```
-- MakeImageUtilities.Mesa  Edited by Sandman on September 13, 1977  5:32 PM

DIRECTORY
  AltoDefs: FROM "altodefs",
  AltoFileDefs: FROM "altofiledefs",
  BFSDefs: FROM "bfsdefs",
  ControlDefs: FROM "controldefs",
  DirectoryDefs: FROM "directorydefs",
  FrameDefs: FROM "framedefs",
  ImageDefs: FROM "imagedefs",
  InlineDefs: FROM "inlinedefs",
  LoadStateDefs: FROM "LoadStateDefs",
  MakeImageUtilDefs: FROM "makeimageutildefs",
  MiscDefs: FROM "miscdefs",
  ProcessDefs: FROM "processdefs",
  SegmentDefs: FROM "segmentdefs",
  StringDefs: FROM "stringdefs",
  SymDefs: FROM "symdefs",
  SystemDefs: FROM "systemdefs";

DEFINITIONS FROM ImageDefs, MakeImageUtilDefs;

MakeImageUtilities: PROGRAM
  IMPORTS DirectoryDefs, FrameDefs, LoadStateDefs, MiscDefs, SegmentDefs,
    StringDefs, SystemDefs
  EXPORTS ImageDefs, MakeImageUtilDefs
  SHARES ProcessDefs, SegmentDefs, ControlDefs, ImageDefs, MakeImageUtilDefs =

PUBLIC BEGIN


-- file requests

  RequestHead: POINTER TO FileRequest;

  InitFileRequest: PROCEDURE = BEGIN  RequestHead ← NIL; END;

  AddFileRequest: PROCEDURE [r: POINTER TO FileRequest] =
    BEGIN
    r.link ← RequestHead;
    RequestHead ← r;
    END;

  DropFileRequest: PROCEDURE [f: FileHandle] =
    BEGIN
    r: POINTER TO FileRequest;
    prev: POINTER TO FileRequest ← NIL;
    FOR r ← RequestHead, r.link UNTIL r = NIL DO
      BEGIN
      IF r.file = f THEN
        IF prev = NIL THEN RequestHead ← r.link
        ELSE prev.link ← r.link;
        EXIT;
        END;
      prev ← r;
      ENDLOOP;
    END;

  ProcessFileRequests: PROCEDURE =
    BEGIN OPEN AltoFileDefs;
    checkone: PROCEDURE [fp: POINTER TO FP, dname: STRING] RETURNS [BOOLEAN] =
      BEGIN
      ss: StringDefs.SubStringDescriptor ← [dname,0,dname.length];
      r: POINTER TO FileRequest;
      prev: POINTER TO FileRequest ← NIL;
      FOR r ← RequestHead, r.link UNTIL r = NIL DO
        IF (WITH r SELECT FROM
              long => StringDefs.EquivalentSubStrings[@ss,@name],
              short => StringDefs.EquivalentString[dname,name],
              ENDCASE => FALSE) THEN
          BEGIN
          IF r.file = NIL THEN r.file ← SegmentDefs.InsertFile[fp,r.access]
          ELSE r.file.fp ← fp↑;
          IF prev = NIL THEN RequestHead ← r.link
          ELSE prev.link ← r.link;
          END
```

```
        ELSE prev ← r;
        ENDLOOP;
      RETURN[RequestHead = NIL]
      END;

    DirectoryDefs.EnumerateDirectory[checkone];
    END;

-- symbol tables

RequestSymbolFiles: PROCEDURE =
  BEGIN
  next, head: POINTER TO FileRequest ← NIL;
  BuildRequests: PROCEDURE [s: FileSegmentHandle] RETURNS [BOOLEAN] =
    BEGIN
    r: POINTER TO shortFileRequest;
    p: POINTER TO FileRequest;
    IF s.class = symbols THEN
      FOR p ← head, p.link UNTIL p = NIL DO
        IF p.file = s.file THEN EXIT;
        REPEAT FINISHED =>
          BEGIN
          r ← GetSpace[SIZE[shortFileRequest]];
          r↑ ← FileRequest[,,,short[,]];
          r.file ← s.file;
          r.link ← head; head ← r;
          END;
        ENDLOOP;
    RETURN[FALSE];
    END;
  GetNames: PROCEDURE [fp: POINTER TO FP, s: STRING] RETURNS[BOOLEAN] =
    BEGIN
    r, prev: POINTER TO FileRequest;
    prev ← NIL;
    FOR r ← head, r.link UNTIL r = NIL DO
      IF fp.serial = r.file.fp.serial THEN
        BEGIN
        WITH r SELECT FROM
          short => name ← GetString[s];
          ENDCASE;
        IF prev = NIL THEN head ← r.link
        ELSE prev.link ← r.link;
        AddFileRequest[r];
        EXIT;
        END;
      prev ← r;
      ENDLOOP;
    RETURN[head = NIL];
    END;
  [] ← SegmentDefs.EnumerateFileSegments[BuildRequests];
  DirectoryDefs.EnumerateDirectory[GetNames];
  END;

-- bcd file names

GetBcdFileNames: PROCEDURE  [nbcds: ConfigIndex]
 RETURNS [names: DESCRIPTOR FOR ARRAY OF STRING] =
  BEGIN
  nfound: ConfigIndex ← 0;
  GetNames: PROCEDURE [fp: POINTER TO FP, s: STRING] RETURNS[BOOLEAN] =
    BEGIN
    FindBcd: PROCEDURE [config: ConfigIndex, bcd: LoadStateDefs.BcdAddress] RETURNS [BOOLEAN] =
      BEGIN
      IF fp↑ = bcd.fp THEN
        BEGIN
        names[config] ← GetString[s];
        nfound ← nfound + 1;
        RETURN[TRUE];
        END;
      RETURN[FALSE];
      END;
    [] ← LoadStateDefs.EnumerateLoadStateBcds[recentfirst, FindBcd];
    RETURN[nfound = nbcds];
    END;
  names ← DESCRIPTOR[GetSpace[nbcds], nbcds];
  MiscDefs.SetBlock[BASE[names], GetString["(anon)"], nbcds];
```

```
  DirectoryDefs.EnumerateDirectory[GetNames];
  RETURN[names];
  END;

-- space allocation

SpaceList: POINTER TO SpaceHeader ← NIL;
SpacePointer: POINTER TO SpaceHeader;
SpaceLeft: CARDINAL;

InitSpace: PROCEDURE = BEGIN SpaceLeft ← 0; END;

GetSpace: PROCEDURE [n: CARDINAL] RETURNS [p: POINTER] =
  BEGIN
  newseg: DataSegmentHandle;
  IF n > SpaceLeft THEN
    BEGIN
    newseg ← SegmentDefs.NewDataSegment[SegmentDefs.DefaultBase,1];
    SpacePointer ← SegmentDefs.DataSegmentAddress[newseg];
    SpacePointer.link ← SpaceList;
    SpacePointer.segment ← newseg;
    SpaceList ← SpacePointer;
    SpacePointer ← SpacePointer + SIZE[SpaceHeader];
    SpaceLeft ← AltoDefs.PageSize - SIZE[SpaceHeader];
    END;
  p ← SpacePointer;
  SpacePointer ← SpacePointer + n;
  SpaceLeft ← SpaceLeft - n;
  END;

GetString: PROCEDURE [oldstring: STRING] RETURNS [newstring: STRING] =
  BEGIN
  i, length: CARDINAL;
  string: TYPE = POINTER TO MACHINE DEPENDENT RECORD[length,maxlength: CARDINAL];
  length ← oldstring.length;
  newstring ← GetSpace[(length+5)/2];
  newstring.length ← length;
  LOOPHOLE[newstring, string].maxlength ← length;
  FOR i IN [0..length) DO newstring[i] ← oldstring[i]; ENDLOOP;
  RETURN;
  END;

FreeAllSpace: PROCEDURE =
  BEGIN
  next: POINTER TO SpaceHeader;
  UNTIL SpaceList = NIL DO
    next ← SpaceList.link;
    SegmentDefs.DeleteDataSegment[SpaceList.segment];
    SpaceList ← next;
    ENDLOOP;
  END;

-- image file management

LockCodeSegment: PROCEDURE [p: ProcDesc] =
  BEGIN OPEN ControlDefs;
  FrameDefs.LockCode[LOOPHOLE[p, ControlLink]];
  END;

UnlockCodeSegment: PROCEDURE [p: ProcDesc] =
  BEGIN OPEN ControlDefs;
  SegmentDefs.Unlock[LOOPHOLE[REGISTER[GFTreg],gftp]↑[p.gftindex].frame.codesegment];
  END;

KDSegment: PROCEDURE RETURNS [FileSegmentHandle] =
  BEGIN OPEN SegmentDefs;
  DiskKDFile: FileHandle = NewFile["DiskDescriptor", Read, DefaultVersion];
  FindKD: PROCEDURE [s: FileSegmentHandle] RETURNS [BOOLEAN] =
    BEGIN
    RETURN[s.file = DiskKDFile];
    END;
  RETURN[SegmentDefs.EnumerateFileSegments[FindKD]];
  END;

DAofPage: PROCEDURE [file: FileHandle, page: PageNumber] RETURNS [next: vDA] =
  BEGIN
```

```
  cfa: CFA;
  buf: POINTER = SystemDefs.AllocatePages[1];
  cfa.fp ← file.fp;
  cfa.fa ← AltoFileDefs.FA[file.fp.leaderDA,0,0];
  next ← SegmentDefs.JumpToPage[@cfa,page-1,buf].next;
  SystemDefs.FreePages[buf];
  RETURN
  END;

FillInCAs: PROCEDURE [Image: POINTER TO ImageHeader, mapindex: MapIndexType, ca: POINTER] =
  BEGIN
  i: CARDINAL;
  addr: POINTER;
  FOR i IN [0..mapindex) DO
    addr ← SegmentDefs.AddressFromPage[Image.map[i].page];
    THROUGH [0..Image.map[i].count) DO
      ca↑ ← addr;
      ca ← ca + 1;
      addr ← addr + AltoDefs.PageSize;
      ENDLOOP;
    ENDLOOP;
  END;

SwapOutUnlockedCode: PROCEDURE [f: GlobalFrameHandle] RETURNS [BOOLEAN] =
  BEGIN
  cseg: FileSegmentHandle ← f.codesegment;
  IF cseg.swappedin AND cseg.lock = 0 THEN FrameDefs.SwapOutCode[f];
  RETURN[FALSE]
  END;

SwapOutUnlocked: PROCEDURE [s: FileSegmentHandle] RETURNS [BOOLEAN] =
  BEGIN
  IF s.lock = 0 THEN SegmentDefs.SwapOut[s];
  RETURN[FALSE];
  END;

BashHint: PROCEDURE [s: FileSegmentHandle] RETURNS [BOOLEAN] =
  BEGIN
  s.hint ← SegmentDefs.FileHint[da:AltoFileDefs.eofDA, page:0];
  RETURN[FALSE];
  END;

BashFile: PROCEDURE [f: FileHandle] RETURNS [BOOLEAN] =
  BEGIN OPEN AltoFileDefs;
  f.open ← f.lengthvalid ← f.lengthchanged ← FALSE;
  IF f.fp.serial # SN[1,0,0,0,DirSN] THEN
    f.fp ← FP[SN[1,0,1,17777B,177777B],eofDA];
  RETURN[FALSE];
  END;

PatchUpGFT: PROCEDURE =
  BEGIN OPEN ControlDefs;
  i: GFTIndex;
  sd: POINTER TO ARRAY [0..1) OF GFTIndex = REGISTER[SDreg];
  GFT: POINTER TO ARRAY [0..1) OF GFTItem = REGISTER[GFTreg];
  FOR i IN [1..sd[sGFTLength]) DO
    IF GFT↑[i] = [frame: NULLFrame, epbase: 177777B] THEN
      GFT↑[i] ← [frame: NULLFrame, epbase: 0];
    ENDLOOP;
  RETURN
  END;

InitLoadStateGFT: PROCEDURE [initgft: LoadStateGFT, merge: BOOLEAN, nbcds: ConfigIndex] =
  BEGIN OPEN ControlDefs, LoadStateDefs;
  rgfi, cgfi: GFTIndex ← 0;
  i: ConfigIndex;
  sd: POINTER TO ARRAY [0..1) OF GFTIndex = REGISTER[SDreg];
  GFT: POINTER TO ARRAY [0..1) OF GFTItem = REGISTER[GFTreg];
  MiscDefs.SetBlock[
    p: BASE[initgft], v: ConfigGFI[config: ConfigNull, gfi: 0], l: sd[sGFTLength]];
  IF merge THEN
    FOR rgfi IN [1..sd[sGFTLength]) DO
      IF GFT↑[rgfi].frame # NULLFrame THEN initgft[rgfi] ←
        [config: 0, gfi: (cgfi←cgfi+1)];
      ENDLOOP
  ELSE
```

```
    FOR i IN [0..nbcds) DO
      cgfi ← 0;
      FOR rgfi IN [1..sd[sGFTLength]) DO
        IF GFT↑[rgfi].frame # NULLFrame AND
          LoadStateDefs.MapRealToConfig[rgfi].config = i
          THEN initgft[rgfi] ← [config: i, gfi: (cgfi←cgfi+1)];
        ENDLOOP;
      ENDLOOP;
  END;

NumberGFIInConfig: PROCEDURE [initgft: LoadStateGFT, con: ConfigIndex]
 RETURNS [ngfi: ControlDefs.GFTIndex] =
  BEGIN
  i: ControlDefs.GFTIndex;
  ngfi ← 0;
  FOR i IN [0..LENGTH[initgft]) DO
    IF initgft[i].config = con THEN ngfi ← ngfi + 1;
    ENDLOOP;
  RETURN
  END;

END.
```