```
-- Loader.mesa
-- Last Modified by Sandman,  Aug 16, 1978 10:52 AM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [PageCount],
  BcdDefs: FROM "bcddefs" USING [
    ControlLink, EPIndex, EPLimit, EXPHandle, EXPIndex, EXPNull, FTHandle,
    FTIndex, FTSelf, GFTIndex, IMPHandle, IMPIndex, MTHandle, MTIndex, MTNull,
    NameRecord, NameString, NullLink, PackedString, UnboundLink, VersionID,
    VersionStamp],
  ControlDefs: FROM "controldefs" USING [
    Free, GFT, GlobalFrameHandle, NullGlobalFrame],
  FrameDefs: FROM "framedefs" USING [EnterGlobalFrame, EnumerateGlobalFrames],
  InlineDefs: FROM "inlinedefs" USING [BITAND, COPY],
  LoaderBcdUtilDefs: FROM "loaderbcdutildefs" USING [
    BcdBase, EnumerateExportTable, EnumerateImportTable, EnumerateModuleTable,
    ReleaseBcdSeg, SetUpBcd],
  LoaderDefs: FROM "loaderdefs" USING [FileSegmentHandle, LoaderErrorType],
  LoaderUtilityDefs: FROM "loaderutilitydefs" USING [
    AllocateSingleModule, AssignControlModules, Binding, ControlModuleFrame,
    EnterCodeFileNames, FinalizeUtilities, FindCodeSegment, FindFrameIndex,
    InitializeUtilities, InitImportBinding, LookupFileTable,
    RequiredFrameSpace],
  LoadStateDefs: FROM "loadstatedefs" USING [
    BcdHasExports, BcdHasUnresolvedImports, BcdSegFromLoadState, ConfigIndex,
    EnterGfi, InitializeRelocation, InputLoadState, MapConfigToReal,
    ReleaseLoadState, ReleaseRelocation, Relocation, SetUnresolvedImports,
    UpdateLoadState],
  SDDefs: FROM "sddefs" USING [SD, sNew],
  SegmentDefs: FROM "segmentdefs" USING [
    FileHandle, FileSegmentAddress, FileSegmentHandle, MoveFileSegment, NewFile, NewFileSegment, OldFil
**eOnly, Read, SwapIn,
    SwapUp, Unlock],
  StringDefs: FROM "stringdefs" USING [
    AppendSubString, EqualSubStrings, EquivalentSubStrings,
    SubStringDescriptor],
  SystemDefs: FROM "systemdefs" USING [
    AllocateHeapNode, AllocateHeapString, AllocateResidentSegment,
    FreeHeapNode, FreeHeapString, FreePages];

DEFINITIONS FROM SegmentDefs, BcdDefs, LoaderDefs;

Loader: PROGRAM
  IMPORTS FrameDefs, LoaderBcdUtilDefs, LoaderUtilityDefs, LoadStateDefs,
    SegmentDefs, StringDefs, SystemDefs
  EXPORTS LoaderDefs =

  BEGIN

  SSD: TYPE = StringDefs.SubStringDescriptor;
  Relocation: TYPE = LoadStateDefs.Relocation;
  BcdBase: TYPE = LoaderBcdUtilDefs.BcdBase;
  ConfigIndex: TYPE = LoadStateDefs.ConfigIndex;
  Binding: TYPE = LoaderUtilityDefs.Binding;
  GlobalFrameHandle: TYPE = ControlDefs.GlobalFrameHandle;

  LoaderError: PUBLIC SIGNAL [error: LoaderErrorType] = CODE;
  InvalidBcd: PUBLIC ERROR [bcdfile: FileHandle] = CODE;
  InvalidFile: PUBLIC ERROR [name: STRING] = CODE;
  VersionMismatch: PUBLIC SIGNAL [name: STRING] = CODE;

  NewNew: PROCEDURE [name: STRING] RETURNS [frame: GlobalFrameHandle] =
    BEGIN
    RETURN[New[Load[name], TRUE, FALSE]];
    END;

  Load: PUBLIC PROCEDURE [name: STRING] RETURNS [FileSegmentHandle] =
    BEGIN
    RETURN [LoadBcd[SegmentDefs.NewFile[name, Read, OldFileOnly]
      ! InvalidBcd => ERROR InvalidFile[name]]];
    END;

  LoadBcd: PUBLIC PROCEDURE [bcdfile: FileHandle] RETURNS [bcdseg: FileSegmentHandle] =
    BEGIN OPEN SegmentDefs;
    pages: AltoDefs.PageCount;
    bcd: BcdBase;
```

```
    bcdseg ← NewFileSegment[bcdfile, 1, 1, Read];
    bcd ← LoaderBcdUtilDefs.SetUpBcd[bcdseg];
    pages ← bcd.nPages;
    IF pages > 1 THEN
      BEGIN
      Unlock[bcdseg];
      MoveFileSegment[bcdseg, 1, pages];
      bcd ← LoaderBcdUtilDefs.SetUpBcd[bcdseg];
      END;
    BEGIN
      ENABLE UNWIND => LoaderBcdUtilDefs.ReleaseBcdSeg[bcdseg];
      IF bcd.versionident # BcdDefs.VersionID OR bcd.definitions THEN
        ERROR InvalidBcd[bcdfile];
    END; -- OF OPEN
    Unlock[bcdseg];
    END;

  New: PUBLIC PROCEDURE [bcdseg: FileSegmentHandle, framelinks, alloc: BOOLEAN]
    RETURNS [frame: GlobalFrameHandle] =
    BEGIN OPEN SegmentDefs, SystemDefs;
    NullRel: Relocation = DESCRIPTOR[NIL, 0];
    NullBind: Binding = DESCRIPTOR[NIL, 0];
    loadee, system: BcdBase ← NIL;
    LReloc: Relocation ← NullRel;
    SReloc: Relocation ← NullRel;
    LBind: Binding ← NullBind;
    SBind: Binding ← NullBind;
    frames: POINTER ← NIL;
    nbcds, i: CARDINAL;
    systemseg: FileSegmentHandle ← NIL;
    single, unresolved, sysunres, initial: BOOLEAN;
    CleanUpNew: PROCEDURE =
      BEGIN
      LoadStateDefs.ReleaseLoadState[];
      IF loadee # NIL THEN LoaderBcdUtilDefs.ReleaseBcdSeg[bcdseg];
      IF LReloc # NullRel THEN SystemDefs.FreeHeapNode[BASE[LReloc]];
      IF LBind # NullBind THEN SystemDefs.FreeHeapNode[BASE[LBind]];
      LoaderUtilityDefs.FinalizeUtilities[];
      END;

    BEGIN ENABLE UNWIND =>
      BEGIN
      CleanUpNew[];
      IF frames # NIL THEN
        IF single THEN ControlDefs.Free[frames] ELSE FreePages[frames];
      END;
    loadee ← LoaderBcdUtilDefs.SetUpBcd[bcdseg];
    LoaderUtilityDefs.InitializeUtilities[loadee];
    LoaderUtilityDefs.EnterCodeFileNames[loadee];
    LoaderUtilityDefs.LookupFileTable[];
    single ← loadee.nModules = 1;
    frames ← AllocateFrames[loadee, single, alloc, framelinks];
    LReloc ←
      DESCRIPTOR[AllocateHeapNode[loadee.firstdummy], loadee.firstdummy];
    nbcds ← LoadStateDefs.InputLoadState[];
    AssignFrameAddresses[frames, loadee, LReloc, nbcds, single, alloc, framelinks];
    LBind ← LoaderUtilityDefs.InitImportBinding[loadee.nDummies];
    unresolved ← loadee.nImports # 0;
    initial ← TRUE;
    IF ~unresolved THEN RelocateOnly[loadee, LReloc];
    FOR i DECREASING IN [0..nbcds) DO
      IF unresolved AND LoadStateDefs.BcdHasExports[i] THEN
        BEGIN
          ENABLE UNWIND =>
            IF systemseg # NIL THEN LoaderBcdUtilDefs.ReleaseBcdSeg[systemseg];
          systemseg ← LoadStateDefs.BcdSegFromLoadState[i];
          system ← LoaderBcdUtilDefs.SetUpBcd[systemseg];
        BindImports[loadee, system, LBind];
        unresolved ← ProcessControlLinks[loadee, system, LReloc, LBind, i, initial];
        initial ← FALSE;
        END;
      IF LoadStateDefs.BcdHasUnresolvedImports[i] AND loadee.nExports # 0 THEN
        BEGIN
          ENABLE UNWIND =>
            BEGIN
            IF systemseg # NIL THEN LoaderBcdUtilDefs.ReleaseBcdSeg[systemseg];
```

```
          IF SReloc # NullRel THEN LoadStateDefs.ReleaseRelocation[SReloc];
          IF SBind # NullBind THEN SystemDefs.FreeHeapNode[BASE[SBind]];
          END;
        IF systemseg = NIL THEN
          BEGIN
          systemseg ← LoadStateDefs.BcdSegFromLoadState[i];
          system ← LoaderBcdUtilDefs.SetUpBcd[systemseg];
          END;
      SReloc ← LoadStateDefs.InitializeRelocation[i];
      SBind ← LoaderUtilityDefs.InitImportBinding[system.nDummies];
      BindImports[system, loadee, SBind];
      sysunres ←
        ProcessControlLinks[system, loadee, SReloc, SBind, nbcds, FALSE];
      LoadStateDefs.SetUnresolvedImports[i, sysunres];
      LoadStateDefs.ReleaseRelocation[SReloc];  SReloc ← NullRel;
      SystemDefs.FreeHeapNode[BASE[SBind]];  SBind ← NullBind;
      END;
    IF systemseg # NIL THEN
      BEGIN
      LoaderBcdUtilDefs.ReleaseBcdSeg[systemseg];
      systemseg ← NIL;
      END;
    ENDLOOP;
  LoadStateDefs.UpdateLoadState[
    nbcds, bcdseg, unresolved, (loadee.nExports # 0 OR single)];
  frame ← IF single THEN LOOPHOLE[frames]
    ELSE LoaderUtilityDefs.ControlModuleFrame[loadee, LReloc];
  CleanUpNew[];
  END;
  END;

AllocateFrames: PROCEDURE [loadee: BcdBase, single, alloc, framelinks: BOOLEAN]
  RETURNS [POINTER] =
  BEGIN OPEN SegmentDefs;
  RETURN[IF single
    THEN LoaderUtilityDefs.AllocateSingleModule[loadee, framelinks]
    ELSE SystemDefs.AllocateResidentSegment[
      LoaderUtilityDefs.RequiredFrameSpace[loadee, alloc, framelinks]]];
  END;

AssignFrameAddresses: PROCEDURE [p: POINTER, loadee: BcdBase, Reloc: Relocation,
config: ConfigIndex, single, alloc, allframelinks: BOOLEAN] =
  BEGIN
  frame: GlobalFrameHandle ← p;

  ModuleSearch: PROCEDURE [mth: MTHandle, mti: MTIndex] RETURNS [BOOLEAN] =
    BEGIN
    seg: SegmentDefs.FileSegmentHandle;
    gfi: GFTIndex;
    i: CARDINAL;
    framelinks: BOOLEAN;

    framelinks ← allframelinks OR mth.links = frame OR ~mth.code.linkspace;
    IF ~single AND alloc THEN
      BEGIN
      p ← NextMultipleOfFour[p+1];
      (p-1)↑ ← LoaderUtilityDefs.FindFrameIndex[mth, framelinks];
      END;
    IF ~single AND framelinks THEN p ← p + mth.frame.length;
    frame ← NextMultipleOfFour[p];
    p ← frame + mth.framesize;
    gfi ← FrameDefs.EnterGlobalFrame[frame, mth.ngfi];
    FOR i IN [0..mth.ngfi) DO
      Reloc[mth.gfi+i] ← gfi + i;
      LoadStateDefs.EnterGfi[mth.gfi+i, gfi+i, config];
      ENDLOOP;
    seg ← LoaderUtilityDefs.FindCodeSegment[loadee, mth, frame];
    seg.class ← code;
    frame↑ ← [gfi: gfi, unused: 0, alloced: alloc OR single, shared: FALSE,
      copied: FALSE, started: FALSE, trapxfers: FALSE, codelinks: ~framelinks,
      code: [out[mth.code.offset]], codesegment: seg, global:];
    frame.code.swappedout ← TRUE;
    RETURN[FALSE];
    END;
  FindSharedModules: PROCEDURE [f: GlobalFrameHandle] RETURNS [BOOLEAN] =
    BEGIN
```

```
    Search: PROCEDURE [mth: MTHandle, mti: MTIndex] RETURNS [BOOLEAN] =
      BEGIN
      frame: GlobalFrameHandle ← ControlDefs.GFT[Reloc[mth.gfi]].frame;
      IF f = frame THEN RETURN[FALSE];
      IF f.codesegment = frame.codesegment THEN
        BEGIN f.shared ← TRUE; frame.shared ← TRUE; END;
      RETURN[FALSE];
      END;
    IF f.shared THEN RETURN[FALSE];
    [] ← LoaderBcdUtilDefs.EnumerateModuleTable[loadee, Search];
    RETURN[FALSE];
    END;
  gfi: GFTIndex;
  Reloc[0] ← 0;
  [] ← LoaderBcdUtilDefs.EnumerateModuleTable[loadee, ModuleSearch];
  FOR gfi IN [1..LENGTH[Reloc]) DO
    LoadStateDefs.EnterGfi[gfi, Reloc[gfi], config];
    ENDLOOP;
  LoaderUtilityDefs.AssignControlModules[loadee, Reloc];
  [] ← FrameDefs.EnumerateGlobalFrames[FindSharedModules];
  END;

NextMultipleOfFour: PROCEDURE [n: POINTER] RETURNS [POINTER] =
  BEGIN
  RETURN[n + InlineDefs.BITAND[-LOOPHOLE[n, INTEGER], 3B]];
  END;

BindImports: PROCEDURE [loadee, system: BcdBase, ImportBinding: Binding] =
  BEGIN
  ForEachImport: PROCEDURE [ith: IMPHandle, iti: IMPIndex] RETURNS [BOOLEAN] =
    BEGIN
    i: CARDINAL;
    iname, sysname: SSD;
    issb, sysssb: POINTER TO BcdDefs.PackedString;
    module: MTIndex;    export: EXPIndex;
    ExportMatch: PROCEDURE [eth: EXPHandle, eti: EXPIndex] RETURNS [BOOLEAN] =
      BEGIN OPEN StringDefs;
      sysname.offset ← eth.name;  sysname.length ← sysssb.size[eth.name];
      RETURN[eth.port = ith.port AND EqualSubStrings[@iname, @sysname] AND
        EqualFiles[loadee, system, ith.file, eth.file]]
      END;
    ModuleMatch: PROCEDURE [mth: MTHandle, mti: MTIndex] RETURNS [BOOLEAN] =
      BEGIN OPEN StringDefs;
      sysname.offset ← mth.name;  sysname.length ← sysssb.size[mth.name];
      RETURN[EqualSubStrings[@iname, @sysname] AND
        EqualFiles[loadee, system, ith.file, mth.file]]
      END;

    issb ← LOOPHOLE[loadee+loadee.ssOffset];
    iname ← SSD[base: @issb.string, offset: ith.name, length: issb.size[ith.name]];
    sysssb ← LOOPHOLE[system+system.ssOffset];
    sysname.base ← @sysssb.string;
    export ← LoaderBcdUtilDefs.EnumerateExportTable[system, ExportMatch].eti;
    IF  export # EXPNull THEN
      FOR i IN [0..ith.ngfi) DO
        ImportBinding[ith.gfi-loadee.firstdummy+i] ←
          [whichgfi: i, body: interface[export]];
        ENDLOOP
    ELSE
      BEGIN
      module ←
        LoaderBcdUtilDefs.EnumerateModuleTable[system, ModuleMatch].mti;
      FOR i IN [0..ith.ngfi) DO
        IF module = MTNull
          THEN ImportBinding[ith.gfi-loadee.firstdummy+i] ←
            [whichgfi: i, body: notbound[]]
          ELSE ImportBinding[ith.gfi-loadee.firstdummy+i] ←
            [whichgfi: i, body: module[module]];
        ENDLOOP;
      END;
    RETURN[FALSE];
    END;

  [] ← LoaderBcdUtilDefs.EnumerateImportTable[loadee, ForEachImport];
  END;
```

```
EqualFiles: PROCEDURE [bcd1, bcd2: BcdBase, file1, file2: FTIndex] RETURNS [BOOLEAN] =
  BEGIN
  name1, name2: SSD;
  v1, v2: POINTER TO BcdDefs.VersionStamp;
  ps1: BcdDefs.NameString ← LOOPHOLE[bcd1+bcd1.ssOffset];
  ps2: BcdDefs.NameString ← LOOPHOLE[bcd2+bcd2.ssOffset];
  name1.base ← @ps1.string;
  name2.base ← @ps2.string;
  IF file1 = FTSelf THEN
    BEGIN
    name: NameRecord =
      (LOOPHOLE[bcd1+bcd1.mtOffset, CARDINAL] + FIRST[MTIndex]).name;
    name1.offset ← name;
    name1.length ← ps1.size[name];
    v1 ← @bcd1.version;
    END
  ELSE
    BEGIN
    file: FTHandle ← LOOPHOLE[bcd1+bcd1.ftOffset, CARDINAL] + file1;
    name1.offset ← file.name;
    name1.length ← ps1.size[file.name];
    v1 ← @file.version;
    END;
  IF file2 = FTSelf THEN
    BEGIN
    name: NameRecord =
      (LOOPHOLE[bcd2+bcd2.mtOffset, CARDINAL] + FIRST[MTIndex]).name;
    name2.offset ← name;
    name2.length ← ps2.size[name];
    v2 ← @bcd2.version;
    END
  ELSE
    BEGIN
    file: FTHandle ← LOOPHOLE[bcd2+bcd2.ftOffset, CARDINAL] + file2;
    name2.offset ← file.name;
    name2.length ← ps2.size[file.name];
    v2 ← @file.version;
    END;
  IF StringDefs.EquivalentSubStrings[@name1, @name2] THEN
    IF EqVer[v1, v2] THEN RETURN[TRUE]
    ELSE
      BEGIN OPEN SystemDefs;
      filename: STRING ← AllocateHeapString[name1.length];
      StringDefs.AppendSubString[filename, @name1];
      SIGNAL VersionMismatch[filename ! UNWIND => FreeHeapString[filename]];
      FreeHeapString[filename];
      END;
  RETURN[FALSE];
  END;

EqVer: PROCEDURE [v1, v2: POINTER TO BcdDefs.VersionStamp] RETURNS [BOOLEAN] =
  BEGIN
  RETURN [v1.zapped OR v2.zapped OR v1↑ = v2↑]
  END;


ProcessControlLinks: PROCEDURE [loadee, system: BcdBase, Reloc: Relocation,
ImportBinding: Binding, config: ConfigIndex, initial: BOOLEAN]
RETURNS [BOOLEAN]=
  BEGIN
  smtb: CARDINAL = LOOPHOLE[system+system.mtOffset];
  setb: CARDINAL = LOOPHOLE[system+system.expOffset];
  unresolved: BOOLEAN ← FALSE;

  MungeControlLink: PROCEDURE [link: ControlLink, addr: POINTER TO ControlLink]
    RETURNS [changedLink: BOOLEAN] =
    BEGIN
    gfi: GFTIndex;
    ep: EPIndex;
    SELECT link.tag FROM
      procedure =>
        IF addr↑ = UnboundLink THEN
          BEGIN
          IF link.gfi >= loadee.firstdummy THEN
            WITH ImportBinding[link.gfi-loadee.firstdummy] SELECT FROM
              module => SIGNAL LoaderError[Impossible];  -- Shouldn't Happen
```

```
            interface =>
              BEGIN OPEN e: setb+eti;
              SELECT e.port FROM
                interface =>
                  BEGIN
                  ep ← link.ep+(whichgfi*EPLimit);
                  gfi ← LoadStateDefs.MapConfigToReal[e.links[ep].gfi, config];
                  IF gfi = 0 THEN unresolved ← TRUE
                  ELSE
                    BEGIN
                    changedLink ← TRUE;
                    SELECT e.links[ep].tag FROM
                      procedure =>
                        BEGIN
                        addr.gfi ← gfi;
                        addr.ep ← e.links[ep].ep;
                        addr.tag ← procedure;
                        END;
                      frame => addr↑ ←
                        LOOPHOLE[ControlDefs.GFT[gfi].frame];
                      ENDCASE;
                    END;
                  END;
                module => SIGNAL LoaderError[Impossible];
                ENDCASE;
              END;
            notbound => unresolved ← TRUE;
            ENDCASE
        ELSE
          BEGIN
          addr↑ ← link;
          addr.gfi ← Reloc[link.gfi];
          changedLink ← TRUE;
          END;
      END;
    frame =>
      IF addr↑ = NullLink THEN
        BEGIN
        IF link.gfi >= loadee.firstdummy THEN
          BEGIN
          WITH ImportBinding[link.gfi-loadee.firstdummy] SELECT FROM
            module => gfi ← LoadStateDefs.MapConfigToReal[(smtb+mti).gfi, config];
            interface =>
              BEGIN OPEN e: setb+eti;
              SELECT e.port FROM
                interface => ep ← link.ep+(whichgfi*EPLimit);
                module => ep ← 0;
                ENDCASE;
              gfi ← LoadStateDefs.MapConfigToReal[e.links[ep].gfi, config];
              END;
            notbound => gfi ← 0;
            ENDCASE;
          IF gfi = 0 THEN unresolved ← TRUE
          ELSE
            BEGIN
            changedLink ← TRUE;
            addr↑ ← LOOPHOLE[ControlDefs.GFT[gfi].frame];
            END;
          END
        ELSE
          BEGIN
          changedLink ← TRUE;
          addr↑ ← LOOPHOLE[ControlDefs.GFT[Reloc[link.gfi]].frame];
          END;
        END;
    ENDCASE;
  END;

ModuleSearch: PROCEDURE [mth: MTHandle, mti: MTIndex] RETURNS [BOOLEAN] =
  BEGIN
  i: CARDINAL;
  frame: GlobalFrameHandle ← ControlDefs.GFT[Reloc[mth.gfi]].frame;
  codesegment: SegmentDefs.FileSegmentHandle;
  linkbase: POINTER TO ControlLink;
  changed: BOOLEAN;
  IF frame = ControlDefs.NullGlobalFrame THEN RETURN[FALSE];
```

```
      codesegment ← frame.codesegment;
      IF frame.codelinks THEN
        BEGIN OPEN SegmentDefs;
        SwapIn[codesegment];
        linkbase ← FileSegmentAddress[codesegment] + mth.code.offset;
        END
      ELSE linkbase ← LOOPHOLE[frame];
      linkbase ← linkbase - mth.frame.length;
      IF initial THEN
        BEGIN
        InlineDefs.COPY[from: @mth.frame.frag[0],
          to: linkbase, nwords: mth.frame.length];
        FOR i IN [0..mth.frame.length) DO
          SELECT (linkbase+i)↑.tag FROM
            procedure => (linkbase+i)↑ ← UnboundLink;
            frame => (linkbase+i)↑ ← NullLink;
            ENDCASE;
          ENDLOOP;
        END;
      changed ← FALSE;
      FOR i IN [0..mth.frame.length) DO
        changed ← MungeControlLink[mth.frame.frag[i], linkbase+i] OR changed;
        ENDLOOP;
      IF frame.codelinks THEN
        BEGIN
        SegmentDefs.Unlock[codesegment];
        IF changed OR initial THEN
          BEGIN
          codesegment.write ← TRUE;
          SegmentDefs.SwapUp[codesegment];
          codesegment.write ← FALSE;
          END;
        END;
      RETURN[FALSE];
      END;

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

  RelocateOnly: PROCEDURE [loadee: BcdBase, Reloc: Relocation]=
    BEGIN
    ModuleSearch: PROCEDURE [mth: MTHandle, mti: MTIndex] RETURNS [BOOLEAN] =
      BEGIN
      i: CARDINAL;
      frame: GlobalFrameHandle ← ControlDefs.GFT[Reloc[mth.gfi]].frame;
      codesegment: SegmentDefs.FileSegmentHandle ← frame.codesegment;
      codelinks: BOOLEAN ← frame.codelinks;
      linkbase: POINTER TO ControlLink;
      IF mth.frame.length = 0 THEN RETURN[FALSE];
      IF codelinks THEN
        BEGIN OPEN SegmentDefs;
        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
        OPEN link: mth.frame.frag[i];
        SELECT link.tag FROM
          procedure =>
            BEGIN
            (linkbase+i)↑ ← link;
            (linkbase+i).gfi ← Reloc[link.gfi];
            END;
          frame =>
            (linkbase+i)↑ ← LOOPHOLE[ControlDefs.GFT[Reloc[link.gfi]].frame];
          ENDCASE;
        ENDLOOP;
      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;

  SDDefs.SD[SDDefs.sNew] ← NewNew;

  END...
```