```
-- BcdMerge.Mesa  Edited by Sandman on September 26, 1977  4:53 PM

DIRECTORY
  AltoDefs: FROM "altodefs",
  AltoFileDefs: FROM "altofiledefs",
  BcdDefs: FROM "bcddefs",
  BcdMergeDefs: FROM "bcdmergedefs",
  BcdTabDefs: FROM "bcdtabdefs",
  BcdTableDefs: FROM "bcdtabledefs",
  LoaderBcdUtilDefs: FROM "loaderbcdutildefs",
  ControlDefs: FROM "ControlDefs",
  FrameDefs: FROM "framedefs",
  InlineDefs: FROM "inlinedefs",
  LoadStateDefs: FROM "loadstatedefs",
  MiscDefs: FROM "miscdefs",
  OsStaticDefs: FROM "osstaticdefs",
  SegmentDefs: FROM "segmentdefs",
  StringDefs: FROM "stringdefs",
  SystemDefs: FROM "systemdefs",
  TimeDefs: FROM "timedefs";

DEFINITIONS FROM LoadStateDefs, LoaderBcdUtilDefs, BcdDefs;

BcdMerge: PROGRAM
  IMPORTS BcdTabDefs, BcdTableDefs, FrameDefs, LoaderBcdUtilDefs, LoadStateDefs,
    MiscDefs, SegmentDefs, StringDefs, SystemDefs
  EXPORTS BcdMergeDefs = PUBLIC
  BEGIN

  GlobalFrameHandle: TYPE = ControlDefs.GlobalFrameHandle;
  HTIndex: TYPE = BcdTabDefs.HTIndex;
  SubStringDescriptor: TYPE = StringDefs.SubStringDescriptor;
  SubString: TYPE = StringDefs.SubString;

  bcdheader: BcdDefs.BCD;
  bcd: BcdBase;  -- bcd being merged;
  configOffset: CARDINAL;
  NullVersion: BcdDefs.VersionStamp = [zapped: FALSE, net: 0, host: 0, time: [0,0]];
  mtb, ftb, ctb, itb, etb, ntb: BcdTableDefs.TableBase;
  ssb: STRING;

  Notifier: PRIVATE BcdTableDefs.TableNotifier =
    BEGIN OPEN BcdTableDefs;
    mtb ← base[mttype];
    ftb ← base[fttype];
    ctb ← base[cttype];
    itb ← base[imptype];
    etb ← base[exptype];
    ntb ← base[nttype];
    ssb ← LOOPHOLE[base[sstype]];
    END;

  EnterName: PROCEDURE [ss: SubString] RETURNS [NameRecord] =
    BEGIN OPEN BcdTabDefs;
    lss: SubStringDescriptor;
    hti: HTIndex = EnterString[ss];
    SubStringForHash[@lss,hti];
    RETURN[[lss.offset, lss.length]];
    END;

  MapName: PROCEDURE [bcd: BcdBase, n: NameRecord] RETURNS [NameRecord] =
    BEGIN
    ss: SubStringDescriptor ←
      [base: LOOPHOLE[bcd+bcd.ssOffset], offset: n.offset, length: n.length];
    RETURN[EnterName[@ss]];
    END;

  MapEquivalentName: PROCEDURE [bcd: BcdBase, n: NameRecord] RETURNS [NameRecord] =
    BEGIN
    found: BOOLEAN;
    hti: HTIndex;
    ss: SubStringDescriptor ←
      [base: LOOPHOLE[bcd+bcd.ssOffset], offset: n.offset, length: n.length];
    [found, hti] ← BcdTabDefs.FindEquivalentString[@ss];
    RETURN[IF found THEN NameForHti[hti] ELSE EnterName[@ss]];
    END;
```

```
HtiForName: PROCEDURE [bcd: BcdBase, n: NameRecord] RETURNS [HTIndex] =
  BEGIN
  RETURN[HtiName[LOOPHOLE[bcd+bcd.ssOffset], n]];
  END;

HtiName: PROCEDURE [ssb: STRING, n: NameRecord] RETURNS [HTIndex] =
  BEGIN
  ss: SubStringDescriptor ← [base: ssb, offset: n.offset, length: n.length];
  RETURN[BcdTabDefs.FindString[@ss].hti];
  END;

NameForHti: PROCEDURE [hti: BcdTabDefs.HTIndex] RETURNS [NameRecord] =
  BEGIN
  ss: SubStringDescriptor;
  BcdTabDefs.SubStringForHash[@ss, hti];
  RETURN[[ss.offset, ss.length]];
  END;

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

MergeFile: PROCEDURE [bcd: BcdBase, oldfti: FTIndex]
  RETURNS [fti: FTIndex] =
  BEGIN OPEN BcdTableDefs;
  oldftb: CARDINAL = LOOPHOLE[bcd+bcd.ftOffset];
  ftLimit: FTIndex = LOOPHOLE[TableBounds[fttype].size];
  fn: NameRecord;
  IF oldfti = FTSelf THEN RETURN[WhoIsFTSelf[]];
  fn ← MapEquivalentName[bcd, (oldftb+oldfti).name];
  FOR fti ← FIRST[FTIndex], fti+SIZE[FTRecord]
  UNTIL fti = ftLimit DO
    OPEN new: ftb+fti, old: oldftb+oldfti;
    IF new.name = fn THEN
      BEGIN
      SELECT TRUE FROM
        (new.version = NullVersion) =>
          BEGIN new.version ← old.version; RETURN END;
        EquivalentVersions[@new.version, @old.version],
        (old.version = NullVersion) =>
          BEGIN
          IF old.version.zapped THEN new.version.zapped ← TRUE;
          RETURN
          END;
        ENDCASE;
      END;
    ENDLOOP;
  fti ← Allocate[fttype, SIZE[FTRecord]];
  (ftb+fti)↑ ← [name: fn, version: (oldftb+oldfti).version];
  RETURN
  END;
```

```
  nextDummyGfi: GFTIndex;

  GetDummyGfi: PROCEDURE [n: CARDINAL] RETURNS [gfi: GFTIndex] =
    BEGIN
    gfi ← nextDummyGfi;
    nextDummyGfi ← nextDummyGfi + n;
    RETURN
    END;

  nextGfi: GFTIndex;

  GetGfi: PROCEDURE [n: CARDINAL] RETURNS [gfi: GFTIndex] =
    BEGIN
    gfi ← nextGfi;
    nextGfi ← nextGfi + n;
    RETURN
    END;

  MergeModule: PUBLIC PROCEDURE [frame, copied: GlobalFrameHandle, initialGFT: LoadStateGFT] =
    BEGIN OPEN BcdTableDefs;
    ccgfi: GFTIndex;
    mti, newmti: MTIndex;
    i: CARDINAL;
    ccgfi ← initialGFT[copied.gftindex.gftindex].gfi;
    FOR mti ← FIRST[MTIndex], mti + SIZE[MTRecord] + (mtb+mti).frame.length - 1
    UNTIL mti = LOOPHOLE[TableBounds[mttype].size, MTIndex] DO
      IF (mtb+mti).gfi = ccgfi THEN EXIT;
      REPEAT FINISHED => ERROR;
      ENDLOOP;
    newmti ← Allocate[mttype, SIZE[MTRecord] + (mtb+mti).frame.length - 1
      ! TableOverflow =>
          BEGIN
          ExpandTable[];
          RESUME [table, tablePages*AltoDefs.PageSize];
          END];
    InlineDefs.COPY[
      from: mtb+mti, to: mtb+newmti, nwords: SIZE[MTRecord]+(mtb+mti).frame.length-1];
    (mtb+newmti).namedinstance ← FALSE;
    (mtb+newmti).gfi ← initialGFT[frame.gftindex.gftindex].gfi;
    FOR i IN [0..(mtb+newmti).frame.length) DO
      OPEN new: mtb+newmti, copy: mtb+mti;
      IF new.frame.frag[i].gfi IN [copy.gfi..copy.gfi+copy.ngfi) THEN
        new.frame.frag[i].gfi ← new.gfi + copy.frame.frag[i].gfi - copy.gfi;
      ENDLOOP;
    END;
```

```
MergeModuleTable: PROCEDURE [Reloc: Relocation,
 config: ConfigIndex, initialGFT: LoadStateGFT, code, symbols: BOOLEAN] =
  BEGIN OPEN BcdTableDefs;
  MoveModule: PROCEDURE [oldmtb: CARDINAL, mti: MTIndex] RETURNS [BOOLEAN] =
    BEGIN OPEN old: oldmtb+mti;
    rgfi: GFTIndex ← Reloc[old.gfi];
    GFT: POINTER TO ARRAY [0..1) OF ControlDefs.GFTItem = REGISTER[ControlDefs.GFTreg];
    frame: GlobalFrameHandle ← GFT↑[rgfi].frame;
    newmti: MTIndex;
    i: CARDINAL;
    IF ~FrameDefs.DeletedFrame[rgfi] THEN
      BEGIN OPEN new: mtb+newmti;
      bcdheader.nModules ← bcdheader.nModules + 1;
      newmti ← Allocate[mttype, SIZE[MTRecord]+old.frame.length-1];
      new.name ← MapName[bcd, old.name];
      IF old.namedinstance THEN
        BEGIN
        EnterNameInTable[[module[newmti]], MapName[bcd, FindName[bcd, [module[mti]]]]];
        new.namedinstance ← TRUE
        END
      ELSE new.namedinstance ← FALSE;
      new.file ← MergeFile[bcd, old.file];
      new.cseg ← old.cseg;
      new.sseg ← old.sseg;
      new.cseg.file ←
        IF code THEN FTSelf ELSE new.cseg.file ← MergeFile[bcd, old.cseg.file];
      new.sseg.file ←
        IF symbols THEN FTSelf ELSE new.sseg.file ← MergeFile[bcd, old.sseg.file];
      new.gfi ← initialGFT[Reloc[old.gfi]].gfi;
      new.fsi ← old.fsi;
      new.ngfi ← old.ngfi;
      new.frame ← old.frame;
      InlineDefs.COPY[
        from: frame+old.frame.offset,
        to: @new.frame.frag,
        nwords: old.frame.length];
      FOR i IN [0..old.frame.length) DO
        OPEN link: new.frame.frag[i];
        IF link = UnboundLink OR link = NullLink THEN
          link ← AddImport[old.frame.frag[i]]
        ELSE
          SELECT link.tag FROM
            frame =>
              BEGIN OPEN f: LOOPHOLE[link, GlobalFrameHandle];
              IF FrameDefs.DeletedFrame[link.gfi]
                THEN link ← AddNewImport[old.frame.frag[i]]
                ELSE link.gfi ← initialGFT[f.gftindex.gftindex].gfi;
              END;
            procedure =>
              IF FrameDefs.DeletedFrame[link.gfi]
                THEN link ← AddNewImport[old.frame.frag[i]]
                ELSE link.gfi ← initialGFT[link.gfi].gfi;
            ENDCASE;
        ENDLOOP;
      IF old.config # CTNull THEN
        BEGIN
        new.config ← old.config+configOffset;
        IF (new.config+ctb).control = mti THEN (new.config+ctb).control ← newmti;
        END
      ELSE new.config ← CTNull;
      END
    ELSE IF old.config # CTNull AND (old.config+ctb+configOffset).control = mti THEN
      (old.config+ctb+configOffset).control ← MTNull;
    RETURN[FALSE];
    END;
  [] ← EnumerateModuleTable[bcd, MoveModule];
  END;
```

```
MergeExportTable: PROCEDURE [Reloc: Relocation, initialGFT: LoadStateGFT] =
  BEGIN
  MapExport: PROCEDURE [oldetb: CARDINAL, eti: EXPIndex] RETURNS [BOOLEAN] =
    BEGIN OPEN old: oldetb+eti;
    neweti: EXPIndex;
    oldftb: CARDINAL ← LOOPHOLE[bcd+bcd.ftOffset];
    oldssb: STRING ← LOOPHOLE[bcd+bcd.ssOffset];
    i, size: CARDINAL;
    found: BOOLEAN;
    hti: HTIndex;
    oldname: StringDefs.SubStringDescriptor;
    oldname ← [base: oldssb, offset: old.name.offset, length: old.name.length];
    [found, hti] ← BcdTabDefs.FindString[@oldname];
    IF found THEN
      FOR neweti ← FIRST[EXPIndex], neweti + size
      UNTIL neweti = LOOPHOLE[BcdTableDefs.TableBounds[BcdTableDefs.exptype].size] DO
        OPEN new: etb+neweti;
        size ← new.size+SIZE[EXPRecord]-1;
        IF hti = HtiName[ssb, new.name] AND new.port = old.port THEN
          BEGIN OPEN oldfile: oldftb+old.file, newfile: ftb+new.file;
          oldname.offset ← oldfile.name.offset;  oldname.length ← oldfile.name.length;
          IF BcdTabDefs.FindEquivalentString[@oldname].found AND
           EquivalentVersions[@oldfile.version, @newfile.version] THEN
            BEGIN
            FOR i IN [0..old.size) DO
              IF old.links[i].gfi # 0 THEN  -- assumes that most recently loaded config
                                            -- merged last
                BEGIN
                new.links[i] ← old.links[i];
                new.links[i].gfi ← initialGFT[Reloc[old.links[i].gfi]].gfi;
                END;
              ENDLOOP;
            IF ~new.namedinstance AND old.namedinstance THEN
              BEGIN
              new.namedinstance ← TRUE;
              EnterNameInTable[[export[neweti]],
                MapName[bcd, FindName[bcd, [export[eti]]]]];
              END;
            RETURN[FALSE];
          END;
        END;
      ENDLOOP;
    [] ← MakeNewExport[oldetb, eti, Reloc, initialGFT];
    RETURN[FALSE];
    END;
  [] ← EnumerateExportTable[bcd, MapExport];
  END;

MakeNewExport: PROCEDURE [
 oldetb: CARDINAL, eti: EXPIndex, Reloc: Relocation, initialGFT: LoadStateGFT]
 RETURNS [neweti: EXPIndex] =
  BEGIN OPEN old: oldetb+eti, new: etb+neweti;
  i: CARDINAL;
  bcdheader.nExports ← bcdheader.nExports + 1;
  neweti ← BcdTableDefs.Allocate[BcdTableDefs.exptype, old.size+SIZE[EXPRecord]-1];
  FOR i IN [0..old.size) DO
    new.links[i] ← old.links[i];
    new.links[i].gfi ← initialGFT[Reloc[old.links[i].gfi]].gfi;
    ENDLOOP;
  new.name ← MapName[bcd, old.name]; new.file ← MergeFile[bcd, old.file];
  new.port ← old.port;   new.size ← old.size;
  IF old.namedinstance THEN
    BEGIN
    new.namedinstance ← TRUE;
    EnterNameInTable[[export[neweti]], MapName[bcd, FindName[bcd, [export[eti]]]]];
    END
  ELSE new.namedinstance ← FALSE;
  END;
```

```
AddImport: PROCEDURE [link: ControlLink] RETURNS [ControlLink] =
  BEGIN
  FindImport: PROCEDURE [olditb: CARDINAL, iti: IMPIndex] RETURNS [BOOLEAN] =
    BEGIN OPEN imp: olditb+iti;
    RETURN [link.gfi IN [imp.gfi .. imp.gfi+imp.ngfi)];
    END;
  iti, newiti: IMPIndex;
  oldftb: CARDINAL ← LOOPHOLE[bcd+bcd.ftOffset];
  olditb: CARDINAL ← LOOPHOLE[bcd+bcd.impOffset];
  oldssb: STRING ← LOOPHOLE[bcd+bcd.ssOffset];
  oldname: SubStringDescriptor;
  found: BOOLEAN;
  hti: HTIndex;
  iti ← EnumerateImportTable[bcd, FindImport];
  IF iti = BcdDefs.IMPNull THEN RETURN[AddNewImport[link]];
  BEGIN OPEN  new: itb+newiti, old: olditb+iti;
    oldname ← [oldssb, old.name.offset, old.name.length];
    [found, hti] ← BcdTabDefs.FindString[@oldname];
    IF found THEN
      FOR newiti ← FIRST[IMPIndex], newiti + SIZE[IMPRecord]
      UNTIL newiti = LOOPHOLE[BcdTableDefs.TableBounds[BcdTableDefs.imptype].size] DO
        IF hti = HtiName[ssb, new.name] THEN
          BEGIN OPEN oldfile: oldftb+old.file, newfile: ftb+new.file;
          oldname.offset ← oldfile.name.offset;  oldname.length ← oldfile.name.length;
          IF BcdTabDefs.FindEquivalentString[@oldname].found AND
           EquivalentVersions[@oldfile.version, @newfile.version]  THEN
            BEGIN
            IF ~new.namedinstance AND old.namedinstance THEN
              BEGIN
              new.namedinstance ← TRUE;
              EnterNameInTable[[import[newiti]],
                MapName[bcd, FindName[bcd, [import[iti]]]]];
              END;
            link.gfi ← new.gfi + link.gfi - old.gfi;
            RETURN[link];
            END;
          END;
        ENDLOOP;
    bcdheader.nImports ← bcdheader.nImports + 1;
    newiti ← BcdTableDefs.Allocate[BcdTableDefs.imptype, SIZE[IMPRecord]];
    new.name ← MapName[bcd, old.name];
    new.file ← MergeFile[bcd, old.file];
    IF old.namedinstance THEN
      BEGIN
      new.namedinstance ← TRUE;
      EnterNameInTable[[import[newiti]], MapName[bcd, FindName[bcd, [import[iti]]]]];
      END
    ELSE new.namedinstance ← FALSE;
    new.gfi ← GetDummyGfi[new.ngfi ← old.ngfi];
    new.port ← old.port;
    link.gfi ← new.gfi + link.gfi - old.gfi;
    END;  -- of OPEN
  RETURN[link];
  END;

AddNewImport: PROCEDURE [link: ControlLink] RETURNS [ControlLink] =
  BEGIN
  link.gfi ← 0;
  link.ep ← 0;
  RETURN[link];  -- Currently cannot make a new import
  END;
```

```
MergeConfigTable: PROCEDURE [size: CARDINAL] RETURNS [delta: CARDINAL] =
  BEGIN
  ConfigMap: PROCEDURE [oldctb: CARDINAL, cti: CTIndex] RETURNS [BOOLEAN] =
    BEGIN
    OPEN new: delta+ctb+cti, old: oldctb+cti;
    new.name ← MapName[bcd, old.name];
    IF old.namedinstance THEN
      BEGIN
      EnterNameInTable[[config[delta+cti]], MapName[bcd, FindName[bcd, [config[cti]]]]];
      new.namedinstance ← TRUE
      END
    ELSE new.namedinstance ← FALSE;
    new.control ← old.control;
    new.file ← IF old.file = FTSelf THEN FTSelf ELSE MergeFile[bcd, old.file];
    new.config ← IF old.config = CTNull THEN CTNull ELSE old.config+delta;
    RETURN[FALSE];
    END;

  delta ← LOOPHOLE[BcdTableDefs.Allocate[BcdTableDefs.cttype, size]];
  bcdheader.nConfigs ← bcdheader.nConfigs + bcd.nConfigs;
  [] ← EnumerateConfigTable[bcd, ConfigMap];
  RETURN
  END;

EnterNameInTable: PROCEDURE [owner: Namee, name: NameRecord] =
  BEGIN
  nti: NTIndex ← BcdTableDefs.Allocate[BcdTableDefs.nttype, SIZE[NTRecord]];
  (ntb+nti)↑ ← [name, owner];
  END;

bcdFile: FTIndex;
name: STRING;

WhoIsFTSelf: PROCEDURE RETURNS [FTIndex] =
  BEGIN
  ss: StringDefs.SubStringDescriptor ← [base: name, offset: 0, length: name.length];
  IF bcdFile = BcdDefs.FTNull THEN
    BEGIN
    bcdFile ← BcdTableDefs.Allocate[BcdTableDefs.fttype, SIZE[BcdDefs.FTRecord]];
    (ftb+bcdFile)↑ ← [EnterName[@ss], bcd.version];
    END;
  RETURN[bcdFile];
  END;

MergeBcd: PUBLIC PROCEDURE [mergee: BcdBase, RealFromRel: Relocation, config: ConfigIndex,
 initialGFT: LoadStateGFT, code, symbols: BOOLEAN, bcdname: STRING] =
  BEGIN
  BEGIN
    ENABLE BcdTableDefs.TableOverflow =>
      BEGIN
      ExpandTable[];
      RESUME [table, tablePages*AltoDefs.PageSize];
      END;
  bcd ← mergee;
  bcdFile ← FTNull;
  name ← bcdname;
  configOffset ← IF bcd.nConfigs = 0 THEN 0
    ELSE MergeConfigTable[LOOPHOLE[bcd.ctLimit, CARDINAL]];
  MergeModuleTable[RealFromRel, config, initialGFT, code, symbols];
  MergeExportTable[RealFromRel, initialGFT];
  END;
  END;

MergedBcdSize: PUBLIC PROCEDURE RETURNS [size: CARDINAL] =
  BEGIN OPEN bcdheader, BcdTableDefs;
  s: CARDINAL;
  size ← SIZE[BCD];
  ssOffset ← size;  size ← size + (ssLimit ← TableBounds[sstype].size);
  ctOffset ← size;  size ← size + (s ← TableBounds[cttype].size);
  ctLimit ← LOOPHOLE[s, CTIndex];
  mtOffset ← size;  size ← size + (s ← TableBounds[mttype].size);
  mtLimit ← LOOPHOLE[s, MTIndex];
  impOffset ← size;  size ← size + (s ← TableBounds[imptype].size);
  impLimit ← LOOPHOLE[s, IMPIndex];
  expOffset ← size;  size ← size + (s ← TableBounds[exptype].size);
```

```
    expLimit ← LOOPHOLE[s, EXPIndex];
    ftOffset ← size;  size ← size + (s ← TableBounds[fttype].size);
    ftLimit ← LOOPHOLE[s, FTIndex];
    ntOffset ← size;  size ← size + (s ← TableBounds[nttype].size);
    ntLimit ← LOOPHOLE[s, NTIndex];
    nPages ← SystemDefs.PagesForWords[size];
    nDummies ← GetDummyGfi[0]-firstdummy;
    END;
```

```
  WriteMergedBcd: PUBLIC PROCEDURE [movewords: PROCEDURE [POINTER, CARDINAL]] =
    BEGIN  OPEN BcdTableDefs;
    base: TableBase;
    size: CARDINAL;
    movewords[@bcdheader, SIZE[BcdDefs.BCD]];
    [base, size] ← TableBounds[sstype];
    movewords[LOOPHOLE[base], size];
    [base, size] ← TableBounds[cttype];
    movewords[LOOPHOLE[base], size];
    [base, size] ← TableBounds[mttype];
    movewords[LOOPHOLE[base], size];
    [base, size] ← TableBounds[imptype];
    movewords[LOOPHOLE[base], size];
    [base, size] ← TableBounds[exptype];
    movewords[LOOPHOLE[base], size];
    [base, size] ← TableBounds[fttype];
    movewords[LOOPHOLE[base], size];
    [base, size] ← TableBounds[nttype];
    movewords[LOOPHOLE[base], size];
    END;

  -- Administrative Procedures

  table: CARDINAL;
  expandedtable: BOOLEAN ← FALSE;
  tableSegment: SegmentDefs.FileSegmentHandle;
  tablePages: CARDINAL;

  InitializeMerge: PUBLIC PROCEDURE [sizeoftable: CARDINAL, lastrealgfi: GFTIndex] =
    BEGIN OPEN BcdTableDefs;
    time: AltoFileDefs.TIME;
    net: CARDINAL ← MiscDefs.GetNetworkNumber[];
    tablePages ← SystemDefs.PagesForWords[sizeoftable];
    table ← LOOPHOLE[SystemDefs.AllocatePages[tablePages], CARDINAL];
    InitializeTable[table, tablePages*AltoDefs.PageSize];
    AddNotify[Notifier];
    BcdTabDefs.BcdTabInit[];
    nextGfi ← 1;
    MiscDefs.Zero[@bcdheader, SIZE[BcdDefs.BCD]];
    bcdheader.firstdummy ← nextDummyGfi ← lastrealgfi+1;
    bcdheader.versionident ← BcdDefs.VersionID;
    time ← MiscDefs.DAYTIME[];
    bcdheader.version ← BcdDefs.VersionStamp[
      time: TimeDefs.PackedTime[lowbits: time.low, highbits: time.high],
      zapped: FALSE,
      net: net,
      host: OsStaticDefs.OsStatics.SerialNumber];
    bcdheader.definitions ← FALSE;
    RETURN
    END;

  FinalizeMerge: PUBLIC PROCEDURE =
    BEGIN OPEN BcdTableDefs;
    BcdTabDefs.BcdTabErase[];
    DropNotify[Notifier];
    EraseTable[];
    IF expandedtable THEN
      BEGIN OPEN SegmentDefs;
      Unlock[tableSegment];
      DeleteFileSegment[tableSegment]
      END
    ELSE SystemDefs.FreeSegment[LOOPHOLE[table]];
    RETURN
    END;

  ExpandTable: PROCEDURE =
    BEGIN OPEN SegmentDefs;
    IF ~expandedtable THEN
      BEGIN
      tableSegment ← NewFileSegment[
        NewFile["swatee", Read+Write+Append, DefaultVersion],
        1,
        tablePages,
        Read+Write];
      ChangeDataToFileSegment[VMtoDataSegment[LOOPHOLE[table]], tableSegment];
```

```
      expandedtable ← TRUE;
      END;
    Unlock[tableSegment];  SwapOut[tableSegment];
    MoveFileSegment[tableSegment, DefaultBase, tablePages ← tablePages + 1];
    SwapIn[tableSegment];
    table ← LOOPHOLE[FileSegmentAddress[tableSegment], CARDINAL];
    END;

  END.
```