```
-- File: BootImage.Mesa
-- Last edited by Sandman; May 10, 1978  1:41 PM

DIRECTORY
  AltoDefs: FROM "altodefs",
  AltoFileDefs: FROM "altofiledefs",
  BcdDefs: FROM "bcddefs",
  BootCacheDefs: FROM "bootcachedefs",
  BootmesaDefs: FROM "bootmesadefs",
  ControlDefs: FROM "controldefs",
  FakeSegDefs: FROM "fakesegdefs",
  FileLookupDefs: FROM "filelookupdefs",
  FrameDefs: FROM "framedefs",
  IODefs: FROM "iodefs",
  ImageDefs: FROM "imagedefs",
  InlineDefs: FROM "inlinedefs",
  MiscDefs: FROM "miscdefs",
  OsStaticDefs: FROM "osstaticdefs",
  SegmentDefs: FROM "segmentdefs",
  StreamDefs: FROM "streamdefs",
  StringDefs: FROM "stringdefs",
  SystemDefs: FROM "systemdefs",
  WartDefs: FROM "wartdefs";

DEFINITIONS FROM FakeSegDefs, AltoDefs, ControlDefs, WartDefs, BootmesaDefs;

BootImage: PROGRAM [data: POINTER TO BootData]
  IMPORTS BootCacheDefs, BootmesaDefs, FileLookupDefs, FrameDefs, ImageDefs,
    IODefs, MiscDefs, StreamDefs, SegmentDefs, StringDefs, FakeSegDefs, SystemDefs
  EXPORTS BootmesaDefs, FakeSegDefs
  SHARES ImageDefs, ControlDefs =
BEGIN

FileHandle: TYPE = SegmentDefs.FileHandle;
FileSegmentHandle: TYPE = SegmentDefs.FileSegmentHandle;
DataSegmentHandle: TYPE = SegmentDefs.DataSegmentHandle;


-- image file management

mapindex: CARDINAL ← 0;

MapOverflow: PUBLIC ERROR = CODE;

EnterMapItem: PROCEDURE [base: Address, pages: PageCount] =
  BEGIN
  map: POINTER TO ARRAY [0..0) OF normal ImageDefs.MapItem =
    LOOPHOLE[@data.image.map];
  IF mapindex > AltoDefs.PageSize - SIZE[ImageDefs.ImagePrefix] THEN
    BootmesaError["MapOverflow"L];
  IF pages > 127 THEN BootmesaError["Segment too big"L];
  map[mapindex] ← ImageDefs.MapItem[base/PageSize, pages, normal[]];
  mapindex ← mapindex + SIZE[normal ImageDefs.MapItem];
  RETURN
  END;

EnterMapSegment: PUBLIC PROCEDURE [s: FakeSegmentHandle] =
  BEGIN
  IF s.SwappedIn THEN EnterMapItem[s.VMaddress, s.Pages];
  RETURN
  END;

SegmentToMap: PUBLIC PROCEDURE [s: FakeSegmentHandle] =
  BEGIN OPEN IODefs;
  octal: NumberFormat = NumberFormat[8,FALSE,FALSE,1];
  base: NumberFormat = NumberFormat[8,FALSE,FALSE,4];
  pages: NumberFormat = NumberFormat[8,FALSE,FALSE,6];
  address: NumberFormat = NumberFormat[8,FALSE,TRUE,9];
  filename: STRING;

  s.ImageBase ← StreamDefs.GetIndex[data.imageStream].page+1;
  WriteNumber[s.ImageBase,base];
  WriteNumber[s.Pages,pages];
  WriteNumber[s.VMaddress,address];
  WriteString["  "L];
  filename ← IF s.File = data.vmFile THEN "VM"
```

```
    ELSE FileLookupDefs.GetFileName[IF s.TrueFile=NIL THEN s.File ELSE s.TrueFile];
  WriteString[filename];
  WriteString[" ["L];
  WriteNumber[s.Base,octal];
  WriteChar[',];
  WriteNumber[s.Pages,octal];
  WriteChar[']];
  IF s.Class = code THEN PrintModuleNames[s];
  WriteChar[CR];
  RETURN
  END;

WriteSwappedIn: PUBLIC PROCEDURE [s: FakeSegmentHandle] RETURNS [BOOLEAN] =
  BEGIN
  IF s.SwappedIn THEN WriteSegment[s];
  RETURN [FALSE]
  END;

WriteSwappedOut: PUBLIC PROCEDURE [s: FakeSegmentHandle] RETURNS [BOOLEAN] =
  BEGIN
  IF ~s.SwappedIn AND s.File = data.imageFile THEN WriteSegment[s];
  RETURN [FALSE]
  END;

WriteSegment: PROCEDURE [fs: FakeSegmentHandle] =
  BEGIN OPEN SegmentDefs;
  f: FileHandle ← fs.File;
  s: FileSegmentHandle;
  sp: POINTER TO Segment BootScriptEntry;

  IF fs = data.vmTableSeg THEN RETURN;
  SegmentToMap[fs];
  IF f = data.vmFile THEN WriteVMSegment[fs]
  ELSE
    BEGIN
    EnterMapSegment[fs];
    IF f = data.imageFile THEN
      BEGIN
      IF (f ← fs.TrueFile) = NIL THEN ERROR;
      fs.TrueFile ← NIL;
      sp ← GetSegmentBootLink[fs] + LOOPHOLE[data.tableBase];
      sp.base ← StreamDefs.GetIndex[data.imageStream].page+1;
      END;
    s ← NewFileSegment[f,fs.Base,fs.Pages,Read];
    SwapIn[s];
    IF StreamDefs.WriteBlock[data.imageStream, FileSegmentAddress[s],
      s.pages*PageSize] # CARDINAL[s.pages*PageSize] THEN ERROR;
    Unlock[s]; DeleteFileSegment[s];
    END;
  RETURN
  END;

WriteVMSegment: PROCEDURE [fs: FakeSegmentHandle] =
  BEGIN OPEN BootCacheDefs;
  p, basepage: PageNumber;
  pi: PageItem;
  segbase: Address;
  segcount: PageCount ← 0;
  getda: PROCEDURE RETURNS [AltoFileDefs.vDA] =
    BEGIN
    fa: AltoFileDefs.FA;
    StreamDefs.GetFA[data.imageStream,@fa];
    RETURN[fa.da]
    END;

  basepage ← fs.VMaddress/PageSize;
  FOR p IN[basepage..basepage+fs.Pages) DO
    IF segcount = 0 THEN segbase ← p*PageSize;
    IF GetPageItem[p].p.page = 0 THEN
      BEGIN
      IF segcount # 0 THEN
        BEGIN
        EnterMapItem[segbase,segcount];
        segcount ← 0;
        END;
      END
    ELSE
      BEGIN
      segcount ← segcount + 1;
      pi ← PageItem[StreamDefs.GetIndex[data.imageStream].page+1,getda[]];
      IF StreamDefs.WriteBlock[data.imageStream,
        SegmentDefs.FileSegmentAddress[GetCS[GetPageItem[p]]],
          PageSize] # PageSize THEN ERROR;
      SetPageItem[p, pi];
      END;
    ENDLOOP;
  IF segcount # 0 THEN EnterMapItem[segbase,segcount];
  RETURN
  END;

MarkImageSegment: PROCEDURE [fs: FakeSegmentHandle] RETURNS [BOOLEAN] =
  BEGIN
  f: FileHandle ← fs.File;
  IF f # data.vmFile AND
   (fs.SwappedIn OR fs.Class = code OR fs.CopyToImage) THEN
    BEGIN
    IF fs.TrueFile # NIL THEN ERROR;
    fs.TrueFile ← f;
    fs.File ← data.imageFile;
    data.imageFile.segcount ← data.imageFile.segcount + 1;
    IF fs.SwappedIn THEN data.imageFile.swapcount ← data.imageFile.swapcount + 1;
    END;
  RETURN [FALSE]
  END;

InitializeImage: PUBLIC PROCEDURE =
  BEGIN OPEN ImageDefs, StreamDefs, SegmentDefs, IODefs;
  name: STRING ← [40];
  time: AltoFileDefs.TIME ← MiscDefs.DAYTIME[];

  StringDefs.AppendString[name,data.imageFileRoot];
  StringDefs.AppendString[name,".Image"L];
  data.imageFile ← NewFile[name,Read+Write+Append, DefaultVersion];
  data.imageStream ← CreateWordStream[data.imageFile,Write+Append];
  SetIndex[data.imageStream, StreamIndex[FirstImageDataPage-1,0]];
  CleanupDiskStream[data.imageStream];
  SwapIn[data.headerSeg←NewFileSegment[data.imageFile,1,HeaderPages,Write]];
  data.image ← FileSegmentAddress[data.headerSeg];
  MiscDefs.Zero[data.image, HeaderPages*AltoDefs.PageSize];
  data.image.prefix.versionident ← ImageDefs.VersionID;
  data.image.prefix.options ← 0;
  data.image.prefix.type ← bootmesa;
  data.image.prefix.leaderDA ← AltoFileDefs.eofDA;
  data.image.prefix.creator ← ImageDefs.ImageVersion[];

  InitializeHeap[];

  [] ← FakeEnumerateSegments[MarkImageSegment];
  WriteChar[CR];
  WriteLine["  Image"L];
  WriteLine["Base Pages  Address  Source [base,pages]"L];
  RETURN
  END;

DeclareSegments: PUBLIC PROCEDURE =
  BEGIN OPEN FakeSegDefs;
  [] ← FakeEnumerateSegments[DeclareSegment];
  [] ← FrameDefs.EnumerateGlobalFrames[DeclareCodeLink];
  [] ← FakeEnumerateSegments[FindModuleNames];
  WriteScriptSegments[];
  RETURN
  END;

PrintModuleNames: PROCEDURE [seg: FakeSegmentHandle] =
  BEGIN
  s: SegHandle;
  name: NameList;
  FOR s ← segs, s.next UNTIL s = NIL DO
    IF s.seg = seg THEN EXIT;
    REPEAT
      FINISHED => BootmesaError["Messed up segment names"L];
    ENDLOOP;
  FOR name ← s.modules, name.next UNTIL name = NIL DO
    IODefs.WriteChar[' '];
    IODefs.WriteString[name.name];
    ENDLOOP;
  RETURN
  END;

SegmentItem: TYPE = RECORD [
  next: SegHandle,
  seg: FakeSegmentHandle,
  modules: NameList];

SegHandle: TYPE = POINTER TO SegmentItem;

NameItem: TYPE = RECORD [
  next: NameList,
  name: STRING];

NameList: TYPE = POINTER TO NameItem;

segs: SegHandle ← NIL;

FindModuleNames: PROCEDURE [seg: FakeSegmentHandle] RETURNS [BOOLEAN] =
  BEGIN OPEN SystemDefs;
  name: STRING ← [60];
  s: SegHandle;
  ni: NameList;
  FindFrames: PROCEDURE [frame: GlobalFrameHandle] RETURNS [BOOLEAN] =
    BEGIN
    IF BootCacheDefs.READ[@frame.codesegment] # seg THEN RETURN[FALSE];
    ModuleName[frame, name];
    ni ← AllocateHeapNode[SIZE[NameItem]];
    ni↑ ← [s.modules, AllocateHeapString[name.length]];
    s.modules ← ni;
    StringDefs.AppendString[ni.name, name];
    name.length ← 0;
    RETURN[FALSE]
    END;
  IF seg.Class # code THEN RETURN[FALSE];
  s ← AllocateHeapNode[SIZE[SegmentItem]];
  s↑ ← SegmentItem[segs, seg, NIL];
  segs ← s;
  [] ← FrameDefs.EnumerateGlobalFrames[FindFrames];
  RETURN[FALSE]
  END;


END...
```