```
-- File: BootScript.Mesa
-- Last edited by Sandman; May 10, 1978  1:40 PM

DIRECTORY
  AltoDefs: FROM "altodefs",
  AltoFileDefs: FROM "altofiledefs",
  BootCacheDefs: FROM "BootCacheDefs",
  BootmesaDefs: FROM "bootmesadefs",
  ControlDefs: FROM "controldefs",
  FakeSegDefs: FROM "fakesegdefs",
  InlineDefs: FROM "inlinedefs",
  SegmentDefs: FROM "segmentdefs",
  StreamDefs: FROM "streamdefs",
  WartDefs: FROM "wartdefs";

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

BootScript: PROGRAM [data: POINTER TO BootData]
  IMPORTS BootCacheDefs, BootmesaDefs, StreamDefs, SegmentDefs, FakeSegDefs
  EXPORTS BootmesaDefs, FakeSegDefs
  SHARES ControlDefs =
BEGIN

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



-- Wart Script Management

TableSeg: DataSegmentHandle;    -- segment for the script table
TableFileSegment: FileSegmentHandle ← NIL;
TableIndex: BootIndex;          -- index of next free slot
TableSize: CARDINAL;            -- size of segment
VMtablebase: Address;           -- location of table in user space

GetBootIndex: PUBLIC PROCEDURE RETURNS [BootIndex] =
  BEGIN RETURN[TableIndex]; END;

GetBootTable: PUBLIC PROCEDURE RETURNS [CARDINAL] =
  BEGIN RETURN[LOOPHOLE[data.tableBase]]; END;

InitializeBootScript: PUBLIC PROCEDURE =
  BEGIN OPEN SegmentDefs;
  TableSeg ← NewDataSegment[DefaultBase, 1];
  TableIndex ← LOOPHOLE[0];
  data.tableBase ← DataSegmentAddress[TableSeg];
  TableSize ← PageSize*TableSeg.pages;
  data.bsheader ← @BSHeader;
  RETURN
  END;

BSHeader: BootScriptHeader;

FinishBootScript: PUBLIC PROCEDURE [startframe: GlobalFrameHandle] RETURNS [headerloc: Address] =
  BEGIN
  r: Stop BootScriptEntry ← BootScriptEntry[Stop[]];
  BSHeader.user ← UserControl[];
  BSHeader.nub ← NubFrame[];
  BSHeader.av ← data.AVbase;
  BSHeader.gft ← data.GFTbase;
  BSHeader.sd ← data.SDbase;
  BSHeader.tablebase ← VMtablebase;
  AppendBootWords[@r, SIZE[Stop BootScriptEntry]];
  headerloc ← LOOPHOLE[TableIndex,CARDINAL] + VMtablebase;
  AppendBootWords[@BSHeader, SIZE[BootScriptHeader]];
  SegmentDefs.Unlock[TableFileSegment];
  SegmentDefs.DeleteFileSegment[TableFileSegment];
  RETURN
  END;

DeclareVMBounds: PUBLIC PROCEDURE [b: VMBounds] =
  BEGIN
  BSHeader.ffvmp ← b.ffvmp;
  BSHeader.lfvmp ← b.lfvmp;
```

```
  END;

BootTableOverflow: ERROR = CODE;
ExpandingAllowed: BOOLEAN ← TRUE;

ExpandBootTable: PROCEDURE =
  BEGIN OPEN SegmentDefs;
  newseg: DataSegmentHandle;
  newbase: POINTER;

  IF ~ExpandingAllowed THEN BootmesaError["BootTableOverflow"L];
  newseg ← NewDataSegment[DefaultBase, TableSeg.pages+1];
  newbase ← DataSegmentAddress[newseg];
  InlineDefs.COPY[
    from:data.tableBase, to:newbase, nwords:TableSeg.pages*PageSize];
  DeleteDataSegment[TableSeg];
  TableSeg ← newseg;  TableSize ← TableSeg.pages*PageSize;
  data.tableBase ← newbase;
  RETURN
  END;

AppendBootWords: PROCEDURE [addr: POINTER, count: CARDINAL] =
  BEGIN
  WHILE LOOPHOLE[TableIndex,CARDINAL] + count > TableSize DO
    ExpandBootTable[];
    ENDLOOP;
  IF count = 0 THEN ERROR;
  InlineDefs.COPY[
    from:addr, to:data.tableBase+LOOPHOLE[TableIndex, CARDINAL], nwords:count];
  TableIndex ← TableIndex + count;
  RETURN
  END;

DeclareBootCommand: PUBLIC PROCEDURE [c: WartDefs.BootCommand] =
  BEGIN
  r: Command BootScriptEntry ←
    BootScriptEntry[Command[command: c, fill:]];
  AppendBootWords[@r, SIZE[Command BootScriptEntry]];
  RETURN
  END;

DeclareSwapOut: PUBLIC PROCEDURE [f: GlobalFrameHandle] =
  BEGIN
  r: SwapOutCode BootScriptEntry ←
    BootScriptEntry[SwapOutCode[frame: f, fill:]];
  AppendBootWords[@r, SIZE[SwapOutCode BootScriptEntry]];
  RETURN
  END;

DeclareOpenFiles: PUBLIC PROCEDURE =
  BEGIN
  r: OpenFile BootScriptEntry ← BootScriptEntry[OpenFile[fill:]];
  AppendBootWords[@r, SIZE[OpenFile BootScriptEntry]];
  END;

NILZ: POINTER = LOOPHOLE[0];

PagePointer: TYPE = MACHINE DEPENDENT RECORD [page, byte: [0..256)];

DeclareSegment: PUBLIC PROCEDURE [s: FakeSegmentHandle] RETURNS [BOOLEAN] =
  BEGIN
  r: Segment BootScriptEntry ←
    BootScriptEntry[Segment[
      fill1:,
      data: s.File = data.vmFile,
      access: s.Access,
      base: s.Base,
      pages: s.Pages,
      vmpage: 0,
      handle: NILZ]];
  IF s.SwappedIn THEN r.vmpage ← LOOPHOLE[s.VMaddress, PagePointer].page;
  SetSegmentBootLink[s, LOOPHOLE[TableIndex, SegmentBootIndex]];
  AppendBootWords[@r, SIZE[Segment BootScriptEntry]];
  RETURN [FALSE]
  END;
```

```
DeclareCodeLink: PUBLIC PROCEDURE [f: GlobalFrameHandle] RETURNS [BOOLEAN] =
  BEGIN
  cseg: FakeSegmentHandle = BootCacheDefs.READ[@f.codesegment];
  r: CodeLink BootScriptEntry ← BootScriptEntry[CodeLink[
    frame:f,
    codeseg: GetSegmentBootLink[cseg]]];
  AppendBootWords[@r, SIZE[CodeLink BootScriptEntry]];
  RETURN [FALSE]
  END;

DeclareUnlock: PUBLIC PROCEDURE [s: FakeSegmentHandle] =
  BEGIN
  r: Unlock BootScriptEntry;
  i: SegmentBootIndex;
  IF (i←GetSegmentBootLink[s]) # NullBootIndex
    THEN
      BEGIN
      r ← BootScriptEntry[Unlock[seg: i]];
      AppendBootWords[@r, SIZE[Unlock BootScriptEntry]];
      END;
  RETURN
  END;

UnlockFrame: PUBLIC PROCEDURE [f: GlobalFrameHandle] =
  BEGIN
  DeclareUnlock[BootCacheDefs.READ[@f.codesegment]]
  END;

WriteScriptSegments: PUBLIC PROCEDURE =
  BEGIN OPEN StreamDefs, SegmentDefs;
  base: PageNumber;
  pages: PageCount;
  tseg: FakeSegmentHandle;
  freespace: CARDINAL ← NumberSwappedIn[]*SIZE[Unlock BootScriptEntry] + 16
    + SIZE[WartDefs.BootScriptHeader];
  WHILE TableSize < freespace + LOOPHOLE[TableIndex,CARDINAL] DO
    ExpandBootTable[];
    ENDLOOP;
  pages ← TableSeg.pages;
  data.vmTableSeg ← tseg ←
    FakeNewSegment[DefaultFile, GetVMBounds[].ffvmp, pages, Read+Write];
  SegmentToMap[tseg];
  VMtablebase ← tseg.VMaddress;
  [] ← DeclareSegment[tseg];
  EnterMapSegment[tseg];
  base ← GetIndex[data.imageStream].page+1;
  IF WriteBlock[data.imageStream,data.tableBase,pages*PageSize] #
    CARDINAL[pages*PageSize] THEN ERROR;
  CleanupDiskStream[data.imageStream];
  DeleteDataSegment[TableSeg]; TableSeg ← NIL;
  TableFileSegment ← NewFileSegment[data.imageFile,base,pages,Read+Write];
  SwapIn[TableFileSegment];
  data.tableBase ← FileSegmentAddress[TableFileSegment];
  ExpandingAllowed ← FALSE;
  CleanupDiskStream[data.imageStream];
  RETURN
  END;

END...
```