```
-- Miscellaneous.mesa; edited by Sandman, July 17, 1978  11:47 AM

DIRECTORY
  AllocDefs: FROM "allocdefs" USING [AllocInfo, MakeSwappedIn],
  AltoDefs: FROM "altodefs" USING [PageSize],
  BcdDefs: FROM "bcddefs" USING [VersionStamp],
  ControlDefs: FROM "controldefs" USING [
    FrameHandle, GFT, GFTIndex, GlobalFrameHandle, NullEpBase,
    NullGlobalFrame],
  FrameDefs: FROM "framedefs" USING [
    GlobalFrame, RemoveGlobalFrame, SwapInCode, SwapOutCode,
    ValidateGlobalFrame],
  ImageDefs: FROM "imagedefs" USING [
    AbortMesa, CleanupItem, CleanupMask, CleanupProcedure, ImageHeader,
    StopMesa],
  InlineDefs: FROM "inlinedefs" USING [BITAND, COPY],
  MiscDefs: FROM "miscdefs",
  Mopcodes: FROM "mopcodes" USING [zSTARTIO],
  NucleusDefs: FROM "nucleusdefs" USING [Resident],
  OsStaticDefs: FROM "osstaticdefs" USING [OsStatics],
  ProcessDefs: FROM "processdefs" USING [DisableInterrupts, EnableInterrupts],
  SDDefs: FROM "sddefs" USING [SD, sGoingAway],
  SegmentDefs: FROM "segmentdefs" USING [
    DefaultBase, DeleteFileSegment, FileHandle, FileSegmentAddress,
    FileSegmentHandle, NewFileSegment, Read, SwapIn, Unlock],
  TrapDefs: FROM "trapdefs";

DEFINITIONS FROM ControlDefs;

Miscellaneous: PROGRAM
  IMPORTS AllocDefs, FrameDefs, SegmentDefs, NucleusDefs
  EXPORTS FrameDefs, ImageDefs, MiscDefs, NucleusDefs, TrapDefs
  SHARES ControlDefs, ImageDefs = BEGIN

  DeletedFrame: PUBLIC PROCEDURE [gfi: GFTIndex] RETURNS [BOOLEAN] =
    BEGIN
    RETURN[GFT[gfi] = [frame: NullGlobalFrame, epbase: NullEpBase]];
    END;

  LockCode: PUBLIC PROCEDURE [link: UNSPECIFIED] =
    BEGIN
    FrameDefs.SwapInCode[FrameDefs.GlobalFrame[link]];
    RETURN
    END;

  UnlockCode: PUBLIC PROCEDURE [link: UNSPECIFIED] =
    BEGIN
    SegmentDefs.Unlock[FrameDefs.GlobalFrame[link].codesegment];
    RETURN
    END;

  CodeSegment: PUBLIC PROCEDURE [frame:FrameHandle]
    RETURNS [codeseg: SegmentDefs.FileSegmentHandle] =
    BEGIN
    FrameDefs.ValidateGlobalFrame[frame.accesslink];
    RETURN[frame.accesslink.codesegment]
    END;

  MakeCodeResident: PUBLIC PROCEDURE [f:GlobalFrameHandle] =
    BEGIN OPEN SegmentDefs, FrameDefs;
    seg: FileSegmentHandle = f.codesegment;
    info: AllocDefs.AllocInfo = [unused: 0, effort: hard, direction: bottomup,
      request: initial, class: code, swapunlocked: TRUE, compact: FALSE];
    ValidateGlobalFrame[f];
    IF f.codesegment.lock = 0 THEN SwapOutCode[f];
    AllocDefs.MakeSwappedIn[seg, DefaultBase, info];
    RETURN
    END;

  -- data shuffling

  SetBlock: PUBLIC PROCEDURE [p:POINTER, v:UNSPECIFIED, l:CARDINAL] =
    BEGIN
    IF l=0 THEN RETURN;  p↑ ← v;
    InlineDefs.COPY[from:p, to:p+1, nwords:l-1];
    END;
```

```
  -- Image Version

  ImageVersion: PUBLIC PROCEDURE RETURNS [version: BcdDefs.VersionStamp] =
    BEGIN OPEN ControlDefs, SegmentDefs;
    imagefile: FileHandle ←
      FrameDefs.GlobalFrame[NucleusDefs.Resident].codesegment.file;
    headerseg: FileSegmentHandle ← NewFileSegment[imagefile, 1, 1, Read];
    image: POINTER TO ImageDefs.ImageHeader;
    SwapIn[headerseg];
    image ← FileSegmentAddress[headerseg];
    version ← image.prefix.version;
    Unlock[headerseg];
    DeleteFileSegment[headerseg];
    RETURN
    END;


  -- Fake Modules

  DestroyFakeModule: PUBLIC PROCEDURE [f: GlobalFrameHandle]
    RETURNS [seg: SegmentDefs.FileSegmentHandle, offset: CARDINAL] =
    BEGIN
    seg ← f.codesegment;
    IF ~f.shared THEN seg.class ← other;
    FrameDefs.RemoveGlobalFrame[f];
    ProcessDefs.DisableInterrupts[];
    IF f.code.swappedout THEN
      BEGIN
      f.code.swappedout ← FALSE;
      offset ← f.code.offset;
      END
    ELSE offset ← f.code.offset - seg.VMpage*AltoDefs.PageSize;
    ProcessDefs.EnableInterrupts[];
    RETURN
    END;

  -- Get Network Number

wordsPerPup: INTEGER = 280;
Byte: TYPE = [0..255];

PupHeader:TYPE= MACHINE DEPENDENT RECORD [
  eDest, eSource: Byte,
  eWord2, pupLength: INTEGER,
  transportControl, pupType: Byte,
  pupID1, pupID2: INTEGER,
  destNet, destHost: Byte,
  destSocket1, destSocket2: INTEGER,
  sourceNet, sourceHost: Byte,
  sourceSocket1, sourceSocket2: INTEGER,
  xSum:CARDINAL];

Pup:TYPE= MACHINE DEPENDENT RECORD [
  head:PupHeader,
  junk:ARRAY [0..100] OF WORD];

EthernetDeviceBlock: TYPE = MACHINE DEPENDENT RECORD [
    EPLocMicrocodeStatus, EPLocHardwareStatus: Byte,
    EBLocInterruptBit: WORD,
    EELocInputFinishCount: INTEGER,
    ELLocCollisionMagic: WORD,
    EILocInputCount: INTEGER,
    EILocInputPointer: POINTER,
    EOLocOutputCount: INTEGER,
    EOLocOutputPointer: POINTER];

-- StartIO is Mesa bytecode used to control Ethernet interface
StartIO: PROCEDURE [WORD] = MACHINE CODE BEGIN Mopcodes.zSTARTIO END;
outputCommand: WORD = 1;
inputCommand: WORD = 2;
resetCommand: WORD = 3;

timer: POINTER TO INTEGER = LOOPHOLE[430B];
```

```
GetNetworkNumber: PUBLIC PROCEDURE RETURNS[CARDINAL] =
  BEGIN
  myHost: Byte ← OsStaticDefs.OsStatics.SerialNumber;
  then: INTEGER;
  now: INTEGER;
  device: POINTER TO EthernetDeviceBlock ← LOOPHOLE[600B];
  xpup: Pup;
  pup: POINTER TO Pup = @xpup;
  gatewayRequest:PupHeader ← [
    eDest: 0,                eSource: myHost,
    eWord2: 1000B,           pupLength: 22,
    transportControl: 0,     pupType: 200B,
    pupID1:,                 pupID2:,
    destNet: 0,              destHost: 0,
    destSocket1: 0,          destSocket2: 2,
    sourceNet: 0,            sourceHost: myHost,
    sourceSocket1: 0,        sourceSocket2: 2,
    xSum: 177777B];
  device.EBLocInterruptBit ← 0;
  StartIO[resetCommand];
  THROUGH [0..2) DO
    device↑ ← EthernetDeviceBlock[
      EPLocMicrocodeStatus: 0,
      EPLocHardwareStatus: 0,
      EBLocInterruptBit: 0,
      EELocInputFinishCount: 0,
      ELLocCollisionMagic: 0,
      EILocInputCount: 0,
      EILocInputPointer: pup,
      EOLocOutputCount: 13,
      EOLocOutputPointer: @gatewayRequest];
    StartIO[outputCommand];
    then ← timer↑;
    DO
      IF device.EPLocHardwareStatus#0 THEN
        BEGIN
        IF device.EPLocMicrocodeStatus = 0
          AND pup.head.eWord2 = 1000B
          AND wordsPerPup+2-device.EELocInputFinishCount > 13
          AND pup.head.destSocket1 = 0
          AND pup.head.destSocket2 = 2
          AND pup.head.pupType = 201B
          THEN RETURN[pup.head.sourceNet];
        device↑ ← EthernetDeviceBlock[
            EPLocMicrocodeStatus: 0,
            EPLocHardwareStatus: 0,
            EBLocInterruptBit: 0 ,
            EELocInputFinishCount: 0,
            ELLocCollisionMagic: 0,
            EILocInputCount: wordsPerPup+2,
            EILocInputPointer: pup,
            EOLocOutputCount: 0,
            EOLocOutputPointer: NIL];
        StartIO[inputCommand];
        END;
      now ← timer↑;
      IF now-then > 14 THEN EXIT;
      ENDLOOP;
    ENDLOOP;
  RETURN[0];
  END;

  -- procedure lists

  UserCleanupList: POINTER TO ImageDefs.CleanupItem ← NIL;

  AddCleanupProcedure: PUBLIC PROCEDURE [item: POINTER TO ImageDefs.CleanupItem] =
    BEGIN
    ProcessDefs.DisableInterrupts[];
    RemoveCleanupProcedure[item];
    item.link ← UserCleanupList;
    UserCleanupList ← item;
    ProcessDefs.EnableInterrupts[];
    END;

  RemoveCleanupProcedure: PUBLIC PROCEDURE [item: POINTER TO ImageDefs.CleanupItem] =
```

```
    BEGIN
    prev, this: POINTER TO ImageDefs.CleanupItem;
    IF UserCleanupList = NIL THEN RETURN;
    ProcessDefs.DisableInterrupts[];
    prev ← this ← UserCleanupList;
    IF this = item THEN UserCleanupList ← this.link
    ELSE UNTIL (this ← this.link) = NIL DO
      IF this = item THEN
        BEGIN prev.link ← this.link; EXIT END;
      prev ← this;
      ENDLOOP;
    ProcessDefs.EnableInterrupts[];
    END;

  UserCleanupProc: PUBLIC ImageDefs.CleanupProcedure =
    BEGIN -- all interrupts off if why = finish or abort
    this, next: POINTER TO ImageDefs.CleanupItem;
    this ← UserCleanupList;
    UserCleanupList ← NIL;
    WHILE this # NIL DO
      next ← this.link;
      IF InlineDefs.BITAND[ImageDefs.CleanupMask[why], this.mask] # 0 THEN
        this.proc[why !
          ANY => IF why = Abort OR why = Finish THEN CONTINUE];
      AddCleanupProcedure[this];
      this ← next;
      ENDLOOP;
    SELECT why FROM
      Finish => ImageDefs.StopMesa[];
      Abort => ImageDefs.AbortMesa[];
      ENDCASE;
    END;

-- Main Body;

  SDDefs.SD[SDDefs.sGoingAway] ← UserCleanupProc;

END...
```