```
-- ImageInfo.mesa; edited by Sandman, May 25, 1978  12:17 PM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [PageSize],
  AltoFileDefs: FROM "altofiledefs" USING [FP],
  BcdDefs: FROM "bcddefs" USING [
    FTHandle, FTIndex, FTNull, FTSelf, MTIndex, MTHandle, NameString,
    SGIndex, SGHandle, VersionStamp],
  ControlDefs: FROM "controldefs" USING [
    FrameCodeBase, GFT, GFTItem, GlobalFrame, NullGlobalFrame],
  DirectoryDefs: FROM "directorydefs" USING [EnumerateDirectory],
  ImageFileInfoDefs: FROM "imagefileinfodefs",
  ImageDefs: FROM "imagedefs" USING [ImageHeader],
  LoaderBcdUtilDefs: FROM "loaderbcdutildefs" USING [
    BcdBase, EnumerateModuleTable, EnumerateSegTable],
  LoadStateDefs: FROM "loadstatedefs" USING [
    BcdAddress, ConfigIndex, ConfigNull, EnumerationDirection, LoadState,
    LoadStateGFT, Relocation],
  MiscDefs: FROM "miscdefs" USING [Zero],
  SDDefs: FROM "sddefs" USING [SD, sGFTLength],
  SegmentDefs: FROM "segmentdefs" USING [
    AddressFromPage, DeleteFileSegment, FileHandle, FileSegmentAddress,
    FileSegmentObject, InsertFile, NewFile, NewFileSegment, OldFileOnly,
    Read, SwapIn, SwapOut, Unlock, VMtoFileSegment],
  StringDefs: FROM "stringdefs" USING [
    AppendSubString, EquivalentSubStrings, SubString, SubStringDescriptor],
  SystemDefs: FROM "systemdefs" USING [
    AllocateHeapNode, AllocateResidentPages, FreeHeapNode, FreeSegment];

DEFINITIONS FROM ImageFileInfoDefs;

ImageInfo: PROGRAM
  IMPORTS DirectoryDefs, ImageFileInfoDefs, LoaderBcdUtilDefs, MiscDefs, SegmentDefs, StringDefs, Syste
**mDefs
  EXPORTS ImageFileInfoDefs, LoadStateDefs
  SHARES ImageDefs = BEGIN


  SetImage: PUBLIC PROCEDURE [name: STRING] =
    BEGIN OPEN SegmentDefs;
    ClearFileObjectTable[];
    ClearFrameObjectTable[];
    IF headerSeg # NIL THEN DeleteFileSegment[headerSeg];
    headerSeg ← NewFileSegment[NewFile[name, Read, OldFileOnly], 1, 1, Read];
    SwapIn[headerSeg];
    InitImageLoadState[headerSeg];
    InitializeImageCache[headerSeg];
    Unlock[headerSeg];
    RETURN
    END;

  headerSeg: FileSegmentHandle ← NIL;

  Version: PUBLIC PROCEDURE RETURNS [v: BcdDefs.VersionStamp] =
    BEGIN OPEN SegmentDefs;
    image: POINTER TO ImageDefs.ImageHeader;
    IF headerSeg = NIL THEN RETURN[[FALSE, 0, 0, [0, 0]]];
    SwapIn[headerSeg];
    image ← FileSegmentAddress[headerSeg];
    v ← image.prefix.version;
    Unlock[headerSeg];
    RETURN
    END;

  HeaderSegment: PUBLIC PROCEDURE RETURNS [FileSegmentHandle] =
    BEGIN
    RETURN[headerSeg];
    END;


  -- Global Frame Table management

  EnumerateGlobalFrames: PUBLIC PROCEDURE [
    proc: PROCEDURE [GlobalFrameHandle] RETURNS [BOOLEAN]]
    RETURNS [GlobalFrameHandle] =
    BEGIN
```

```
    i: CARDINAL;
    frame: GlobalFrameHandle;
    gft: POINTER TO ARRAY [0..0) OF ControlDefs.GFTItem ← ControlDefs.GFT;
    FOR i IN [0..SDDefs.SD[SDDefs.sGFTLength]) DO
      frame ← READ[@gft[i].frame];
      IF frame # ControlDefs.NullGlobalFrame AND READ[@gft[i].epbase] = 0
      AND proc[frame] THEN RETURN[frame];
      ENDLOOP;
    RETURN[ControlDefs.NullGlobalFrame]
    END;

  GlobalFrame: TYPE = ControlDefs.GlobalFrame;
  globalFrame: GlobalFrame;

  VirtualGlobalFrame: PUBLIC PROCEDURE [frame: GlobalFrameHandle]
    RETURNS [GlobalFrameHandle] =
    BEGIN
    CopyRead[to: @globalFrame, from: frame, nwords: SIZE[GlobalFrame]];
    RETURN[@globalFrame]
    END;


  Relocation: TYPE = LoadStateDefs.Relocation;
  BcdBase: TYPE = LoaderBcdUtilDefs.BcdBase;
  BcdAddress: TYPE = LoadStateDefs.BcdAddress;
  EnumerationDirection: TYPE = LoadStateDefs.EnumerationDirection;
  ConfigIndex: TYPE = LoadStateDefs.ConfigIndex;

  loadstate: LoadStateDefs.LoadState;
  gft: LoadStateDefs.LoadStateGFT;
  nbcds: CARDINAL;


  InputLoadState: PUBLIC PROCEDURE RETURNS [ConfigIndex] =
    BEGIN OPEN LoadStateDefs, SegmentDefs;
    i: CARDINAL;
    SwapIn[state];
    loadstate ← FileSegmentAddress[state];
    gft ← DESCRIPTOR[@loadstate.gft, READ[@SDDefs.SD[SDDefs.sGFTLength]]];
    nbcds ← 0;
    FOR i IN [0..LENGTH[gft]) DO
      IF gft[i].config # ConfigNull THEN nbcds ← MAX[nbcds, gft[i].config];
      ENDLOOP;
    nbcds ← nbcds + 1;
    RETURN[nbcds]
    END;

  ReleaseLoadState: PUBLIC PROCEDURE =
    BEGIN OPEN SegmentDefs;
    IF ~state.swappedin THEN RETURN;
    Unlock[state];
    IF state.lock = 0 THEN
      BEGIN
      SwapOut[state];
      loadstate ← NIL;
      nbcds ← 0;
      END;
    END;

  MapConfigToReal: PUBLIC PROCEDURE [cgfi: GFTIndex, config: ConfigIndex] RETURNS [rgfi: GFTIndex] =
    BEGIN
    IF cgfi = 0 THEN RETURN[0];
    FOR rgfi IN [0..LENGTH[gft]) DO
      IF gft[rgfi] = [config, cgfi] THEN RETURN [rgfi];
      ENDLOOP;
    RETURN[0];
    END;

  MapRealToConfig: PUBLIC PROCEDURE [rgfi: GFTIndex] RETURNS [cgfi: GFTIndex, config: ConfigIndex] =
    BEGIN
    RETURN[gft[rgfi].gfi, gft[rgfi].config];
    END;

  InitializeRelocation: PUBLIC PROCEDURE [config: ConfigIndex] RETURNS [rel: Relocation] =
    BEGIN
    max: CARDINAL ← 0;
```

```
    i: GFTIndex;
    FOR i IN [0..LENGTH[gft]) DO
      IF gft[i].config = config THEN max ← MAX[max, gft[i].gfi];
      ENDLOOP;
    rel ← DESCRIPTOR[SystemDefs.AllocateHeapNode[max+1], max+1];
    MiscDefs.Zero[BASE[rel], max+1];
    FOR i IN [0..LENGTH[gft]) DO
      IF gft[i].config = config THEN rel[gft[i].gfi] ← i;
      ENDLOOP;
    END;

  ReleaseRelocation: PUBLIC PROCEDURE [rel: Relocation] =
    BEGIN
    SystemDefs.FreeHeapNode[BASE[rel]];
    END;

  BcdSegFromLoadState: PUBLIC PROCEDURE [bcd: ConfigIndex] RETURNS [seg: FileSegmentHandle] =
    BEGIN OPEN SegmentDefs, b: loadstate.bcds[bcd];
    seg ← NewFileSegment[state.file, b.base, b.pages, Read];
    RETURN
    END;

  EnumerateLoadStateGFT: PUBLIC PROCEDURE [
   proc: PROCEDURE [GFTIndex, GFTIndex, ConfigIndex] RETURNS [BOOLEAN]]
    RETURNS [GFTIndex] =
    BEGIN
    i: GFTIndex;
    FOR i IN [0..LENGTH[gft]) DO
      IF proc[i, gft[i].gfi, gft[i].config] THEN RETURN[i];
      ENDLOOP;
    RETURN[0]
    END;

  EnumerateLoadStateBcds: PUBLIC PROCEDURE [dir: EnumerationDirection,
    proc: PROCEDURE [ConfigIndex, BcdAddress] RETURNS [BOOLEAN]]
    RETURNS [config: ConfigIndex, bcd: BcdAddress] =
    BEGIN
    i: CARDINAL;
    SELECT dir FROM
      recentfirst =>
        FOR i DECREASING IN [0..nbcds) DO
          IF proc[i, @loadstate.bcds[i]] THEN RETURN[i, @loadstate.bcds[i]];
          ENDLOOP;
      recentlast =>
        FOR i IN [0..nbcds) DO
          IF proc[i, @loadstate.bcds[i]] THEN RETURN[i, @loadstate.bcds[i]];
          ENDLOOP;
      ENDCASE;
    RETURN[LoadStateDefs.ConfigNull, NIL]
    END;

  InitImageLoadState: PUBLIC PROCEDURE [seg: FileSegmentHandle] =
    BEGIN OPEN SegmentDefs;
    image: POINTER TO ImageDefs.ImageHeader;
    SwapIn[seg];
    image ← FileSegmentAddress[seg];
    state ← NewFileSegment[seg.file, image.prefix.initialLoadStateBase,
      image.prefix.loadStatePages, Read];
    END;

  state: FileSegmentHandle;

  -- SymbolTable Lookup

  FTIndex: TYPE = BcdDefs.FTIndex;
  FTHandle: TYPE = BcdDefs.FTHandle;
  MTIndex: TYPE = BcdDefs.MTIndex;
  MTHandle: TYPE = BcdDefs.MTHandle;
  SGIndex: TYPE = BcdDefs.SGIndex;
  SGHandle: TYPE = BcdDefs.SGHandle;

  AddSymbolFileName: PUBLIC PROCEDURE [
    bcd: BcdBase, sgi: SGIndex, sgh: SGHandle] =
    BEGIN
    ExistingFile: PROCEDURE [f: FileItem] RETURNS [BOOLEAN] =
      BEGIN
```

```
      RETURN[f.fti = sgh.file];
      END;
    f: FileItem;
    IF sgh.class = code THEN RETURN;
    IF EnumFileObjects[ExistingFile] = NIL THEN
      BEGIN
      f ← GetFileObject[];
      f↑ ← [fileHandle: NIL, fti: sgh.file];
      END;
    IF sgh.file = BcdDefs.FTSelf THEN
      f.fileHandle ← SegmentDefs.VMtoFileSegment[bcd].file;
    RETURN;
    END;

  AddModuleSymbols: PUBLIC PROCEDURE [
    bcd: BcdBase, config: ConfigIndex, mti: MTIndex, mth: MTHandle] =
    BEGIN
    sgh: SGHandle = LOOPHOLE[bcd+bcd.sgOffset, CARDINAL]+mth.sseg;
    SymbolFile: PROCEDURE [f: FileItem] RETURNS [BOOLEAN] =
      BEGIN
      RETURN[f.fti = sgh.file];
      END;
    fi: FileItem;
    fr: FrameItem;
    symfile: SegmentDefs.FileHandle;
    frame: GlobalFrameHandle;
    symfile ←
      IF (fi ← EnumFileObjects[SymbolFile]) = NIL THEN NIL ELSE fi.fileHandle;
    frame ← READ[@ControlDefs.GFT[MapConfigToReal[mth.gfi, config]].frame];
    fr ← GetFrameItem[];
    fr.symbols ← IF symfile = NIL OR sgh.pages = 0 THEN NIL ELSE
      SegmentDefs.NewFileSegment[symfile, sgh.base, sgh.pages+sgh.extraPages,
        SegmentDefs.Read];
    fr.frame ← frame;
    RETURN;
    END;

  bcd: BcdBase;

  FindAllSymbols: PUBLIC PROCEDURE =
    BEGIN
    LookupEachBcd: PROCEDURE [config: ConfigIndex, baddr: BcdAddress]
      RETURNS [BOOLEAN] =
      BEGIN
      EnterFile: PROCEDURE [sgh: SGHandle, sgi: SGIndex]
        RETURNS [BOOLEAN] =
        BEGIN
        AddSymbolFileName[bcd, sgi, sgh];
        RETURN[FALSE];
        END;
      DoModule: PROCEDURE [mth: MTHandle, mti: MTIndex]
        RETURNS [BOOLEAN] =
        BEGIN
        AddModuleSymbols[bcd, config, mti, mth];
        RETURN[FALSE];
        END;
      bcdseg: FileSegmentHandle ← BcdSegFromLoadState[config];
      SegmentDefs.SwapIn[bcdseg];
      bcd ← SegmentDefs.FileSegmentAddress[bcdseg];
      [] ← LoaderBcdUtilDefs.EnumerateSegTable[bcd, EnterFile];
      LookupFiles[];
      [] ← LoaderBcdUtilDefs.EnumerateModuleTable[bcd, DoModule];
      ClearFileObjectTable[];
      RETURN[FALSE]
      END;
    [] ← InputLoadState[];
    [] ← EnumerateLoadStateBcds[recentfirst, LookupEachBcd];
    ReleaseLoadState[];
    RETURN
    END;

  SymbolSegForFrame: PUBLIC PROCEDURE [frame: GlobalFrameHandle] RETURNS [seg: FileSegmentHandle] =
    BEGIN
    SymbolFile: PROCEDURE [f: FrameItem] RETURNS [BOOLEAN] =
      BEGIN
      RETURN[f.frame = frame];
```

```
    END;
  FindModule: PROCEDURE [mth: MTHandle, mti: BcdDefs.MTIndex]
    RETURNS [BOOLEAN] =
    BEGIN
    RETURN[mth.gfi = cgfi];
    END;
  f: FrameItem;
  bcdseg: FileSegmentHandle;
  config: ConfigIndex;  cgfi: GFTIndex;
  mth: MTHandle;  mti: MTIndex;
  sgb: CARDINAL;
  IF frame = ControlDefs.NullGlobalFrame THEN RETURN[NIL];
  IF (f ← EnumFrameObjects[SymbolFile]) # NIL THEN RETURN[f.symbols];
  IF VirtualGlobalFrame[frame].copied THEN
    BEGIN
    original: GlobalFrameHandle ← FindOriginal[frame];
    seg ← SymbolSegForFrame[original];
    AddSymbolsForCopy[frame, seg];
    RETURN
    END;
  [] ← InputLoadState[];
  [config: config, cgfi: cgfi] ←
    MapRealToConfig[VirtualGlobalFrame[frame].gfi];
  bcdseg ← BcdSegFromLoadState[config];
  SegmentDefs.SwapIn[bcdseg];
  bcd ← SegmentDefs.FileSegmentAddress[bcdseg];
  [mth: mth, mti: mti] ←
    LoaderBcdUtilDefs.EnumerateModuleTable[bcd, FindModule];
  sgb ← LOOPHOLE[bcd+bcd.sgOffset];
  AddSymbolFileName[bcd, mth.sseg, mth.sseg+sgb];
  LookupFiles[];
  AddModuleSymbols[bcd, config, mti, mth];
  ClearFileObjectTable[];
  ReleaseLoadState[];
  SegmentDefs.Unlock[bcdseg];
  SegmentDefs.DeleteFileSegment[bcdseg];
  RETURN[EnumFrameObjects[SymbolFile].symbols]
  END;

FindOriginal: PUBLIC PROCEDURE [copy: GlobalFrameHandle]
  RETURNS [GlobalFrameHandle] =
  BEGIN
  Original: PROCEDURE [f: GlobalFrameHandle] RETURNS [BOOLEAN] =
    BEGIN
    RETURN[f # copy AND ~VirtualGlobalFrame[f].copied AND
      SameCode[copy, f]  = identical];
    END;
  RETURN[EnumerateGlobalFrames[Original]]
  END;

CodeMatch: TYPE = {identical, same, different};

SameCode: PROCEDURE [f1, f2: GlobalFrameHandle] RETURNS [CodeMatch] =
  BEGIN
  o1, o2: CARDINAL;
  s1, s2: FileSegmentHandle;
  fcb: ControlDefs.FrameCodeBase;
  s1 ← READ[@f1.codesegment];
  s2 ← READ[@f2.codesegment];
  IF s1 # s2 THEN RETURN[different];
  fcb ← READ[@f1.code];
  IF ~fcb.swappedout THEN
    o1 ← fcb.codebase -
      SegmentDefs.AddressFromPage[VirtualFileSegment[s1].VMpage]
  ELSE
    BEGIN
    fcb.swappedout ← FALSE;
    o1 ← fcb.offset;
    END;
  fcb ← READ[@f2.code];
  IF ~fcb.swappedout THEN
    o2 ← fcb.codebase -
      SegmentDefs.AddressFromPage[VirtualFileSegment[s2].VMpage]
  ELSE
    BEGIN
    fcb.swappedout ← FALSE;
```

```
    o2 ← fcb.offset;
    END;
  RETURN[IF o1 = o2 THEN identical ELSE same];
  END;

FrameToModuleName: PUBLIC PROCEDURE [frame: GlobalFrameHandle, name: STRING] =
  BEGIN
  cgfi: GFTIndex;
  config: ConfigIndex;
  ns: BcdDefs.NameString;
  bcdseg: FileSegmentHandle;
  bcd: BcdBase;
  FindModule: PROCEDURE [mth: MTHandle, mti: MTIndex] RETURNS [BOOLEAN] =
    BEGIN
    ss: StringDefs.SubStringDescriptor;
    IF cgfi IN [mth.gfi..mth.gfi+mth.ngfi) THEN
      BEGIN
      ss ← [base: @ns.string, offset: mth.name, length: ns.size[mth.name]];
      StringDefs.AppendSubString[name,@ss];
      RETURN[TRUE];
      END;
    RETURN[FALSE];
    END;
  IF frame = ControlDefs.NullGlobalFrame THEN RETURN;
  IF VirtualGlobalFrame[frame].copied THEN
    BEGIN
    original: GlobalFrameHandle ← FindOriginal[frame];
    FrameToModuleName[original, name];
    RETURN
    END;
  [] ← InputLoadState[];
  [config: config, cgfi: cgfi] ←
    MapRealToConfig[VirtualGlobalFrame[frame].gfi];
  bcdseg ← BcdSegFromLoadState[config];
  SegmentDefs.SwapIn[bcdseg];
  bcd ← SegmentDefs.FileSegmentAddress[bcdseg];
  ns ← LOOPHOLE[bcd+bcd.ssOffset];
  [] ← LoaderBcdUtilDefs.EnumerateModuleTable[bcd, FindModule];
  ReleaseLoadState[];
  SegmentDefs.Unlock[bcdseg];
  SegmentDefs.DeleteFileSegment[bcdseg];
  RETURN
  END;

segment: SegmentDefs.FileSegmentObject;

VirtualFileSegment: PUBLIC PROCEDURE [seg: FileSegmentHandle]
  RETURNS [FileSegmentHandle] =
  BEGIN OPEN SegmentDefs;
  CopyRead[to: @segment, from: seg, nwords: SIZE[FileSegmentObject]];
  RETURN[@segment]
  END;

AddSymbolsForCopy: PROCEDURE [f: GlobalFrameHandle, seg: FileSegmentHandle] =
  BEGIN
  fr: FrameItem;
  fr ← GetFrameItem[];
  fr↑ ← [frame: f, symbols: seg];
  RETURN
  END;



 -- File Lookup Object Management

FileObject: TYPE = RECORD [
  fileHandle: SegmentDefs.FileHandle,
  fti: BcdDefs.FTIndex];

FileItem: TYPE = POINTER TO FileObject;

FileObjectTable: TYPE = RECORD [
  link: POINTER TO FileObjectTable,
  subTable: ARRAY [0..0) OF FileObject];

fileTable: POINTER TO FileObjectTable ← NIL;
```

```
  nFiles: CARDINAL ← MaxNFileObjects;
  MaxNFileObjects: CARDINAL = (AltoDefs.PageSize-1)/SIZE[FileObject];

  GetFileObject: PROCEDURE RETURNS [f: FileItem] =
    BEGIN
    newTable: POINTER TO FileObjectTable;
    IF nFiles < MaxNFileObjects THEN
      BEGIN  f ← @fileTable.subTable[nFiles]; nFiles ← nFiles+1; END
    ELSE
      BEGIN
      newTable ← SystemDefs.AllocateResidentPages[1];
      newTable.link ← fileTable;
      fileTable ← newTable;
      nFiles ← 1;
      RETURN[@fileTable.subTable[0]];
      END;
    END;

  EnumFileObjects: PROCEDURE [proc: PROCEDURE [f: FileItem] RETURNS [BOOLEAN]]
    RETURNS [f: FileItem] =
    BEGIN
    i: CARDINAL;
    table: POINTER TO FileObjectTable;
    IF fileTable = NIL THEN RETURN[NIL];
    IF nFiles < MaxNFileObjects THEN
      FOR i IN [0..nFiles) DO
        IF proc[f ← @fileTable.subTable[i]] THEN RETURN;
        ENDLOOP;
    FOR table ← fileTable.link, table.link UNTIL table = NIL DO
      FOR i IN [0..MaxNFileObjects) DO
        IF proc[f ← @table.subTable[i]] THEN RETURN;
        ENDLOOP;
      ENDLOOP;
    RETURN[NIL];
    END;

  ClearFileObjectTable: PROCEDURE =
    BEGIN
    table: POINTER TO FileObjectTable;
    UNTIL fileTable = NIL DO
      table ← fileTable;
      fileTable ← fileTable.link;
      SystemDefs.FreeSegment[table];
      ENDLOOP;
    nFiles ← MaxNFileObjects;
    RETURN
    END;

  -- Frame Symbol Table Management

  FrameObject: TYPE = RECORD [
    frame: GlobalFrameHandle,
    symbols: FileSegmentHandle];

  FrameItem: TYPE = POINTER TO FrameObject;

  FrameObjectTable: TYPE = RECORD [
    link: POINTER TO FrameObjectTable,
    subTable: ARRAY [0..0) OF FrameObject];

  frameTable: POINTER TO FrameObjectTable ← NIL;
  nFrames: CARDINAL ← MaxNFrameObjects;
  MaxNFrameObjects: CARDINAL = (AltoDefs.PageSize-1)/SIZE[FrameObject];

  GetFrameItem: PROCEDURE RETURNS [f: FrameItem] =
    BEGIN
    newTable: POINTER TO FrameObjectTable;
    IF nFrames < MaxNFileObjects THEN
      BEGIN  f ← @frameTable.subTable[nFrames]; nFrames ← nFrames+1; END
    ELSE
      BEGIN
      newTable ← SystemDefs.AllocateResidentPages[1];
      newTable.link ← frameTable;
      frameTable ← newTable;
      nFrames ← 1;
      RETURN[@frameTable.subTable[0]];
```

```
    END;
  END;

EnumFrameObjects: PROCEDURE [proc: PROCEDURE [f: FrameItem] RETURNS [BOOLEAN]]
  RETURNS [f: FrameItem] =
  BEGIN
  i: CARDINAL;
  table: POINTER TO FrameObjectTable;
  IF frameTable = NIL THEN RETURN[NIL];
  IF nFrames < MaxNFrameObjects THEN
    FOR i IN [0..nFrames) DO
      IF proc[f ← @frameTable.subTable[i]] THEN RETURN;
      ENDLOOP;
  FOR table ← frameTable.link, table.link UNTIL table = NIL DO
    FOR i IN [0..MaxNFrameObjects) DO
      IF proc[f ← @table.subTable[i]] THEN RETURN;
      ENDLOOP;
    ENDLOOP;
  RETURN[NIL];
  END;

ClearFrameObjectTable: PROCEDURE =
  BEGIN
  table: POINTER TO FrameObjectTable;
  UNTIL frameTable = NIL DO
    table ← frameTable;
    frameTable ← frameTable.link;
    SystemDefs.FreeSegment[table];
    ENDLOOP;
  nFrames ← MaxNFrameObjects;
  RETURN
  END;


-- Directory Lookup of File Table

nFilesToFind: CARDINAL;

LookupFiles: PUBLIC PROCEDURE =
  BEGIN
  name: STRING ← [40];
  CountFilesToLookup: PROCEDURE [f: FileItem] RETURNS [BOOLEAN] =
    BEGIN
    nFilesToFind ← nFilesToFind + 1;
    RETURN[FALSE];
    END;
  nFilesToFind ← 0;
  [] ← EnumFileObjects[CountFilesToLookup];
  DirectoryDefs.EnumerateDirectory[CheckOne];
  RETURN
  END;

CheckOne: PROCEDURE [fp: POINTER TO AltoFileDefs.FP, name: STRING]
 RETURNS [found: BOOLEAN] =
  BEGIN OPEN StringDefs;
  i: CARDINAL;
  dirName: SubStringDescriptor;
  bcd: SubStringDescriptor ← [base: "bcd"L, offset: 0, length: 3];
  file: FileItem;
  FOR i IN [0..name.length) DO
    IF name[i] = '. THEN
      BEGIN
      IF name.length-i # 5 THEN GOTO UseWholeName;
      dirName ← [base: name, offset: i+1, length: 3];
      IF ~EquivalentSubStrings[@dirName, @bcd] THEN GOTO UseWholeName;
      dirName.offset ← 0;  dirName.length ← i;
      GOTO HasBCDExtension;
      END;
    REPEAT
      UseWholeName => NULL;
      HasBCDExtension =>
        BEGIN
        [found, file] ← FindFileName[@dirName];
        IF found THEN RETURN [ThisIsTheOne[fp, file]];
        END;
    ENDLOOP;
```

```
    dirName ← [base: name, offset: 0, length: name.length-1];
    [found, file] ← FindFileName[@dirName];
    RETURN [IF found THEN ThisIsTheOne[fp, file] ELSE FALSE];
    END;

  ThisIsTheOne: PROCEDURE [fp: POINTER TO AltoFileDefs.FP, file: FileItem]
   RETURNS [BOOLEAN] =
    BEGIN
    file.fileHandle ← SegmentDefs.InsertFile[fp, SegmentDefs.Read];
    RETURN [(nFilesToFind ← nFilesToFind-1) = 0];
    END;

  FindFileName: PUBLIC PROCEDURE [name: StringDefs.SubString]
   RETURNS [found: BOOLEAN, file: FileItem] =
    BEGIN OPEN StringDefs;
    ns: BcdDefs.NameString ← LOOPHOLE[bcd+bcd.ssOffset];
    ftb: CARDINAL ← LOOPHOLE[bcd+bcd.ftOffset];
    filename: SubStringDescriptor ← [base: @ns.string, offset:, length:];
    MatchName: PROCEDURE [f: FileItem] RETURNS [BOOLEAN] =
      BEGIN
      IF f.fti = BcdDefs.FTNull THEN RETURN[FALSE];
      IF f.fti = BcdDefs.FTSelf THEN RETURN[FALSE];
      filename.offset ← (ftb+f.fti).name;
      filename.length ← ns.size[(ftb+f.fti).name];
      IF LastCharIsDot[@filename] THEN name.length ← name.length + 1;
      RETURN[EquivalentSubStrings[@filename, name]];
      END;
    f: FileItem;
    f ← EnumFileObjects[MatchName];
    RETURN[f#NIL, f];
    END;

  LastCharIsDot: PUBLIC PROCEDURE [name: StringDefs.SubString] RETURNS [BOOLEAN] =
    BEGIN
    RETURN[name.base[name.offset+name.length-1] = '.];
    END;

END...
```