```
-- ExternalUtilities.Mesa
-- Edited by:
--             Johnsson August 29, 1978  9:37 AM
--             Sandman May 22, 1978  11:51 AM
--             Barbara July 18, 1978  2:59 PM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [PageSize],
  AllocDefs: FROM "allocdefs" USING [DefaultDataSegmentInfo],
  AltoFileDefs: FROM "altofiledefs" USING [CFA, FA],
  ControlDefs: FROM "controldefs" USING [
    EPIndex, EPRange, FrameHandle, GFT, GFTIndex, GFTItem, GlobalFrame,
    GlobalFrameHandle, Lreg, NullGlobalFrame, ProcDesc, SD, StateVector,
    SVPointer, WDCreg, XTSreg],
  CoreSwapDefs: FROM "coreswapdefs" USING [
    callDP, ExternalStateVector, PuntInfo, PuntTable, startDP, SwapReason],
  DebugBreakptDefs: FROM "debugbreakptdefs" USING [Break, ValidateBreakpoint],
  DebugCacheDefs: FROM "debugcachedefs" USING [ProcessMapLog],
  DebugContextDefs: FROM "debugcontextdefs" USING [InitBCD],
  DebugData: FROM "debugdata" USING [
    altoXM, DebuggeeFP, debugPilot, ESV, level, mdsContext, onDO,
    restartmessage, SelfFP, worryentry],
  DebuggerDefs: FROM "debuggerdefs" USING [LA],
  DebugMiscDefs: FROM "debugmiscdefs" USING [
    CopyRead, CopyWrite, DebugAbort, DebugCommand,
    UCSHandler],
  DebugUtilityDefs: FROM "debugutilitydefs" USING [
    FlushCodeSegmentCache, FlushCoreCache,
    LongREAD,MREAD, MWRITE, SREAD, SWRITE],
  ImageDefs: FROM "imagedefs" USING [StopMesa, UserCleanupProc],
  InlineDefs: FROM "inlinedefs" USING [DIVMOD, LongMult],
  IODefs: FROM "iodefs" USING [WriteChar, WriteString],
  LoadStateDefs: FROM "loadstatedefs" USING [
    FileSegmentHandle, GetLoadState, SetLoadState],
  NovaOps: FROM "novaops" USING [NovaInLd, NovaJSR, NovaOutLd],
  OsStaticDefs: FROM "osstaticdefs" USING [OsStatics],
  ProcessDefs: FROM "processdefs" USING [
    CurrentPSB, DefaultPriority, DisableInterrupts, EnableInterrupts,
    Priority, ProcessHandle, PSB],
  SDDefs: FROM "sddefs" USING [
    SD, sFirstProcess, sFirstStateVector, sGFTLength, sLastProcess,
    sXferTrap],
  SegmentDefs: FROM "segmentdefs" USING [
    DeleteFileSegment, FileHandle, FileSegmentHandle, InsertFile,
    NewFileSegment, Read, SetFileSegmentDA, SwapIn, Unlock, Write],
  StringDefs: FROM "stringdefs" USING [SubString];

DEFINITIONS FROM DebugMiscDefs, DebugUtilityDefs;

ExternalUtilities: PROGRAM
  IMPORTS DDptr: DebugData, DebugBreakptDefs, DebugCacheDefs,
    DebugContextDefs, DebugMiscDefs, DebugUtilityDefs,
    ImageDefs, IODefs, LoadStateDefs, SegmentDefs
  EXPORTS DebugUtilityDefs
  SHARES ProcessDefs =

  BEGIN

  FrameHandle: TYPE = ControlDefs.FrameHandle;
  GlobalFrameHandle: TYPE = ControlDefs.GlobalFrameHandle;
  FileHandle: TYPE = SegmentDefs.FileHandle;
  FileSegmentHandle: TYPE = SegmentDefs.FileSegmentHandle;
  SVPointer: TYPE = ControlDefs.SVPointer;
  LA: TYPE = DebuggerDefs.LA;

  DebuggerBooted: BOOLEAN ← FALSE;

  InitDebugger: PUBLIC SIGNAL [sp: SVPointer, message: STRING] = CODE;
  InitUCSHandler: PUBLIC SIGNAL [
    sp: SVPointer, signal: UNSPECIFIED] = CODE;
  InitMapCleanup: PUBLIC SIGNAL = CODE;
  DebuggerError: PUBLIC SIGNAL = CODE;
  saveXMDisplayWordTaskBank: WORD ← 0;
  XMDisplayWordTaskBank: POINTER = LOOPHOLE[177751B];

  CoreSwap: PUBLIC PROCEDURE [why: CoreSwapDefs.SwapReason] =
```

```
BEGIN OPEN DebugCacheDefs, NovaOps, ControlDefs, SDDefs;
wdc: CARDINAL;
xferTrapStatus: CARDINAL;
xferTrapHandler: UNSPECIFIED;
e: CoreSwapDefs.ExternalStateVector ← DDptr.ESV;
flag: [0..2];

IF DebuggerBooted THEN
  BEGIN
  IF why = quit OR why = kill THEN ImageDefs.StopMesa[];
  IODefs.WriteString["Core image not healthy, can't swap"L];
  SIGNAL DebugAbort;
  END;
DO
e.reason ← why;
FlushCodeSegmentCache[];
[] ← FlushCoreCache[0, AllocDefs.DefaultDataSegmentInfo, NIL];
ImageDefs.UserCleanupProc[OutLd];
DDptr.worryentry ← FALSE;
ProcessDefs.DisableInterrupts[];
xferTrapHandler ← SD[sXferTrap];
SD[sXferTrap] ← REGISTER[Lreg];
xferTrapStatus ← REGISTER[XTSreg];
REGISTER[XTSreg] ← 0;
wdc ← REGISTER[WDCreg];
flag ← NovaOutLd[OutLd, @DDptr.SelfFP, @e];
REGISTER[WDCreg] ← wdc;
SD[sXferTrap] ← xferTrapHandler;
ProcessDefs.EnableInterrupts[];
IF flag = 0 THEN    -- just completed OutLd
  BEGIN
  IF why = install THEN ImageDefs.StopMesa[];
  e.level ← DDptr.level;
  XMDisplayWordTaskBank↑ ← saveXMDisplayWordTaskBank;
  NovaInLd[InLd,@DDptr.DebuggeeFP,@e];
  END;

saveXMDisplayWordTaskBank ← XMDisplayWordTaskBank↑;
XMDisplayWordTaskBank↑ ← 0;

REGISTER[ControlDefs.XTSreg] ← xferTrapStatus;

ImageDefs.UserCleanupProc[InLd];

DDptr.ESV ← e;

SELECT flag FROM
  1 =>      -- just started by InLd
    BEGIN OPEN DebugBreakptDefs;
    SetDebugeeType[];
    DebuggerBooted ← FALSE;
    SELECT e.reason FROM
      breakpoint, worrybreak, interrupt =>
        BEGIN
        IF e.reason = worrybreak THEN DDptr.worryentry ← TRUE;
        IF ~ValidateBreakpoint[e.state] OR e.reason = interrupt THEN
          BEGIN
          IF e.level # DDptr.level THEN
            BEGIN
            DDptr.restartmessage ← "*** interrupt ***";
            SetupLoadState[@DDptr.ESV.loadstateCFA, DDptr.ESV.lspages];
            SIGNAL InitDebugger[e.state, NIL];
            END;
          IODefs.WriteString["*** interrupt ***"];
          IF DDptr.debugPilot THEN ProcessMapLog[e.mapLog];
          DebugCommand[e.state];
          END
        ELSE
          BEGIN
          IF DDptr.debugPilot THEN ProcessMapLog[e.mapLog];
          Break[e.state];
          END;
        why ← proceed;
        END;
      explicitcall, cleanmaplog =>
        BEGIN
```

```
            usermessage: STRING ← GetUserMessage[e.state];
            IF e.level # DDptr.level THEN
              BEGIN
              SetupLoadState[@DDptr.ESV.loadstateCFA, DDptr.ESV.lspages];
              IF e.reason = explicitcall THEN
                BEGIN
                SIGNAL InitDebugger[e.state, usermessage];
                END
              ELSE
                BEGIN
                DDptr.restartmessage ← "*** Processing VM Map ***";
                SIGNAL InitMapCleanup;
                END;
              END;
            IF DDptr.debugPilot THEN
              BEGIN
              IF e.reason = cleanmaplog THEN
                IODefs.WriteString["*** Processing VM Map ***"];
              ProcessMapLog[e.mapLog];
              END
            ELSE
              BEGIN
              IF usermessage # NIL THEN UserWriteString[usermessage];
              DebugCommand[e.state]
              END;
            why ← proceed;
            END;
          uncaughtsignal =>
            BEGIN
            IF e.level # DDptr.level THEN
               BEGIN
               SetupLoadState[@DDptr.ESV.loadstateCFA, DDptr.ESV.lspages];
               SIGNAL InitUCSHandler [e.state, MREAD[@e.state.stk[1]]];
               END;
            IF DDptr.debugPilot THEN ProcessMapLog[e.mapLog];
            UCSHandler[e.state, MREAD[@e.state.stk[1]]];
            why ← resume;
            END;
          punt =>
            BEGIN
            DDptr.restartmessage ← "*** Fatal System Error (Punt) ***";
            GOTO booted;
            END;
          return => EXIT;
          ENDCASE => SIGNAL DebuggerError;
        END;
      2 =>      -- just booted
        BEGIN OPEN CoreSwapDefs;
        SetDebugeeType[];
        DDptr.restartmessage ← "*** Debugger Bootloaded ***";
        CopyRead[
          from: @LOOPHOLE[SREAD[PuntInfo], POINTER TO PuntTable].puntESV,
          to: @DDptr.ESV, nwords: SIZE[ExternalStateVector]];
        GOTO booted;
        END;
      ENDCASE;
    REPEAT
      booted =>
        BEGIN
        DebuggerBooted ← TRUE;
        DDptr.ESV.state ← GetCurrentStateFromPSB[];
        SetupLoadState[@DDptr.ESV.loadstateCFA, DDptr.ESV.lspages];
        SIGNAL InitDebugger[DDptr.ESV.state, NIL];
        END;
    ENDLOOP;

    DDptr.ESV ← e;
    RETURN
    END;

  GetUserMessage: PROCEDURE [state: SVPointer] RETURNS [message: STRING] =
    BEGIN OPEN ControlDefs;
    s: StateVector;
    DebugMiscDefs.CopyRead[from: state, to: @s, nwords: SIZE[StateVector]];
    message ← IF s.stkptr = 1 THEN s.stk[0] ELSE NIL;
    s.stkptr ← 0;
```

```
    DebugMiscDefs.CopyWrite[from: @s, to: state, nwords: SIZE[StateVector]];
    RETURN
    END;

  SetDebugeeType: PROCEDURE =
    BEGIN OPEN OsStaticDefs, NovaOps;
    vers: ARRAY [0..1] OF WORD ← [61014B, 1400B];
    OsStatics.AltoVersion ← NovaJSR[JSR, @vers, 0];
    DDptr.onDO ← OsStatics.AltoVersion.engineeringnumber >= 4;
    DDptr.altoXM ← OsStatics.AltoVersion.engineeringnumber = 3;
    DDptr.debugPilot ← FALSE;
    DDptr.mdsContext ← 0;
    IF DDptr.ESV.extension.notMesa40 THEN
      SELECT DDptr.ESV.extension.type FROM
        nonSwappingPilot, pilot =>
          BEGIN DDptr.mdsContext ← DDptr.ESV.mds; DDptr.debugPilot ← TRUE END;
        ENDCASE;
    END;

  GetCurrentStateFromPSB: PUBLIC PROCEDURE RETURNS [SVPointer] =
    BEGIN OPEN SDDefs, ControlDefs, ProcessDefs;
    priority: Priority ← IF ValidatePSB[SREAD[CurrentPSB]]
      THEN GetProcessPriority[SREAD[CurrentPSB]] ELSE DefaultPriority;
    RETURN[LOOPHOLE[SREAD[SD+sFirstStateVector]+priority*SIZE[StateVector]]]
    END;

  GetProcessPriority: PUBLIC PROCEDURE [psb: ProcessDefs.ProcessHandle]
    RETURNS [ProcessDefs.Priority] =
    BEGIN OPEN ProcessDefs;
    localPSB: PSB;
    CopyRead[from: psb, to: @localPSB, nwords: SIZE[PSB]];
    RETURN[localPSB.priority]
    END;

  ValidatePSB: PUBLIC PROCEDURE [psb: ProcessDefs.ProcessHandle] RETURNS [BOOLEAN] =
    BEGIN OPEN ProcessDefs, SDDefs;
    fakePSB: INTEGER ← LOOPHOLE[psb, INTEGER];
    firstPSB: INTEGER ← SREAD[@SD[sFirstProcess]];
    IF fakePSB < firstPSB OR fakePSB > SREAD[@SD[sLastProcess]] THEN
      RETURN[FALSE];
    RETURN[(fakePSB-firstPSB) MOD SIZE[PSB] = 0]
    END;

  SetupLoadState: PROCEDURE [
    cfa: POINTER TO AltoFileDefs.CFA, pages: CARDINAL] =
    BEGIN OPEN SegmentDefs, LoadStateDefs;
    file: FileHandle;
    newseg: FileSegmentHandle;
    IF (newseg ← GetLoadState[]) # NIL THEN
      BEGIN
      UNTIL newseg.lock = 0 DO Unlock[newseg]; ENDLOOP;
      DeleteFileSegment[newseg];
      END;
    file ← InsertFile[@cfa.fp, Read+Write];
    BEGIN
    newseg ← FindLoadState[file, @cfa.fa, pages ! ANY => GOTO badCFA];
    EXITS
      badCFA => newseg ← NIL;
    END;
    SetLoadState[newseg];
    DebugContextDefs.InitBCD[];
    RETURN;
    END;

  FindLoadState: PROCEDURE [
    file: FileHandle, fa: POINTER TO AltoFileDefs.FA, pages: CARDINAL]
    RETURNS [seg: FileSegmentHandle] =
    BEGIN OPEN SegmentDefs;
    seg ← NewFileSegment[file, fa.page, pages, Read];
    SetFileSegmentDA[seg, fa.da ! UNWIND => DeleteFileSegment[seg]];
    SwapIn[seg ! UNWIND => DeleteFileSegment[seg]];
    Unlock[seg];
    RETURN
    END;

  ValidGlobalFrame: PUBLIC PROCEDURE [g: GlobalFrameHandle] RETURNS [BOOLEAN] =
```

```
    BEGIN OPEN ControlDefs;
    IF LOOPHOLE[g, ProcDesc].tag # frame THEN RETURN[FALSE];
    RETURN[MREAD[@GFT[ReadGlobalGFI[g]].frame] = g]
    END;

  ReverseEnumerateGlobalFrames: PUBLIC PROCEDURE [
    proc: PROCEDURE [GlobalFrameHandle] RETURNS [BOOLEAN]]
    RETURNS [GlobalFrameHandle] =
    BEGIN OPEN ControlDefs, SDDefs;
    gft: DESCRIPTOR FOR ARRAY OF GFTItem =
      DESCRIPTOR[GFT, MREAD[SD+sGFTLength]];
    i: GFTIndex;
    frame: GlobalFrameHandle;
    FOR i DECREASING IN [1 .. LENGTH[gft]) DO
      frame ← MREAD[@gft[i].frame];
      IF frame # NullGlobalFrame AND MREAD[@gft[i].epbase] = 0
      AND ValidGlobalFrame[frame] AND proc[frame]
        THEN RETURN [frame];
      ENDLOOP;
    RETURN [NullGlobalFrame]
    END;

  ReadGlobalGFI: PUBLIC PROCEDURE [frame: GlobalFrameHandle]
    RETURNS [ControlDefs.GFTIndex] =
    BEGIN OPEN ControlDefs;
    localFrame: GlobalFrame;
    CopyRead[from: frame, to: @localFrame, nwords: SIZE[GlobalFrame]];
    RETURN[localFrame.gfi]
    END;

  CodeSegment: PUBLIC PROCEDURE [f: FrameHandle]
    RETURNS [SegmentDefs.FileSegmentHandle] =
    BEGIN
    g: GlobalFrameHandle = MREAD[@f.accesslink];
    RETURN [MREAD[@g.codesegment]]
    END;

  CheckFrame: PUBLIC PROCEDURE [f: FrameHandle] RETURNS [BOOLEAN] =
    BEGIN OPEN ControlDefs;
    gf: GlobalFrameHandle;
    IF (LOOPHOLE[f,CARDINAL] MOD 4) # 0 THEN RETURN[FALSE];
    gf ← MREAD[@f.accesslink];
    RETURN[MREAD[@GFT[ReadGlobalGFI[gf]].frame] = gf]
    END;

  MakeProcedureDescriptor: PUBLIC PROCEDURE [ep: CARDINAL,
    gframe: GlobalFrameHandle] RETURNS [pd: ControlDefs.ProcDesc] =
    BEGIN OPEN ControlDefs, InlineDefs;
    base: CARDINAL;
    offset: EPIndex;
    index: GFTIndex;
    [base, offset] ← DIVMOD[ep, EPRange];
    pd.gfi ← index ← ReadGlobalGFI[gframe] + base;
    pd.ep ← offset; pd.tag ← procedure;
    RETURN
    END;

  LAMult: PROCEDURE [CARDINAL, CARDINAL] RETURNS [LA] =
    LOOPHOLE[InlineDefs.LongMult];

  LengthenPointer: PUBLIC PROCEDURE [p: POINTER] RETURNS [LONG POINTER] =
    BEGIN
    mds: LA = LAMult[DDptr.mdsContext, AltoDefs.PageSize];
    lp: LA ← [LA[low: LOOPHOLE[p], high: 0]];
    RETURN[IF p = NIL THEN NIL ELSE lp.lp+mds.li]
    END;

  UserWriteString: PUBLIC PROCEDURE[s: STRING] =
    BEGIN
    length: INTEGER ← MREAD[@s.length];
    UserWriteLongSubString[LengthenPointer[s], 0, length];
    RETURN
    END;

  UserWriteSubString: PUBLIC PROCEDURE[ss: StringDefs.SubString] =
    BEGIN
```

```
    UserWriteLongSubString[LengthenPointer[ss.base], ss.offset, ss.length];
    RETURN
    END;

  UserWriteLongString: PUBLIC PROCEDURE[s: LONG STRING] =
    BEGIN
    length: INTEGER ← LongREAD[@s.length];
    UserWriteLongSubString[s, 0, length];
    RETURN
    END;

  UserWriteLongSubString: PUBLIC PROCEDURE[ls: LONG STRING, offset, length: CARDINAL] =
    BEGIN
    i: CARDINAL;
    s: PACKED ARRAY [0..1] OF CHARACTER;
    p: POINTER = @s;
    IF offset MOD 2 # 0 THEN p↑ ← LongREAD[@ls.text+offset/2];
    FOR i IN [offset..offset+length) DO
      IF i MOD 2 = 0 THEN p↑ ← LongREAD[@ls.text+i/2];
      IODefs.WriteChar[s[i MOD 2]];
      ENDLOOP;
    RETURN
    END;


  UserCall: PUBLIC PROCEDURE [sp: SVPointer] RETURNS [p: SVPointer] =
    BEGIN OPEN CoreSwapDefs, ControlDefs;
    i: CARDINAL;
    sa: DESCRIPTOR FOR ARRAY OF UNSPECIFIED ← DESCRIPTOR[sp,SIZE[StateVector]];

    p ← @LOOPHOLE[DDptr.ESV.parameter, callDP].sv;
    FOR i IN [0..SIZE[StateVector]) DO
      MWRITE[p+i, sa[i]];
      ENDLOOP;
    DebugUtilityDefs.CoreSwap[call];
    p ← @LOOPHOLE[DDptr.ESV.parameter, callDP].sv;
    FOR i IN [0..SIZE[StateVector]) DO
      sa[i] ← MREAD[p+i];
      ENDLOOP;
    RETURN
    END;

  UserStart: PUBLIC PROCEDURE [f: GlobalFrameHandle] =
    BEGIN
    SWRITE[@LOOPHOLE[DDptr.ESV.parameter, CoreSwapDefs.startDP].frame,f];
    CoreSwap[start];
    RETURN
    END;

  END...
```