```
-- BcdMerge.Mesa  Edited by Sandman on May 12, 1978  8:47 AM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [PageSize],
  AltoFileDefs: FROM "altofiledefs" USING [TIME],
  BcdDefs: FROM "bcddefs" USING [
    BCD, BinderNTables, ControlLink, CTHandle, CTIndex, CTNull, CTRecord,
    cttype, EXPHandle, EXPIndex, EXPRecord, exptype, FTIndex, FTNull,
    FTRecord, FTSelf, fttype, GFTIndex, IMPHandle, IMPIndex, IMPNull,
    IMPRecord, imptype, MTHandle, MTIndex, MTNull, MTRecord, mttype, Namee,
    NameRecord, NameString, NTIndex, NTRecord, nttype, NullLink, NullName,
    SGHandle, SGIndex, SGRecord, sgtype, sstype, UnboundLink, VersionID,
    VersionStamp],
  BcdMergeDefs: FROM "bcdmergedefs" USING [MergeData, MergeDataHandle],
  BcdTabDefs: FROM "bcdtabdefs" USING [
    BcdTabErase, BcdTabInit, EnterString, FindEquivalentString, FindString,
    HTIndex, SubStringForHash],
  ControlDefs: FROM "controldefs" USING [GFT, GlobalFrameHandle],
  FrameDefs: FROM "framedefs" USING [
    DeletedFrame, LockCode, SwapOutCode, UnlockCode],
  InlineDefs: FROM "inlinedefs" USING [COPY],
  LoaderBcdUtilDefs: FROM "loaderbcdutildefs" USING [
    BcdBase, EnumerateConfigTable, EnumerateExportTable, EnumerateImportTable,
    EnumerateModuleTable, FindName],
  LoadStateDefs: FROM "loadstatedefs" USING [
    ConfigIndex, LoadStateGFT, Relocation],
  MiscDefs: FROM "miscdefs" USING [DAYTIME, GetNetworkNumber, Zero],
  OsStaticDefs: FROM "osstaticdefs" USING [OsStatics],
  SegmentDefs: FROM "segmentdefs" USING [
    Append, ChangeDataToFileSegment, DefaultBase, DefaultVersion,
    DeleteFileSegment, FileSegmentAddress, MoveFileSegment, NewFile,
    NewFileSegment, Read, SwapIn, SwapOut, Unlock, VMtoDataSegment, Write],
  StringDefs: FROM "stringdefs" USING [SubString, SubStringDescriptor],
  SystemDefs: FROM "systemdefs" USING [
    AllocateHeapNode, AllocatePages, FreeHeapNode, FreeSegment,
    PagesForWords],
  TableDefs: FROM "tabledefs" USING [
    AddNotify, Allocate, DropNotify, EraseTable, InitializeTable, TableBase,
    TableBounds, TableNotifier, TableOverflow],
  TimeDefs: FROM "timedefs" USING [PackedTime];

DEFINITIONS FROM LoadStateDefs, LoaderBcdUtilDefs, BcdDefs;

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

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

  data: BcdMergeDefs.MergeDataHandle;

  Notifier: PRIVATE TableDefs.TableNotifier =
    BEGIN OPEN BcdDefs, data;
    mtb ← base[mttype];
    ftb ← base[fttype];
    ctb ← base[cttype];
    itb ← base[imptype];
    etb ← base[exptype];
    ntb ← base[nttype];
    sgb ← base[sgtype];
    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]];
    END;
```

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

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

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

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

EquivalentVersions: PROCEDURE [v1, v2: POINTER TO 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 data;
  NullVersion: BcdDefs.VersionStamp =
    [zapped: FALSE, net: 0, host: 0, time: [0,0]];
  oldftb: CARDINAL = LOOPHOLE[bcd+bcd.ftOffset];
  ftLimit: FTIndex = LOOPHOLE[TableDefs.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 ← TableDefs.Allocate[fttype, SIZE[FTRecord]];
  (ftb+fti)↑ ← [name: fn, version: (oldftb+oldfti).version];
  RETURN
  END;

MergeSegment: PROCEDURE [bcd: BcdBase, sgh: SGHandle, fti: FTIndex]
  RETURNS [sgi: SGIndex] =
  BEGIN OPEN data;
  sgLimit: SGIndex = LOOPHOLE[TableDefs.TableBounds[sgtype].size];
```

```
    IF fti = FTNull THEN fti ← MergeFile[bcd, sgh.file];
    FOR sgi ← FIRST[SGIndex], sgi+SIZE[SGRecord] UNTIL sgi = sgLimit DO
      OPEN new: sgb+sgi;
      IF new.class = sgh.class AND new.file = fti AND new.base = sgh.base
        AND new.pages = sgh.pages AND new.extraPages = sgh.extraPages
        THEN RETURN;
      ENDLOOP;
    sgi ← TableDefs.Allocate[sgtype, SIZE[SGRecord]];
    (sgb+sgi)↑ ← [class: sgh.class, file: fti, base: sgh.base,
      pages: sgh.pages, extraPages: sgh.extraPages];
    RETURN
    END;


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

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

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

```
MergeModuleTable: PROCEDURE [Reloc: Relocation,
 config: ConfigIndex, initialGFT: LoadStateGFT, code: BOOLEAN] =
  BEGIN OPEN data;
  MoveModule: PROCEDURE [old: MTHandle, mti: MTIndex] RETURNS [BOOLEAN] =
    BEGIN
    new: MTHandle;
    rgfi: GFTIndex ← Reloc[old.gfi];
    frame: GlobalFrameHandle ← ControlDefs.GFT[rgfi].frame;
    newmti: MTIndex;
    segfti: FTIndex;
    oldsgb: CARDINAL ← LOOPHOLE[bcd+bcd.sgOffset];
    IF ~FrameDefs.DeletedFrame[rgfi] THEN
      BEGIN OPEN m: mtb+newmti;
      header.nModules ← header.nModules + 1;
      newmti ← TableDefs.Allocate[mttype, SIZE[MTRecord]+old.frame.length];
      new ← mtb+newmti;
      new↑ ← old↑;
      m.name ← MapName[bcd, old.name];
      IF old.namedinstance THEN
        BEGIN
        EnterNameInTable[[module[newmti]], MapName[bcd, FindName[bcd, [module[mti]]]]];
        m.namedinstance ← TRUE
        END
      ELSE m.namedinstance ← FALSE;
      m.file ← MergeFile[bcd, old.file];
      segfti ← IF code THEN FTSelf ELSE FTNull;
      m.code.sgi ← MergeSegment[bcd, oldsgb+old.code.sgi, segfti];
      m.sseg ← MergeSegment[bcd, oldsgb+old.sseg, FTNull];
      m.gfi ← initialGFT[Reloc[old.gfi]].gfi;
      MergeLinks[frame, newmti, old, initialGFT];
      IF old.config # CTNull THEN
        BEGIN
        m.config ← old.config+configOffset;
        IF (m.config+ctb).control = mti THEN (m.config+ctb).control ← newmti;
        END
      ELSE IF bcd.nConfigs = 0 THEN
        BEGIN
        m.config ←
          TableDefs.Allocate[cttype, SIZE[CTRecord]];
        (ctb+m.config)↑ ← CTRecord[name: m.name, namedinstance: FALSE,
          file: m.file, config: CTNull, control: MTNull];
        END
      ELSE m.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;

MergeLinks: PROCEDURE [frame: GlobalFrameHandle, newmti: MTIndex,
  old: MTHandle, initialGFT: LoadStateGFT] =
  BEGIN OPEN data, m: mtb+newmti;
  linkbase: POINTER TO ControlLink;
  link: ControlLink;
  i: CARDINAL;
  IF frame.codelinks THEN
    BEGIN
    FrameDefs.LockCode[frame];
    linkbase ← frame.code.codebase;
    END
  ELSE linkbase ← LOOPHOLE[frame];
  linkbase ← linkbase - old.frame.length;
  InlineDefs.COPY[
    from: linkbase, to: @m.frame.frag, nwords: old.frame.length];
  FOR i IN [0..old.frame.length) DO
    link ← (linkbase+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.gfi].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 frame.codelinks THEN
    BEGIN
    FrameDefs.UnlockCode[frame];
    IF ~frame.started THEN FrameDefs.SwapOutCode[frame];
    END;
  END;
```

```
MergeExportTable: PROCEDURE [Reloc: Relocation, initialGFT: LoadStateGFT] =
  BEGIN OPEN data;
  MapExport: PROCEDURE [old: EXPHandle, eti: EXPIndex] RETURNS [BOOLEAN] =
    BEGIN
    neweti: EXPIndex;
    oldftb: TableDefs.TableBase ← LOOPHOLE[bcd+bcd.ftOffset];
    oldssb: NameString ← LOOPHOLE[bcd+bcd.ssOffset];
    i, size: CARDINAL;
    found: BOOLEAN;
    hti: HTIndex;
    oldname: StringDefs.SubStringDescriptor;
    oldname ←
      [base: @oldssb.string, offset: old.name, length: oldssb.size[old.name]];
    [found, hti] ← BcdTabDefs.FindString[@oldname];
    IF found THEN
      FOR neweti ← FIRST[EXPIndex], neweti + size
      UNTIL neweti =
        LOOPHOLE[TableDefs.TableBounds[exptype].size]
        DO OPEN new: etb+neweti;
        size ← new.size+SIZE[EXPRecord];
        IF hti = HtiName[new.name] AND new.port = old.port THEN
          BEGIN OPEN oldfile: oldftb+old.file, newfile: ftb+new.file;
          oldname.offset ← oldfile.name+1;
          oldname.length ← oldssb.size[oldfile.name];
          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[old, eti, Reloc, initialGFT];
    RETURN[FALSE];
    END;
  [] ← EnumerateExportTable[bcd, MapExport];
  END;

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

```
AddImport: PROCEDURE [link: ControlLink] RETURNS [ControlLink] =
  BEGIN OPEN data;
  FindImport: PROCEDURE [imp: IMPHandle, iti: IMPIndex] RETURNS [BOOLEAN] =
    BEGIN
    RETURN [link.gfi IN [imp.gfi .. imp.gfi+imp.ngfi)];
    END;
  old: IMPHandle;
  iti, newiti: IMPIndex;
  oldftb: CARDINAL ← LOOPHOLE[bcd+bcd.ftOffset];
  olditb: CARDINAL ← LOOPHOLE[bcd+bcd.impOffset];
  oldssb: NameString ← LOOPHOLE[bcd+bcd.ssOffset];
  oldname: SubStringDescriptor;
  found: BOOLEAN;
  hti: HTIndex;
  iti ← EnumerateImportTable[bcd, FindImport].iti;
  IF iti = BcdDefs.IMPNull THEN RETURN[AddNewImport[link]];
  old ← olditb+iti;
  oldname ← [@oldssb.string, old.name, oldssb.size[old.name]];
  [found, hti] ← BcdTabDefs.FindString[@oldname];
  IF found THEN
    FOR newiti ← FIRST[IMPIndex], newiti + SIZE[IMPRecord]
    UNTIL newiti =
      LOOPHOLE[TableDefs.TableBounds[imptype].size]
      DO OPEN new: itb+newiti;
      IF hti = HtiName[new.name] THEN
        BEGIN OPEN oldfile: oldftb+old.file, newfile: ftb+new.file;
        oldname.offset ← oldfile.name;
        oldname.length ← oldssb.size[oldfile.name];
        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;
  header.nImports ← header.nImports + 1;
  newiti ← TableDefs.Allocate[imptype, SIZE[IMPRecord]];
  (itb+newiti).name ← MapName[bcd, old.name];
  (itb+newiti).file ← MergeFile[bcd, old.file];
  IF old.namedinstance THEN
    BEGIN
    (itb+newiti).namedinstance ← TRUE;
    EnterNameInTable[[import[newiti]], MapName[bcd, FindName[bcd, [import[iti]]]]];
    END
  ELSE (itb+newiti).namedinstance ← FALSE;
  (itb+newiti).gfi ← GetDummyGfi[(itb+newiti).ngfi ← old.ngfi];
  (itb+newiti).port ← old.port;
  link.gfi ← (itb+newiti).gfi + link.gfi - old.gfi;
  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 OPEN data;
  ConfigMap: PROCEDURE [old: CTHandle, cti: CTIndex] RETURNS [BOOLEAN] =
    BEGIN OPEN new: ctb+delta+cti;
    new.name ← MapName[bcd, old.name];
    IF old.namedinstance THEN
      BEGIN
      EnterNameInTable[[config[cti+delta]], 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[TableDefs.Allocate[cttype, size]];
  header.nConfigs ← header.nConfigs + bcd.nConfigs;
  [] ← EnumerateConfigTable[bcd, ConfigMap];
  RETURN
  END;

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

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

MergeBcd: PUBLIC PROCEDURE [mergee: BcdBase, RealFromRel: Relocation, config: ConfigIndex,
 initialGFT: LoadStateGFT, code: BOOLEAN, bcdname: STRING] =
  BEGIN OPEN data;
  BEGIN
    ENABLE TableDefs.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];
  MergeExportTable[RealFromRel, initialGFT];
  END;
  END;

MergedBcdSize: PUBLIC PROCEDURE RETURNS [size: CARDINAL] =
  BEGIN OPEN data.header, BcdDefs, TableDefs;
  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];
```

```
    sgOffset ← size;  size ← size + (s ← TableBounds[sgtype].size);
    sgLimit ← LOOPHOLE[s, SGIndex];
    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 BcdDefs;
    base: TableDefs.TableBase;
    size: CARDINAL;
    movewords[@data.header, SIZE[BcdDefs.BCD]];
    [base, size] ← TableDefs.TableBounds[sstype];
    movewords[LOOPHOLE[base], size];
    [base, size] ← TableDefs.TableBounds[cttype];
    movewords[LOOPHOLE[base], size];
    [base, size] ← TableDefs.TableBounds[mttype];
    movewords[LOOPHOLE[base], size];
    [base, size] ← TableDefs.TableBounds[imptype];
    movewords[LOOPHOLE[base], size];
    [base, size] ← TableDefs.TableBounds[exptype];
    movewords[LOOPHOLE[base], size];
    [base, size] ← TableDefs.TableBounds[sgtype];
    movewords[LOOPHOLE[base], size];
    [base, size] ← TableDefs.TableBounds[fttype];
    movewords[LOOPHOLE[base], size];
    [base, size] ← TableDefs.TableBounds[nttype];
    movewords[LOOPHOLE[base], size];
    END;

  -- Administrative Procedures

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

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

```
  ExpandTable: PROCEDURE =
    BEGIN OPEN SegmentDefs, data;
    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.
```