```
-- MakeImage.Mesa
-- Edited by:
--              Sandman on July 17, 1978  11:48 AM

DIRECTORY
  AllocDefs: FROM "allocdefs" USING [
    AddSwapStrategy, RemoveSwapStrategy, SwappingProcedure, SwapStrategy],
  AltoDefs: FROM "altodefs" USING [
    BytesPerPage, PageCount, PageNumber, PageSize],
  AltoFileDefs: FROM "altofiledefs" USING [
    CFA, CFP, eofDA, fillinDA, FP, TIME, vDA],
  BcdDefs: FROM "bcddefs" USING [FTSelf, MTHandle, MTIndex, VersionStamp],
  BcdMergeDefs: FROM "bcdmergedefs" USING [MergeBcd],
  BcdTabDefs: FROM "bcdtabdefs" USING [FindString],
  BFSDefs: FROM "bfsdefs" USING [ActOnPages, GetNextDA],
  BootDefs: FROM "bootdefs" USING [PositionSeg],
  ControlDefs: FROM "controldefs" USING [
    Alloc, AllocationVector, AllocationVectorSize, ATPreg, AV, ControlLink,
    EntryVectorItem, FrameHandle, FrameVec, Free, GetReturnLink, GFT,
    GFTIndex, GlobalFrameHandle, Greg, Lreg, MaxAllocSlot, OTPreg, ProcDesc,
    SD, StateVector, SVPointer, WDCreg, XTSreg],
  CoreSwapDefs: FROM "coreswapdefs" USING [SetLevel],
  DirectoryDefs: FROM "directorydefs" USING [EnumerateDirectory],
  DiskDefs: FROM "diskdefs" USING [
    DA, DiskPageDesc, DiskRequest, ResetDisk, SwapPages, VirtualDA],
  DiskKDDefs: FROM "diskkddefs" USING [CloseDiskKD, InitializeDiskKD],
  FrameDefs: FROM "framedefs" USING [GlobalFrame, SwapOutCode],
  ImageDefs: FROM "imagedefs" USING [
    FileRequest, FirstImageDataPage, HeaderPages, ImageHeader, ImageVersion,
    MapItem, MapSpace, PuntMesa, StopMesa, UserCleanupProc, VersionID],
  InlineDefs: FROM "inlinedefs" USING [BITAND, COPY],
  LoaderBcdUtilDefs: FROM "loaderbcdutildefs" USING [
    BcdBase, EnumerateModuleTable],
  LoadStateDefs: FROM "loadstatedefs" USING [
    BcdAddress, BcdSegFromLoadState, ConfigIndex, EnumerateLoadStateBcds,
    GetInitialLoadState, GetLoadState, GFTIndex, InputLoadState, LoadState,
    LoadStateGFT, ReleaseLoadState, UpdateLoadState],
  MiscDefs: FROM "miscdefs" USING [DAYTIME, GetNetworkNumber, SetBlock, Zero],
  MIUtilityDefs: FROM "miutilitydefs" USING [
    AddFileRequest, BashFile, BashHint, DAofPage, DropFileRequest, FillInCAs,
    FreeAllSpace, GetBcdFileNames, GetSpace, InitFileRequest,
    InitLoadStateGFT, InitSpace, KDSegment, LockCodeSegment, MergeABcd,
    MergeAllBcds, NewBcdSegmentFromStream, PatchUpGFT, ProcessFileRequests,
    UnlockCodeSegment],
  OsStaticDefs: FROM "osstaticdefs" USING [OsStatics],
  ProcessDefs: FROM "processdefs" USING [
    ActiveWord, CurrentPSB, CurrentState, CV, DisableInterrupts, DIW,
    EnableInterrupts, ProcessHandle, Queue, ReadyList, SDC, WakeupsWaiting],
  SDDefs: FROM "sddefs" USING [
    sAddFileRequest, sAllocTrap, sGFTLength, sGoingAway, sSwapTrap,
    sXferTrap],
  SegmentDefs: FROM "segmentdefs" USING [
    AddressFromPage, Append, CloseFile, DataSegmentAddress, DataSegmentHandle,
    DefaultBase, DefaultVersion, DeleteDataSegment, DeleteFileSegment,
    EnumerateDataSegments, EnumerateFiles, EnumerateFileSegments, FileError,
    FileHandle, FileHint, FileSegmentAddress, FileSegmentHandle,
    MapFileSegment, NewDataSegment, NewFile, PageFromAddress, Read,
    ReleaseFile, SetEndOfFile, SwapIn, SwapOut, SwapUp, Unlock, Write],
  StreamDefs: FROM "streamdefs" USING [
    CreateWordStream, DiskHandle, NewWordStream, ReadBlock, StreamHandle,
    WriteBlock],
  StringDefs: FROM "stringdefs" USING [EquivalentString],
  SystemDefs: FROM "systemdefs" USING [PruneHeap],
  TableDefs: FROM "tabledefs" USING [Allocate],
  TimeDefs: FROM "timedefs" USING [PackedTime];

DEFINITIONS FROM
  LoadStateDefs, DiskDefs, ImageDefs, ControlDefs, SegmentDefs, MIUtilityDefs;

MakeImage: PROGRAM
  IMPORTS AllocDefs, BcdMergeDefs, BcdTabDefs, TableDefs, BFSDefs, BootDefs, CoreSwapDefs,
    DirectoryDefs, DiskDefs, DiskKDDefs, FrameDefs, ImageDefs, LoaderBcdUtilDefs,
    LoadStateDefs, MiscDefs, SegmentDefs, StreamDefs, StringDefs, SystemDefs,
    MIUtilityDefs
  EXPORTS ImageDefs, MIUtilityDefs
  SHARES ProcessDefs, DiskDefs, SegmentDefs, ControlDefs, ImageDefs =
```

```
  BEGIN

  CFA: TYPE = AltoFileDefs.CFA;
  DataSegmentHandle: TYPE = SegmentDefs.DataSegmentHandle;
  FP: TYPE = AltoFileDefs.FP;
  FileHandle: TYPE = SegmentDefs.FileHandle;
  FileSegmentHandle: TYPE = SegmentDefs.FileSegmentHandle;
  PageCount: TYPE = AltoDefs.PageCount;
  PageNumber: TYPE = AltoDefs.PageNumber;
  shortFileRequest: TYPE = short ImageDefs.FileRequest;
  vDA: TYPE = AltoFileDefs.vDA;
  GlobalFrameHandle: TYPE = ControlDefs.GlobalFrameHandle;
  LoadStateGFT: TYPE = LoadStateDefs.LoadStateGFT;
  ConfigIndex: TYPE = LoadStateDefs.ConfigIndex;
  StreamHandle: TYPE = StreamDefs.StreamHandle;
  ProcDesc: TYPE = ControlDefs.ProcDesc;

MoveWords: PUBLIC PROCEDURE [source: POINTER, nwords: CARDINAL] =
  BEGIN
  IF nwords # StreamDefs.WriteBlock[stream: bcdstream, address: source, words: nwords]
    THEN ERROR;
  END;

MapSegmentsInBcd: PROCEDURE [
initialGFT: LoadStateGFT, config: ConfigIndex, bcdseg: FileSegmentHandle]
  RETURNS [unresolved, exports: BOOLEAN] =
  BEGIN OPEN LoaderBcdUtilDefs, LoadStateDefs;
  bcd: BcdBase;
  sgb: CARDINAL;
  MapSegments: PROCEDURE [mth: BcdDefs.MTHandle, mti: BcdDefs.MTIndex]
    RETURNS [BOOLEAN] =
    BEGIN OPEN s: sgb+mth.code.sgi;
    gftLength: CARDINAL = SD[SDDefs.sGFTLength];
    frame: GlobalFrameHandle;
    rgfi: GFTIndex ← 1;
    WHILE rgfi < gftLength DO
      IF initialGFT[rgfi] = [config: config, gfi: mth.gfi] THEN EXIT;
      rgfi ← rgfi + 1;
    ENDLOOP;
    IF s.file = BcdDefs.FTSelf AND s.class = code THEN
      BEGIN
      frame ← GFT[rgfi].frame;
      s.base ← frame.codesegment.base;
      END;
    RETURN[FALSE];
    END;
  SegmentDefs.SwapIn[bcdseg];
  bcd ← SegmentDefs.FileSegmentAddress[bcdseg];
  sgb ← LOOPHOLE[bcd+bcd.sgOffset];
  [] ← EnumerateModuleTable[bcd, MapSegments];
  unresolved ← bcd.nImports # 0;
  exports ← bcd.nExports # 0;
  SegmentDefs.Unlock[bcdseg];
  SegmentDefs.SwapOut[bcdseg];
  END;

bcdstream: StreamDefs.DiskHandle;

DisplayHeader: POINTER TO WORD = LOOPHOLE[420B];

SwapTrapDuringMakeImage: PUBLIC SIGNAL = CODE;
SwapErrorDuringMakeImage: PUBLIC SIGNAL = CODE;
SwapOutDuringMakeImage: PUBLIC SIGNAL = CODE;
NoRoomInImageMap: PUBLIC SIGNAL = CODE;

SwapTrapError: PROCEDURE =
  BEGIN
  dest: ControlDefs.ControlLink;
  s: ControlDefs.StateVector;
  ProcessDefs.DisableInterrupts[];
  s ← STATE;
  dest ← LOOPHOLE[REGISTER[ControlDefs.OTPreg]];
  ProcessDefs.DisableInterrupts[];
  SIGNAL SwapTrapDuringMakeImage;
  RETURN WITH s;
  END;
```

```
SwapOutError: AllocDefs.SwappingProcedure =
  BEGIN
  SIGNAL SwapOutDuringMakeImage;
  RETURN[TRUE];
  END;

-- File Segment Transfer Routines

bufferseg: DataSegmentHandle;
buffer: POINTER;
BufferPages: PageCount;

SwapDR: TYPE = POINTER TO swap DiskRequest;

TransferPages: PROCEDURE [
da: vDA, base: PageNumber, pages: PageCount, fp: POINTER TO FP, sdr: SwapDR]
  RETURNS [next: vDA] =
  BEGIN OPEN DiskDefs;
  sdr.da ← @da;
  sdr.firstPage ← base;
  sdr.lastPage ← base+pages-1;
  sdr.fp ← fp;
  IF SwapPages[sdr].page # base+pages-1 THEN SIGNAL SwapErrorDuringMakeImage;
  next ← sdr.desc.next;
  RETURN[next];
  END;

TransferFileSegment: PROCEDURE [
buffer: POINTER, seg: FileSegmentHandle, file: FileHandle, base: PageNumber, fileda: vDA]
  RETURNS [vDA] =
  BEGIN
  dpd: DiskPageDesc;
  sdr: swap DiskRequest;
  old: FileHandle ← seg.file;
  segbase: PageNumber ← seg.base;
  pages: PageCount ← seg.pages;
  segda: vDA;
  WITH s: seg SELECT FROM
    disk => segda ← s.hint.da;
    ENDCASE => ERROR SwapErrorDuringMakeImage;
  seg.base ← base;
  sdr ← [ca: buffer, da:, firstPage:, lastPage:, fp:, fixedCA: FALSE, action:,
    lastAction:, signalCheckError: FALSE, option: swap[desc: @dpd]];
  IF seg.swappedin THEN
    BEGIN
    sdr.ca ← SegmentDefs.AddressFromPage[seg.VMpage];
    sdr.action ← sdr.lastAction ← WriteD;
    fileda ← TransferPages[fileda, base, pages, @file.fp, @sdr];
    old.swapcount ← old.swapcount - 1;
    file.swapcount ← file.swapcount + 1;
    END
  ELSE
    BEGIN
    WHILE BufferPages < pages DO
      pages ← pages - BufferPages;
      sdr.action ← sdr.lastAction ← ReadD;
      segda ← TransferPages[segda, segbase, BufferPages, @old.fp, @sdr];
      sdr.action ← sdr.lastAction ← WriteD;
      fileda ← TransferPages[fileda, base, BufferPages, @file.fp, @sdr];
      segbase ← segbase + BufferPages;
      base ← base + BufferPages;
      ENDLOOP;
    sdr.action ← sdr.lastAction ← ReadD;
    segda ← TransferPages[segda, segbase, pages, @old.fp, @sdr];
    sdr.action ← sdr.lastAction ← WriteD;
    fileda ← TransferPages[fileda, base, pages, @file.fp, @sdr];
    END;
  old.segcount ← old.segcount - 1;
  seg.file ← file;
  WITH s: seg SELECT FROM
    disk => s.hint ← FileHint[AltoFileDefs.eofDA, 0];
    ENDCASE;
  file.segcount ← file.segcount + 1;
  IF old.segcount = 0 THEN ReleaseFile[old];
  RETURN [fileda];
```

```
  END;

EnumerateNeededModules: PROCEDURE [proc: PROCEDURE [ProcDesc]] =
  BEGIN
  proc[LOOPHOLE[EnumerateNeededModules]];
  proc[LOOPHOLE[MIUtilityDefs.AddFileRequest]];
  proc[LOOPHOLE[BFSDefs.ActOnPages]];
  proc[LOOPHOLE[SegmentDefs.MapFileSegment]];
  proc[LOOPHOLE[SegmentDefs.CloseFile]];
  proc[LOOPHOLE[DiskKDDefs.CloseDiskKD]];
  proc[LOOPHOLE[ImageDefs.UserCleanupProc]];
  proc[LOOPHOLE[DirectoryDefs.EnumerateDirectory]];
  proc[LOOPHOLE[StreamDefs.ReadBlock]];
  proc[LOOPHOLE[StreamDefs.CreateWordStream]];
  proc[LOOPHOLE[StringDefs.EquivalentString]];
  proc[LOOPHOLE[LoadStateDefs.InputLoadState]];
  proc[LOOPHOLE[FrameDefs.GlobalFrame]];
  proc[LOOPHOLE[LoaderBcdUtilDefs.EnumerateModuleTable]];
  proc[LOOPHOLE[SystemDefs.PruneHeap]];
  END;

SwapOutMakeImageCode: PROCEDURE =
  BEGIN OPEN FrameDefs;
  SwapOutCode[GlobalFrame[MIUtilityDefs.AddFileRequest]];
  SwapOutCode[GlobalFrame[TableDefs.Allocate]];
  SwapOutCode[GlobalFrame[BcdTabDefs.FindString]];
  SwapOutCode[GlobalFrame[LoaderBcdUtilDefs.EnumerateModuleTable]];
  SwapOutCode[GlobalFrame[LoadStateDefs.InputLoadState]];
  SwapOutCode[GlobalFrame[BcdMergeDefs.MergeBcd]];
  END;

InvalidImageName: PUBLIC SIGNAL = CODE;

ResidentGFI: CARDINAL = 1;

GetImageFile: PROCEDURE [name: STRING] RETURNS [file: FileHandle] =
  BEGIN OPEN SegmentDefs;
  file ← NewFile[name, Read+Write+Append, DefaultVersion];
  IF file = GFT[ResidentGFI].frame.codesegment.file THEN
    SIGNAL InvalidImageName;
  RETURN
  END;
```

```
InstallImage: PROCEDURE [name: STRING, merge, code: BOOLEAN] =
  BEGIN OPEN DiskDefs, AltoFileDefs;
  wdc: CARDINAL;
  diskrequest: DiskRequest;
  lpn: PageNumber; numChars: CARDINAL;
  savealloctrap, saveswaptrap: ControlLink;
  auxtrapFrame: FrameHandle;
  saveAllocationVector: AllocationVector;
  saveXferTrap, saveXferTrapStatus: UNSPECIFIED;
  nextpage: PageNumber;
  swappedinfilepages, swappedoutfilepages, datapages: PageCount ← 0;
  SwapOutErrorStrategy: AllocDefs.SwapStrategy ←
    AllocDefs.SwapStrategy[link:,proc:SwapOutError];
  mapindex: CARDINAL ← 0;
  maxFileSegPages: CARDINAL ← 0;
  endofdatamapindex: CARDINAL;
  ptSeg: DataSegmentHandle;
  HeaderSeg: DataSegmentHandle;
  Image: POINTER TO ImageHeader;
  imageDA, HeaderDA: vDA;
  ImageFile: FileHandle;
  diskKD: FileSegmentHandle;
  saveDIW: WORD;
  savePV: ARRAY [0..15] OF UNSPECIFIED;
  saveSDC: WORD;
  saveReadyList: ProcessDefs.Queue;
  saveCurrentPSB: ProcessDefs.ProcessHandle;
  saveCurrentState: ControlDefs.SVPointer;
  page: PageNumber;
  maxbcdsize: CARDINAL ← 0;
  bcdnames: DESCRIPTOR FOR ARRAY OF STRING;
  bcds: DESCRIPTOR FOR ARRAY OF FileSegmentHandle;
  unresolved, exports: BOOLEAN;
  con, nbcds: ConfigIndex;
  time: AltoFileDefs.TIME;
  initgft: LoadStateGFT;
  initstateseg: FileSegmentHandle ← LoadStateDefs.GetInitialLoadState[];
  stateseg: FileSegmentHandle ← LoadStateDefs.GetLoadState[];
  initloadstate: LoadStateDefs.LoadState;
  net: CARDINAL ← MiscDefs.GetNetworkNumber[];
  NullFP: AltoFileDefs.FP = [[1,0,1,17777B,177777B], AltoFileDefs.eofDA];

  SaveProcesses: PROCEDURE =
    BEGIN OPEN ProcessDefs;
    saveDIW ← DIW↑;
    savePV ← CV↑;
    DIW↑ ← 2;
    WakeupsWaiting↑ ← 0;
    saveSDC ← SDC↑;
    saveReadyList ← ReadyList↑;
    saveCurrentPSB ← CurrentPSB↑;
    saveCurrentState ← CurrentState↑;
    END;
  RestoreProcesses: PROCEDURE =
    BEGIN OPEN ProcessDefs;
    ActiveWord↑ ← 77777B;
    DIW↑ ← saveDIW;
    CV↑ ← savePV;
    SDC↑ ← saveSDC;
    ReadyList↑ ← saveReadyList;
    CurrentPSB↑ ← saveCurrentPSB;
    CurrentState↑ ← saveCurrentState;
    END;
  EnterMapItem: PROCEDURE [vmpage: PageNumber, pages: PageCount] =
    BEGIN
    map: POINTER TO ARRAY [0..0) OF normal MapItem = LOOPHOLE[@Image.map];
    IF pages > 127 THEN SIGNAL SwapErrorDuringMakeImage;
    IF mapindex >= MapSpace THEN SIGNAL NoRoomInImageMap;
    map[mapindex] ← MapItem[vmpage, pages, normal[]];
    mapindex ← mapindex + SIZE[normal MapItem];
    END;
  CountFileSegments: PROCEDURE [s: FileSegmentHandle] RETURNS [BOOLEAN] =
    BEGIN
    IF s # diskKD THEN
      BEGIN
```

```
      [] ← BootDefs.PositionSeg[s, FALSE];
      IF s.swappedin THEN
        BEGIN
        swappedinfilepages ← swappedinfilepages + s.pages;
        IF s.class=code THEN
          maxFileSegPages ← MAX[maxFileSegPages, s.pages];
        END
      ELSE
        BEGIN
        swappedoutfilepages ← swappedoutfilepages + s.pages;
        maxFileSegPages ← MAX[maxFileSegPages, s.pages];
        END
      END;
    RETURN[FALSE];
    END;
  CountDataSegments: PROCEDURE [s: DataSegmentHandle] RETURNS [BOOLEAN] =
    BEGIN
    IF s # bufferseg THEN datapages ← datapages + s.pages;
    RETURN[FALSE];
    END;
  MapDataSegments: PROCEDURE [s: DataSegmentHandle] RETURNS [BOOLEAN] =
    BEGIN
    IF s # HeaderSeg AND s # bufferseg THEN
      BEGIN
      EnterMapItem[s.VMpage, s.pages];
      nextpage ← nextpage + s.pages;
      END;
    RETURN[FALSE];
    END;
  WriteSwappedIn: PROCEDURE [s: FileSegmentHandle] RETURNS [BOOLEAN] =
    BEGIN
    IF s.swappedin THEN
      BEGIN
      imageDA ← TransferFileSegment[buffer, s, ImageFile, nextpage, imageDA];
      EnterMapItem[s.VMpage, s.pages];
      nextpage ← nextpage + s.pages;
      END;
    RETURN[FALSE];
    END;
  WriteSwappedOutCode: PROCEDURE [s: FileSegmentHandle] RETURNS [BOOLEAN] =
    BEGIN
    IF ~s.swappedin AND s.class = code THEN
      BEGIN
      imageDA ← TransferFileSegment[buffer, s, ImageFile, nextpage, imageDA];
      nextpage ← nextpage + s.pages;
      END;
    RETURN[FALSE];
    END;
  WriteSwappedOutNonCode: PROCEDURE [s: FileSegmentHandle] RETURNS [BOOLEAN] =
    BEGIN
    IF ~s.swappedin AND s.class # code AND s # diskKD THEN
      BEGIN
      imageDA ← TransferFileSegment[buffer, s, ImageFile, nextpage, imageDA];
      nextpage ← nextpage + s.pages;
      END;
    RETURN[FALSE];
    END;
  SaveBcd: PROCEDURE [config: ConfigIndex, addr: BcdAddress] RETURNS [BOOLEAN] =
    BEGIN
    bcds[config] ← LoadStateDefs.BcdSegFromLoadState[config];
    RETURN [FALSE];
    END;

  SD[SDDefs.sAddFileRequest] ← AddFileRequest;
  ImageFile ← GetImageFile[name];
  diskKD ← KDSegment[];
  ProcessDefs.DisableInterrupts[];
  wdc ← REGISTER[WDCreg];
  CoreSwapDefs.SetLevel[-1];
  SaveProcesses[];
  ImageDefs.UserCleanupProc[Save];

-- handle bcds

  SwapIn[initstateseg];
  initloadstate ← FileSegmentAddress[initstateseg];
```

```
  MiscDefs.Zero[initloadstate, initstateseg.pages*AltoDefs.PageSize];
  initgft ← DESCRIPTOR[@initloadstate.gft, SD[SDDefs.sGFTLength]];
  bcdstream ← StreamDefs.NewWordStream["makeimage.scratch$", Read+Write+Append];
  nbcds ← LoadStateDefs.InputLoadState[];  -- bring it in for first time
  bcdnames ← GetBcdFileNames[nbcds];
  nbcds ← IF merge THEN 1 ELSE nbcds;
  bcds ← DESCRIPTOR[GetSpace[nbcds], nbcds];
  page ← 0;
  InitLoadStateGFT[initgft, merge, nbcds];
  IF merge THEN
    BEGIN OPEN MIUtilityDefs;
    MergeAllBcds[initgft, code, bcdnames];
    [page, bcds[0]] ← NewBcdSegmentFromStream[bcdstream, page];
    maxbcdsize ← bcds[0].pages;
    END
  ELSE
    BEGIN OPEN MIUtilityDefs;
    [] ← LoadStateDefs.EnumerateLoadStateBcds[recentlast, SaveBcd];
    FOR con IN [0..nbcds) DO
      MergeABcd[con, initgft, code, bcdnames];
      [page, bcds[con]] ← NewBcdSegmentFromStream[bcdstream, page];
      maxbcdsize ← MAX[maxbcdsize, bcds[con].pages];
      ENDLOOP;
    END;
  bcdstream.destroy[bcdstream];
  IF merge THEN PatchUpGFT[];

  [] ← SystemDefs.PruneHeap[];

  SetupAuxStorage[];
  EnumerateNeededModules[LockCodeSegment];
  HeaderDA ← DAofPage[ImageFile, 1];
  -- [] ← FrameDefs.EnumerateGlobalFrames[SwapOutUnlockedCode];
  -- [] ← EnumerateFileSegments[SwapOutUnlocked];

  -- set up private frame allocation trap
  ControlDefs.Free[ControlDefs.Alloc[0]]; -- flush large frames
  savealloctrap ← SD[SDDefs.sAllocTrap];
  SD[SDDefs.sAllocTrap] ← auxtrapFrame ← auxtrap[];
  saveAllocationVector ← AV↑;
  AV↑ ← LOOPHOLE[DataSegmentAddress[AuxSeg], POINTER TO AllocationVector]↑;

  BufferPages ← maxbcdsize+initstateseg.pages;
  bufferseg ← NewDataSegment[DefaultBase, BufferPages];
  [] ← EnumerateDataSegments[CountDataSegments];
  swappedinfilepages ← swappedoutfilepages ← 0;
  [] ← EnumerateFileSegments[CountFileSegments];
  SetEndOfFile[ImageFile,
    datapages+swappedinfilepages+swappedoutfilepages+FirstImageDataPage-1,
    AltoDefs.BytesPerPage];
  [] ← DiskKDDefs.CloseDiskKD[];

  HeaderSeg ← NewDataSegment[DefaultBase, 1];
  Image ← DataSegmentAddress[HeaderSeg];
  MiscDefs.Zero[Image,ImageDefs.HeaderPages*AltoDefs.PageSize];
  Image.prefix.versionident ← ImageDefs.VersionID;
--Image.prefix.options ← 0;
--Image.prefix.state.stk[0] ← Image.prefix.state.stk[1] ← 0;
  Image.prefix.state.stkptr ← 2;
  Image.prefix.state.dest ← REGISTER[Lreg];
  Image.prefix.type ← makeimage;
  Image.prefix.leaderDA ← ImageFile.fp.leaderDA;
  time ← MiscDefs.DAYTIME[];
  Image.prefix.version ← BcdDefs.VersionStamp[
    time: TimeDefs.PackedTime[lowbits: time.low, highbits: time.high],
    zapped: FALSE,
    net: net,
    host: OsStaticDefs.OsStatics.SerialNumber];
  Image.prefix.creator ← ImageDefs.ImageVersion[]; -- version stamp of currently running image

  nextpage ← FirstImageDataPage;
  [] ← SegmentDefs.EnumerateDataSegments[MapDataSegments];
  IF nextpage # FirstImageDataPage+datapages THEN ERROR;
  endofdatamapindex ← mapindex;

  -- now disable swapping
```

```
  saveswaptrap ← SD[SDDefs.sSwapTrap];
  SD[SDDefs.sSwapTrap] ← SwapTrapError;
  AllocDefs.AddSwapStrategy[@SwapOutErrorStrategy];
  imageDA ← DAofPage[ImageFile, nextpage];
  buffer ← SegmentDefs.DataSegmentAddress[bufferseg];
  [] ← SegmentDefs.EnumerateFileSegments[WriteSwappedIn];
  IF nextpage # FirstImageDataPage+datapages+swappedinfilepages THEN ERROR;
  [] ← SegmentDefs.EnumerateFileSegments[WriteSwappedOutCode];
  [] ← SegmentDefs.EnumerateFileSegments[WriteSwappedOutNonCode];
  SegmentDefs.DeleteDataSegment[bufferseg];

  SegmentDefs.CloseFile[ImageFile ! SegmentDefs.FileError => RESUME];
  ImageFile.write ← ImageFile.append ← FALSE;

  FOR con IN [0..nbcds) DO
    [unresolved, exports] ← MapSegmentsInBcd[initgft, con, bcds[con]];
    initloadstate.bcds[con] ← [fp: NullFP, da: AltoFileDefs.eofDA,
      base: bcds[con].base, unresolved: unresolved, exports: exports,
      pages: bcds[con].pages, fill: 0];
    ENDLOOP;
  SegmentDefs.SwapUp[initstateseg];
  Image.prefix.loadStateBase ← stateseg.base;
  Image.prefix.initialLoadStateBase ← initstateseg.base;
  Image.prefix.loadStatePages ← initstateseg.pages;

  diskrequest ← DiskRequest[
    ca: auxalloc[datapages+3],
    da: auxalloc[datapages+3],
    fixedCA: FALSE,
    fp: auxalloc[SIZE[FP]],
    firstPage: FirstImageDataPage-1,
    lastPage: FirstImageDataPage+datapages-1,
    action: WriteD,
    lastAction: WriteD,
    signalCheckError: FALSE,
    option: update[BFSDefs.GetNextDA]];

  diskrequest.fp↑ ← ImageFile.fp;
  [] ← SegmentDefs.EnumerateFileSegments[BashHint];
  [] ← SegmentDefs.EnumerateFiles[BashFile];
  (diskrequest.ca+1)↑ ← Image;
  FillInCAs[Image, endofdatamapindex, diskrequest.ca+2];
  MiscDefs.SetBlock[diskrequest.da,fillinDA,datapages+3];
  (diskrequest.da+1)↑ ← HeaderDA;

  saveXferTrap ← SD[SDDefs.sXferTrap];
  SD[SDDefs.sXferTrap] ← REGISTER[Lreg];
  saveXferTrapStatus ← REGISTER[XTSreg];

  [lpn,numChars] ← BFSDefs.ActOnPages[LOOPHOLE[@diskrequest]];
  IF lpn # 0 OR numChars # 0 THEN
    BEGIN
    DisplayHeader↑ ← SD[SDDefs.sGoingAway] ← 0;
    ImageDefs.StopMesa[];
    END;
  REGISTER[WDCreg] ← wdc;
  AV↑ ← saveAllocationVector;
  SD[SDDefs.sAllocTrap] ← savealloctrap;
  SD[SDDefs.sXferTrap] ← saveXferTrap;
  REGISTER[XTSreg] ← saveXferTrapStatus;
  SD[SDDefs.sAddFileRequest] ← 0;
  Free[auxtrapFrame];
  SegmentDefs.DeleteDataSegment[HeaderSeg];
  ptSeg ← SegmentDefs.NewDataSegment[PageFromAddress[ptPointer↑],1];
  [] ← DiskDefs.ResetDisk[];
  DiskKDDefs.InitializeDiskKD[];
  BootPageTable[ImageFile, ptPointer↑];
  SegmentDefs.DeleteDataSegment[ptSeg];

  -- turn swapping back on
  AllocDefs.RemoveSwapStrategy[@SwapOutErrorStrategy];
  SD[SDDefs.sSwapTrap] ← saveswaptrap;

  RestoreProcesses[];
  ProcessDefs.EnableInterrupts[];
  ProcessFileRequests[];
```

```
  InlineDefs.COPY[from: initloadstate, to: FileSegmentAddress[stateseg],
    nwords: initstateseg.pages*AltoDefs.PageSize];
  FOR con IN [0..nbcds) DO
    LoadStateDefs.UpdateLoadState[con, bcds[con],
      initloadstate.bcds[con].unresolved, initloadstate.bcds[con].exports];
    DeleteFileSegment[bcds[con]];
    ENDLOOP;
  LoadStateDefs.ReleaseLoadState[];
  SegmentDefs.Unlock[initstateseg];
  SegmentDefs.SwapOut[initstateseg];
  SegmentDefs.DeleteDataSegment[AuxSeg];

  FreeAllSpace[];
  EnumerateNeededModules[UnlockCodeSegment];
  SwapOutMakeImageCode[];
  ImageDefs.UserCleanupProc[Restore];
  RETURN
  END;

-- auxillary storage for frames and non-saved items
AuxSeg: DataSegmentHandle;
freepointer: POINTER;
wordsleft: CARDINAL;

SetupAuxStorage: PROCEDURE =
  BEGIN
  av : POINTER;
  i: CARDINAL;
  AuxSeg ← NewDataSegment[DefaultBase,10];
  av ← freepointer ← DataSegmentAddress[AuxSeg];
  wordsleft ← 10*AltoDefs.PageSize;
  [] ← auxalloc[AllocationVectorSize];
  freepointer ← freepointer+3; wordsleft ← wordsleft-3;
  FOR i IN [0..MaxAllocSlot) DO
    (av+i)↑ ← (i+1)*4+2;
    ENDLOOP;
  (av+6)↑ ← (av+MaxAllocSlot)↑ ← (av+MaxAllocSlot+1)↑ ← 1;
  END;

auxalloc: PROCEDURE [n: CARDINAL] RETURNS [p: POINTER] =
  BEGIN -- allocate in multiples of 4 words
  p ← freepointer;
  n ← InlineDefs.BITAND[n+3,177774B];
  freepointer ← freepointer+n;
  IF wordsleft < n THEN ImageDefs.PuntMesa[];
  wordsleft ← wordsleft-n;
  RETURN
  END;

auxtrap: PROCEDURE RETURNS [myframe: FrameHandle] =
  BEGIN
  state: StateVector;
  newframe: FrameHandle;
  eventry: POINTER TO EntryVectorItem;
  fsize, findex: CARDINAL;
  newG: GlobalFrameHandle;
  dest, tempdest: ControlLink;
  alloc: BOOLEAN;
  gfi: GFTIndex;
  ep: CARDINAL;

  myframe ← LOOPHOLE[REGISTER[Lreg]];
  state.dest ← myframe.returnlink; state.source ← 0;
  state.instbyte←0;
  state.stk[0]←myframe;
  state.stkptr←1;

  ProcessDefs.DisableInterrupts[];

  DO
    ProcessDefs.EnableInterrupts[];
    TRANSFER WITH state;

    ProcessDefs.DisableInterrupts[];
    state ← STATE;
```

```
    dest ← LOOPHOLE[REGISTER[ATPreg]];
    myframe.returnlink ← state.source;
    tempdest ← dest;
    DO
      SELECT tempdest.tag FROM
        frame =>
          BEGIN
          alloc ← TRUE;
          findex ← LOOPHOLE[tempdest, CARDINAL]/4;
          EXIT
          END;
        procedure =>
          BEGIN OPEN proc: LOOPHOLE[tempdest, ControlDefs.ProcDesc];
          gfi ← proc.gfi;  ep ← proc.ep;
          [frame: newG, epbase: findex] ← GFT[gfi];
          eventry ← @newG.code.prefix.entry[findex+ep];
          findex ← eventry.framesize;
          alloc ← FALSE;
          EXIT
          END;
        indirect => tempdest ← tempdest.link↑;
        ENDCASE => ImageDefs.PuntMesa[];
      ENDLOOP;

    IF findex >= MaxAllocSlot THEN ImageDefs.PuntMesa[]
    ELSE
      BEGIN
      fsize ← FrameVec[findex]+1;  -- includes overhead word
      newframe ← LOOPHOLE[freepointer+1];
      freepointer↑ ← findex;
      freepointer ← freepointer + fsize;
      IF wordsleft < fsize THEN ImageDefs.PuntMesa[]
      ELSE wordsleft ← wordsleft - fsize;
      END;

    IF alloc THEN
      BEGIN
      state.dest ← myframe.returnlink;
      state.stk[state.stkptr] ← newframe;
      state.stkptr ← state.stkptr+1;
      END
    ELSE
      BEGIN
      IF dest.tag # indirect THEN
        BEGIN
        state.dest ← newframe;
        newframe.accesslink ← newG;
        newframe.pc ← eventry.initialpc;
        newframe.returnlink ← myframe.returnlink;
        END
      ELSE
        BEGIN
        IF findex = MaxAllocSlot THEN ImageDefs.PuntMesa[];
        state.dest ← dest;
        newframe.accesslink ← LOOPHOLE[AV[findex].frame];
        AV[findex].frame ← newframe;
        END;
      state.source ← myframe.returnlink;
      END;

    ENDLOOP;
    END;

  PageTable: TYPE = MACHINE DEPENDENT RECORD [
    fp: AltoFileDefs.CFP,
    firstpage: CARDINAL,
    table: ARRAY [0..1) OF DiskDefs.DA];
  ptPointer: POINTER TO POINTER TO PageTable = LOOPHOLE[24B];

  BootPageTable: PROCEDURE [file:FileHandle, pt:POINTER TO PageTable] =
    BEGIN OPEN AltoFileDefs;
    lastpage: PageNumber;
    pageInc: PageNumber = pt.firstpage - ImageDefs.FirstImageDataPage;
    PlugHint: PROCEDURE [seg:FileSegmentHandle] RETURNS [BOOLEAN] =
      BEGIN
```

```
      IF seg.file = file THEN
        BEGIN
        seg.base ← seg.base + pageInc;
        IF seg.base IN [pt.firstpage..lastpage] THEN
          WITH s: seg SELECT FROM
            disk => s.hint ← FileHint[
              page: s.base,
              da: DiskDefs.VirtualDA[pt.table[s.base-pt.firstpage]]];
            ENDCASE;
        END;
      RETURN[FALSE]
      END;
    DropFileRequest[file];
    file.open ← TRUE;
    file.fp ← FP[serial: pt.fp.serial, leaderDA: pt.fp.leaderDA];
    FOR lastpage ← 0, lastpage+1
    UNTIL pt.table[lastpage] = DiskDefs.DA[0,0,0,0,0]
      DO NULL ENDLOOP;
    IF lastpage = 0 THEN RETURN;
    lastpage ← lastpage+pt.firstpage-1;
    [] ← EnumerateFileSegments[PlugHint];
    RETURN
    END;

-- The driver

  MakeImage: PUBLIC PROCEDURE [name: STRING] =
    BEGIN
    s: StateVector;
    InitFileRequest[];
    InitSpace[];
    s.stk[0] ← REGISTER[Greg];
    s.stkptr ← 1;
    s.instbyte ← 0;
    s.dest ← FrameDefs.SwapOutCode;
    s.source ← GetReturnLink[];
    InstallImage[name, TRUE, TRUE];
    RETURN WITH s;
    END;

  MakeUnMergedImage: PUBLIC PROCEDURE [name: STRING] =
    BEGIN
    s: StateVector;
    InitFileRequest[];
    InitSpace[];
    s.stk[0] ← REGISTER[Greg];
    s.stkptr ← 1;
    s.instbyte ← 0;
    s.dest ← FrameDefs.SwapOutCode;
    s.source ← GetReturnLink[];
    InstallImage[name, FALSE, TRUE];
    RETURN WITH s;
    END;

  END..
```