```
-- BootLoader.mesa
-- Last Modified by Sandman,  August 14, 1978  10:33 AM

DIRECTORY
  AltoDefs: FROM "altodefs",
  AltoFileDefs: FROM "altofiledefs",
  BcdDefs: FROM "bcddefs",
  BootCacheDefs: FROM "bootcachedefs",
  BootmesaDefs: FROM "bootmesadefs",
  ControlDefs: FROM "controldefs",
  FakeSegDefs: FROM "fakesegdefs",
  FrameDefs: FROM "framedefs",
  InlineDefs: FROM "inlinedefs",
  IODefs: FROM "iodefs",
  LoaderBcdUtilDefs: FROM "loaderbcdutildefs",
  LoaderUtilityDefs: FROM "loaderutilitydefs",
  LoadStateDefs: FROM "loadstatedefs",
  SegmentDefs: FROM "segmentdefs",
  StreamDefs: FROM "streamdefs",
  StringDefs: FROM "stringdefs",
  SystemDefs: FROM "systemdefs";

DEFINITIONS FROM ControlDefs, FakeSegDefs, SegmentDefs, BcdDefs;

BootLoader: PROGRAM
  IMPORTS BootCacheDefs, BootmesaDefs, FakeSegDefs, IODefs, LoaderBcdUtilDefs,
    LoaderUtilityDefs, SegmentDefs, StreamDefs, StringDefs, SystemDefs
  EXPORTS BootmesaDefs, FrameDefs =

  PUBLIC BEGIN

  BcdBase: TYPE = POINTER TO BcdDefs.BCD;
  FakeSegmentHandle: TYPE = FakeSegDefs.FakeSegmentHandle;
  GlobalFrameHandle: TYPE = ControlDefs.GlobalFrameHandle;
  GFTIndex: TYPE = ControlDefs.GFTIndex;

  Load: PROCEDURE [name: STRING] RETURNS [lsseg, initlsseg, fakebcdseg: FakeSegmentHandle] =
    BEGIN OPEN ControlDefs, SegmentDefs, FakeSegDefs;
    lslength: CARDINAL;
    swatee: FileHandle;
    bcdseg: FileSegmentHandle ← LoadBcd[NewFile[name, Read, OldFileOnly]
      ! InvalidBcd =>
        BEGIN OPEN IODefs;
        WriteString["Invalid file "L]; WriteString[name];
        SIGNAL BootmesaDefs.BootAbort;
        END];
    fakebcdseg ← FakeNewSegment[bcdseg.file, bcdseg.base, bcdseg.pages, Read];
    fakebcdseg.Link2 ← bcdseg;
    fakebcdseg.CopyToImage ← TRUE;
    lslength ←
      SystemDefs.PagesForWords[LoadStateDefs.BcdArrayLength + LENGTH[gft]];
    swatee ← SegmentDefs.NewFile["swatee", Read+Write+Append, DefaultVersion];
    lsseg ← FakeNewSegment[swatee, DefaultBase, lslength, Read+Write];
    lsseg.CopyToImage ← TRUE;
    initlsseg ←
      FakeNewSegment[swatee, lsseg.Base+lslength, lslength, Read+Write];
    initlsseg.CopyToImage ← TRUE;
    New[bcdseg];
    RETURN
    END;

  InvalidBcd: SIGNAL [file: FileHandle] = CODE;

  LoadBcd: PROCEDURE [bcdfile: FileHandle] RETURNS [bcdseg: FileSegmentHandle] =
    BEGIN OPEN SegmentDefs;
    pages: AltoDefs.PageCount;
    bcd: BcdBase;
    bcdseg ← NewFileSegment[bcdfile, 1, 1, Read];
    SwapIn[bcdseg];
    bcd ← FileSegmentAddress[bcdseg];
    IF (pages ← bcd.nPages) # 1 THEN
      BEGIN
      Unlock[bcdseg];
      MoveFileSegment[bcdseg, 1, pages];
      SwapIn[bcdseg];
      bcd ← FileSegmentAddress[bcdseg];
      END;
    BEGIN
      ENABLE UNWIND =>
        BEGIN
        Unlock[bcdseg];
        DeleteFileSegment[bcdseg];
        END;
      IF bcd.versionident # BcdDefs.VersionID OR bcd.definitions THEN
        ERROR InvalidBcd[bcdfile];
    END; -- OF OPEN
    Unlock[bcdseg];
    END;

  New: PROCEDURE [bcdseg: FileSegmentHandle] =
    BEGIN OPEN SegmentDefs, LoaderUtilityDefs;
    loadee: BcdBase;
    SwapIn[bcdseg];
    loadee ← FileSegmentAddress[bcdseg];
    InitializeUtilities[loadee];
    ModuleTable ← DESCRIPTOR[SystemDefs.AllocateSegment[
      loadee.nModules*SIZE[ModuleInfo]], loadee.nModules];
    nModulesEntered ← 0;
    GetCodeFileNames[loadee];
    LookupFileTable[ ! FileNotFound =>
      BEGIN OPEN IODefs;
      WriteString["Cant find a code file "L];
      WriteString[name];
      SIGNAL BootmesaDefs.BootAbort;
      END];
    AssignFrameAddresses[loadee];
    RelocateOnly[loadee];
    FinalizeUtilities[];
    SystemDefs.FreeSegment[BASE[ModuleTable]];
    Unlock[bcdseg];
    END;

  FirstFrameWord: TYPE = MACHINE DEPENDENT RECORD [
    gfi: GFTIndex,
    unused: [0..3],
    alloced, shared, started: BOOLEAN,
    trapxfers, codelinks: BOOLEAN];

  GetCodeFileNames: PROCEDURE [loadee: BcdBase] =
    BEGIN
    SegSearch: PROCEDURE [sgh: SGHandle, sgi: SGIndex] RETURNS [BOOLEAN] =
      BEGIN
      IF sgh.class = code THEN LoaderUtilityDefs.AddFileName[sgh.file];
      RETURN[FALSE]
      END;
    [] ← LoaderBcdUtilDefs.EnumerateSegTable[loadee, SegSearch];
    RETURN
    END;

  RelocateOnly: PROCEDURE [loadee: BcdBase]=
    BEGIN
    ModuleSearch: PROCEDURE [mth: MTHandle, mti: MTIndex] RETURNS [BOOLEAN] =
      BEGIN
      i: CARDINAL;
      frame: GlobalFrameHandle ← BootCacheDefs.READ[@gft[mth.gfi].frame];
      codeseg: FakeSegDefs.FakeSegmentHandle;
      codesegment: SegmentDefs.FileSegmentHandle;
      fw: FirstFrameWord ← BootCacheDefs.READ[frame];
      codelinks: BOOLEAN ← fw.codelinks;
      linkbase: POINTER TO ControlLink;
      link: ControlLink;
      IF mth.frame.length = 0 THEN
        BEGIN
        ModuleIsBound[mth];
        RETURN[FALSE];
        END;
      IF codelinks THEN
        BEGIN OPEN SegmentDefs;
        IF IsModuleBound[mth] THEN RETURN[FALSE];
        codeseg ← BootCacheDefs.READ[@frame.codesegment];
        codesegment ← codeseg.Link2;
        SwapIn[codesegment];
        linkbase ← FileSegmentAddress[codesegment]+mth.code.offset;
        END
      ELSE linkbase ← LOOPHOLE[frame];
      linkbase ← linkbase - mth.frame.length;
      FOR i IN [0..mth.frame.length) DO
        link ← mth.frame.frag[i];
        SELECT link.tag FROM
          procedure =>
            BEGIN
            IF link.gfi >= loadee.firstdummy THEN link ← UnboundLink;
            IF codelinks THEN (linkbase+i)↑ ← link
            ELSE BootCacheDefs.WRITE[linkbase+i, link];
            END;
          frame =>
            BEGIN
            link ← IF link.gfi >= loadee.firstdummy THEN NullLink
              ELSE BootCacheDefs.READ[@gft[link.gfi].frame];
            IF codelinks THEN (linkbase+i)↑ ← link
            ELSE BootCacheDefs.WRITE[linkbase+i, link];
            END;
          ENDCASE => BootCacheDefs.WRITE[linkbase+i, UnboundLink];
        ENDLOOP;
      ModuleIsBound[mth];
      IF codelinks THEN
        BEGIN
        SegmentDefs.Unlock[codesegment];
        codesegment.write ← TRUE;
        SegmentDefs.SwapUp[codesegment];
        codesegment.write ← FALSE;
        END;
      RETURN [FALSE];
      END;

    [] ← LoaderBcdUtilDefs.EnumerateModuleTable[loadee, ModuleSearch];
    RETURN
    END;

  AssignFrameAddresses: PROCEDURE [loadee: BcdBase] =
    BEGIN
    ssb: BcdDefs.NameString ← LOOPHOLE[loadee+loadee.ssOffset];
    ModuleSearch: PROCEDURE [mth: MTHandle, mti: MTIndex] RETURNS [BOOLEAN] =
      BEGIN OPEN SegmentDefs;
      gfi: GFTIndex;
      i: CARDINAL;
      frame: GlobalFrameHandle;
      codeseg: FakeSegmentHandle;
      framelinks, shared: BOOLEAN;
      gf: GlobalFrame;
      FindSharedModules: PROCEDURE [smth: MTHandle, smti: MTIndex]
        RETURNS [BOOLEAN] =
        BEGIN
        RETURN[smth # mth AND smth.code.sgi = mth.code.sgi]
        END;

      framelinks ← mth.links = frame OR ~mth.code.linkspace;
      frame ← BootmesaDefs.AllocGlobalFrame[
        mth.framesize, mth.frame.length, framelinks];
      gfi ← EnterGlobalFrame[frame, mth.ngfi];
      IF gfi # mth.gfi THEN
        BEGIN OPEN IODefs;
        WriteString["Invalid bcd"L];
        SIGNAL BootmesaDefs.BootAbort;
        END;
      codeseg ← FindCodeSegment[loadee, mth, frame];
      codeseg.Class ← code;
      codeseg.Link ← NIL;
      shared ← LoaderBcdUtilDefs.EnumerateModuleTable[loadee,
        FindSharedModules].mth # NIL;
      gf ← [gfi: gfi, unused: 0, copied: FALSE, alloced: FALSE,
        shared: shared, started: FALSE, trapxfers: FALSE,
        codelinks: ~framelinks, code: [out[mth.code.offset]],
        codesegment: LOOPHOLE[codeseg], global:];
      gf.code.swappedout ← TRUE;
      CopyWrite[from: @gf, to: frame, size: SIZE[GlobalFrame]];
      IF Loadmap # NIL THEN
        BEGIN OPEN IODefs;
        WriteString["New: g = "L];
        WriteNumber[frame,NumberFormat[8,FALSE,TRUE,6]];
        WriteChar[SP];
        FOR i IN [mth.name .. mth.name+ssb.size[mth.name]) DO
          WriteChar[ssb.string.text[i]];
          ENDLOOP;
        WriteChar[CR];
        END;
      RETURN[FALSE];
      END;

    [] ← LoaderBcdUtilDefs.EnumerateModuleTable[loadee, ModuleSearch];
    AssignControlModules[loadee];
    END;

  CopyWrite: PROCEDURE [from, to: POINTER, size: CARDINAL] =
    BEGIN
    i: CARDINAL;
    FOR i IN [0..size) DO BootCacheDefs.WRITE[to+i, (from+i)↑]; ENDLOOP;
    RETURN
    END;

  FindCodeSegment: PUBLIC PROCEDURE [
    loadee: BcdBase, mth: MTHandle, frame: GlobalFrameHandle]
    RETURNS [seg: FakeSegmentHandle] =
    BEGIN OPEN SegmentDefs;
    file: FileHandle;
    sgh: SGHandle ← mth.code.sgi+LOOPHOLE[loadee+loadee.sgOffset, CARDINAL];
    i: CARDINAL;
    pages: CARDINAL;
    FindSegment: PROCEDURE [fs: FakeSegmentHandle] RETURNS [BOOLEAN] =
      BEGIN
      RETURN[fs.File = file AND fs.Base = sgh.base AND fs.Pages = pages];
      END;
    FOR i IN [0..nModulesEntered) DO
      IF ModuleTable[i].mth.code.sgi = mth.code.sgi THEN
        RETURN[BootCacheDefs.READ[@ModuleTable[i].frame.codesegment]];
      ENDLOOP;
    file ← LoaderUtilityDefs.FileHandleFromTable[sgh.file];
    pages ← sgh.pages+sgh.extraPages;
    seg ← FakeEnumerateSegments[FindSegment];
    IF seg = NIL THEN
      BEGIN
      seg ← FakeNewSegment[file, sgh.base, pages, Read];
      seg.Link2 ← NewFileSegment[file, sgh.base, pages, Read];
      END;
    ModuleTable[nModulesEntered] ←
      ModuleInfo[mth, frame, FALSE, mth.code.sgi];
    nModulesEntered ← nModulesEntered + 1;
    RETURN
    END;

  ModuleInfo: TYPE = RECORD [
    mth: MTHandle,
    frame: GlobalFrameHandle,
    bound: BOOLEAN,
    sgi: SGIndex];

  ModuleTable: DESCRIPTOR FOR ARRAY OF ModuleInfo;
  nModulesEntered: CARDINAL;

  ModuleIsBound: PUBLIC PROCEDURE [mth: MTHandle] =
    BEGIN
    i: CARDINAL;
    FOR i IN [0..nModulesEntered) DO
      IF ModuleTable[i].mth = mth THEN ModuleTable[i].bound ← TRUE;
      ENDLOOP;
    RETURN
    END;

  IsModuleBound: PUBLIC PROCEDURE [mth: MTHandle] RETURNS [BOOLEAN] =
    BEGIN
    i: CARDINAL;
    FOR i IN [0..nModulesEntered) DO
      IF ModuleTable[i].mth = mth AND ModuleTable[i].bound THEN RETURN[TRUE];
      ENDLOOP;
    RETURN[FALSE];
    END;

  AssignControlModules: PUBLIC PROCEDURE [loadee: BcdBase] =
    BEGIN OPEN ControlDefs;
    ctb: CARDINAL ← LOOPHOLE[loadee+loadee.ctOffset];
    mtb: CARDINAL ← LOOPHOLE[loadee+loadee.mtOffset];
    ModuleSearch: PROCEDURE [mth: MTHandle, mti: MTIndex] RETURNS [BOOLEAN] =
      BEGIN OPEN ControlDefs;
      frame: GlobalFrameHandle ← BootCacheDefs.READ[@gft[mth.gfi].frame];
      cti: CTIndex;
      gfi: GFTIndex;
      ControlGfi: PROCEDURE [cti: CTIndex] RETURNS [GFTIndex] =
        BEGIN
        RETURN[IF cti = CTNull OR (ctb+cti).control = MTNull THEN GFTNull
          ELSE (mtb+(ctb+cti).control).gfi];
        END;
      gfi ← ControlGfi[cti ← mth.config];
      WHILE gfi = mth.gfi DO gfi ← ControlGfi[cti ← (ctb+cti).config] ENDLOOP;
      BootCacheDefs.WRITE[@frame.global[0], (IF gfi = GFTNull
        THEN NullGlobalFrame ELSE BootCacheDefs.READ[@gft[gfi].frame])];
      RETURN [FALSE];
      END;

    [] ← LoaderBcdUtilDefs.EnumerateModuleTable[loadee, ModuleSearch];
    END;

-- global frame table management

  gft: PRIVATE DESCRIPTOR FOR ARRAY OF GFTItem;

  InitializeGFT: PROCEDURE [p: POINTER, l: CARDINAL] =
    BEGIN
    gft ← DESCRIPTOR[p, l];
    RETURN
    END;

  EnumerateGlobalFrames: PROCEDURE [
      proc: PROCEDURE [GlobalFrameHandle] RETURNS [BOOLEAN]] RETURNS [GlobalFrameHandle] =
    BEGIN
    i: GFTIndex;
    frame: GlobalFrameHandle;
    FOR i IN [1 .. LENGTH[gft])
      DO
      frame ← BootCacheDefs.READ[@gft[i].frame];
      IF frame # NullGlobalFrame AND BootCacheDefs.READ[@gft[i].epbase] = 0
        AND proc[frame] THEN RETURN [frame];
      ENDLOOP;
    RETURN [NullGlobalFrame]
    END;

  gftrover: PRIVATE CARDINAL ← 0;

  NoGlobalFrameSlots: SIGNAL [CARDINAL] = CODE;

  EnterGlobalFrame: PROCEDURE [frame: GlobalFrameHandle, nslots: CARDINAL]
    RETURNS [entryindex: GFTIndex] =
    BEGIN
    i, imax, n, epoffset: CARDINAL;
    i ← gftrover;  imax ← LENGTH[gft] - nslots;  n ← 0;
      DO
      IF (i ← IF i>=imax THEN 1 ELSE i+1) = gftrover THEN
        BEGIN OPEN IODefs;
        WriteString["GFT Full"L];
        SIGNAL BootmesaDefs.BootAbort;
        END;
      IF BootCacheDefs.READ[@gft[i].frame] # NullGlobalFrame
        THEN n ← 0
        ELSE
          IF (n←n+1)=nslots THEN EXIT;
      ENDLOOP;
    entryindex ← (gftrover←i)-nslots+1;  epoffset ← 0;
    FOR i IN [entryindex..gftrover]
      DO
      BootCacheDefs.WRITE[@gft[i].frame, frame];
      BootCacheDefs.WRITE[@gft[i].epbase, epoffset];
      epoffset ← epoffset + EPRange;
      ENDLOOP;
    RETURN
    END;

-- loadmap management

  Loadmap: PRIVATE StreamDefs.StreamHandle ← NIL;
  DisplayChar: PRIVATE PROCEDURE [StreamDefs.StreamHandle, CHARACTER];

  OpenLoadmap: PROCEDURE [root: STRING] =
    BEGIN OPEN StringDefs;
    default: StreamDefs.DisplayHandle ← StreamDefs.GetDefaultDisplayStream[];
    name: STRING ← [40];

    AppendString[name,root];
    AppendString[name,".Loadmap"L];
    Loadmap ← StreamDefs.NewByteStream[name, Write+Append];
    DisplayChar ← default.put;
    default.put ← LMput;
    RETURN
    END;

  CloseLoadmap: PROCEDURE =
    BEGIN
    default: StreamDefs.DisplayHandle ← StreamDefs.GetDefaultDisplayStream[];

    IF default.put # LMput THEN ERROR;
    default.put ← DisplayChar;
    Loadmap.destroy[Loadmap];
    RETURN
    END;

  LMput: PRIVATE PROCEDURE [s: StreamDefs.StreamHandle, c: CHARACTER] =
    BEGIN
    DisplayChar[s, c];
    Loadmap.put[Loadmap, c];
    RETURN
    END;


  END....
```