puntData.debuggerFP ← DDptr.DebuggerFP;
      puntData.pCoreFP ← @DDptr.SelfFP;
      puntData.coreFP ← DDptr.SelfFP;
      END
    ELSE puntData.pDebuggerFP ← puntData.pCoreFP ← LOOPHOLE[0];
    PuntInfo↑ ← @puntData;
    puntData.puntESV.fill ← [0,0,0];
    END;

  ChangeTypescript: PROCEDURE =
    BEGIN OPEN WindowDefs;
    w: WindowHandle ← GetCurrentDisplayWindow[];
    SetFileForWindow[w, "InternalDebug.ts."];
    SetFileForWindow[DDptr.userwindow, "Debug.Typescript."];
    SetCurrentDisplayWindow[DDptr.userwindow];
    SetCurrentDisplayWindow[w];
    END;

  MoveLoadState: PROCEDURE [fr: POINTER TO short FileRequest] =
    BEGIN OPEN SegmentDefs;
    f: FileHandle;
    new, old: FileSegmentHandle;
    temp: DataSegmentHandle;
    old ← LoadStateDefs.GetLoadState[];
    IF (f ← fr.file) = NIL THEN f ← fr.file ←
      NewFile[fr.name, Read+Write+Append, DefaultVersion]
    ELSE SetFileAccess[f, Read+Write+Append];
    SetEndOfFile[f, 255+old.pages, AltoDefs.BytesPerPage];
    temp ← NewDataSegment[DefaultBase, old.pages];
    new ← NewFileSegment[f, 256, old.pages, Read+Write];
    CopyFileToDataSegment[old, temp];
    CopyDataToFileSegment[temp, new];
    DeleteDataSegment[temp];
    DeleteFileSegment[old];
    LoadStateDefs.SetLoadState[new];
    END;

  CollectDiskAddresses: PROCEDURE =
    BEGIN OPEN SystemDefs, SegmentDefs, AltoFileDefs;
    ImageFile: FileHandle =
      LOOPHOLE[REGISTER[ControlDefs.Greg], GlobalFrameHandle].codesegment.file;
    DAs: DESCRIPTOR FOR ARRAY OF vDA;
    maxunknown, maxknown: CARDINAL ← FIRST[CARDINAL];
    minunknown: CARDINAL ← LAST[CARDINAL];
    maxknownDA: vDA;
    DisplayHead: POINTER TO WORD = LOOPHOLE[420B];
    DisplayInterruptWord: POINTER TO WORD = LOOPHOLE[421B];
    saveDisplay, saveDiw: WORD;
    diskrequest: DiskDefs.DiskRequest;
    bufseg, DAseg: DataSegmentHandle;
    FindEnds: PROCEDURE [seg: FileSegmentHandle] RETURNS [BOOLEAN] =
      BEGIN
      WITH s: seg SELECT FROM
        disk =>
          IF s.file = ImageFile AND s.hint.da = eofDA THEN
            BEGIN
            maxunknown ← MAX[maxunknown,s.base];
            minunknown ← MIN[minunknown,s.base];
            END;
        ENDCASE;
      RETURN[FALSE];
      END;
    FindKnown: PROCEDURE [seg: FileSegmentHandle] RETURNS [BOOLEAN] =
      BEGIN
      WITH s: seg SELECT FROM
```

```
      disk =>
        IF s.file = ImageFile AND s.hint.da # eofDA AND s.base < minunknown
          AND s.base > maxknown THEN
          BEGIN maxknown ← s.base; maxknownDA ← s.hint.da END;
      ENDCASE;
    RETURN[FALSE];
    END;
  PlugDA: PROCEDURE [seg: FileSegmentHandle] RETURNS [BOOLEAN] =
    BEGIN
    WITH s: seg SELECT FROM
      disk =>
        IF s.file = ImageFile AND s.hint.da = eofDA AND
          s.base IN (maxknown..maxunknown] THEN
          SegmentDefs.SetFileSegmentDA[@s,DAs[s.base]];
      ENDCASE;
    RETURN[FALSE];
    END;

  saveDisplay ← DisplayHead↑;
  saveDiw ← DisplayInterruptWord↑;
  DisplayHead↑ ← DisplayInterruptWord↑ ← 0;
  [] ← EnumerateFileSegments[FindEnds];
  [] ← EnumerateFileSegments[FindKnown];
  bufseg ← NewDataSegment[DefaultBase, 1];
  DAseg ← NewDataSegment[
    DefaultBase, PagesForWords[maxunknown-maxknown+3]];
  DAs ← DESCRIPTOR[DataSegmentAddress[DAseg]-(maxknown-1),maxunknown+2];
  diskrequest ← DiskDefs.DiskRequest [
    ca: DataSegmentAddress[bufseg],
    fixedCA: TRUE,
    da: @DAs[0],
    fp: @ImageFile.fp,
    firstPage: maxknown,
    lastPage: maxunknown,
    action: ReadD,
    lastAction: ReadD,
    signalCheckError: FALSE,
    option: update[cleanup: BFSDefs.GetNextDA]];
  MiscDefs.SetBlock[@DAs[maxknown-1],fillinDA,maxunknown-maxknown+3];
  DAs[maxknown] ← maxknownDA;
  [] ← BFSDefs.ActOnPages[LOOPHOLE[@diskrequest]];  -- we know it is an Update diskrequest
  [] ← EnumerateFileSegments[PlugDA];
  DeleteDataSegment[DAseg];
  DeleteDataSegment[bufseg];
  DisplayHead↑ ← saveDisplay;
  DisplayInterruptWord↑ ← saveDiw;
  RETURN;
  END;

GlobalFrameHandle: TYPE = ControlDefs.GlobalFrameHandle;

-- Main body

P: TYPE = MACHINE DEPENDENT RECORD [in, out: UNSPECIFIED]; -- PORT

SetNonSwappableSD[];
SDDefs.SD[SDDefs.sProcessBreakpoint] ← SwatBreak;
LOOPHOLE[DoSwap,P] ← [in: 0, out: SDDefs.SD[SDDefs.sCoreSwap]];
InitBreakBlocks[];
RequestFiles[];

START user;

STOP;

CollectDiskAddresses[];
FrameDefs.UnNew[LOOPHOLE[NucleusDefs.Wart, GlobalFrameHandle]];
SDDefs.SD[SDDefs.sProcessBreakpoint] ← ShowBreak;
RESTART user;

ImagoDefs.StopMesa[];

END...
```