```
-- BcdBind.mesa; edited by Johnsson on August 30, 1978  10:15 PM

DIRECTORY
  BcdControlDefs: FROM "bcdcontroldefs",
  BcdDefs: FROM "bcddefs",
  BcdBindDefs: FROM "bcdbinddefs",
  BcdErrorDefs: FROM "bcderrordefs",
  BcdHeapDefs: FROM "bcdheapdefs",
  BcdTabDefs: FROM "bcdtabdefs",
  BcdTreeDefs: FROM "bcdtreedefs",
  BcdUtilDefs: FROM "bcdutildefs",
  InlineDefs: FROM "inlinedefs",
  SegmentDefs: FROM "segmentdefs",
  StringDefs: FROM "stringdefs",
  TableDefs: FROM "tabledefs";

DEFINITIONS FROM BcdDefs, BcdTabDefs, BcdTreeDefs;

BcdBind: PROGRAM [data: BcdControlDefs.BinderData]
  IMPORTS BcdBindDefs, BcdErrorDefs, BcdHeapDefs, BcdTabDefs, BcdTreeDefs, BcdUtilDefs, TableDefs
  EXPORTS BcdControlDefs =
  BEGIN

  BindError: PUBLIC SIGNAL = CODE;
  tb, stb, ctb, cxb, mtb, etb, itb, ftb: TableDefs.TableBase;
  ssb: BcdDefs.NameString;

  Notifier: TableDefs.TableNotifier =
    BEGIN
    tb ← base[treetype];
    stb ← base[sttype];
    ctb ← base[cttype];
    cxb ← base[cxtype];
    mtb ← base[mttype];
    etb ← base[exptype];
    itb ← base[imptype];
    ftb ← base[fttype];
    ssb ← LOOPHOLE[base[sstype]];
    RETURN
    END;


  error: PROCEDURE = BEGIN SIGNAL BindError END;


  BindRoot: PUBLIC PROCEDURE =
    BEGIN
    TableDefs.AddNotify[Notifier];
    relocationHead ← BcdBindDefs.GetRelocationHead[];
    SetupGFMap[];
    rel ← NIL;
    AssignImports[
      ! BcdErrorDefs.GetSti =>
        IF rel # NIL THEN
          RESUME[StiForContext[
            IF rel.type = inner THEN rel.parentcx ELSE rel.context]]];
    BindModules[];
    ReleaseGFMap[];
    TableDefs.DropNotify[Notifier];
    RETURN
    END;

  LinkType: TYPE = RECORD [
    SELECT tag:* FROM
      gfi => [gfi: GFTIndex],
      import => [impi: IMPIndex],
      ENDCASE];

  GFMapItem: TYPE = RECORD [
    linkitem: LinkType,
    expi: EXPIndex,
    offset: [0..4)];

  gfMap: DESCRIPTOR FOR ARRAY OF GFMapItem;
  relMap: DESCRIPTOR FOR ARRAY OF CARDINAL;
```

```
SetupGFMap: PROCEDURE =
  BEGIN
  i: CARDINAL;
  rel: POINTER TO BcdRelocations;
  iti: IMPIndex;
  nDummies: CARDINAL ← BcdUtilDefs.GetDummyGfi[0]-1;
  nImports: CARDINAL = TableDefs.TableBounds[imptype].size/SIZE[IMPRecord];
  p: POINTER;
  IF nDummies = 0 THEN p ← NIL
  ELSE
    BEGIN
    nDummies ← nDummies + 1;
    p ← BcdHeapDefs.GetSpace[nDummies*SIZE[GFMapItem]];
    END;
  gfMap ← DESCRIPTOR[p, nDummies];
  FOR i IN [0..nDummies) DO gfMap[i] ← [[gfi[0]],EXPNull,0] ENDLOOP;
  p ← IF nImports = 0 THEN NIL ELSE BcdHeapDefs.GetSpace[nImports];
  relMap ← DESCRIPTOR[ p, nImports];
  FOR rel ← relocationHead, rel.link UNTIL rel = NIL DO
    FOR iti ← FIRST[IMPIndex]+rel.import, iti+SIZE[IMPRecord]
      UNTIL iti = rel.importLimit DO
      OPEN imp: itb+iti;
      relMap[LOOPHOLE[iti,CARDINAL]/SIZE[IMPRecord]] ←
        imp.gfi+ rel.dummygfi-rel.originalfirstdummy;
      ENDLOOP;
    ENDLOOP;
  RETURN
  END;

RelocatedGfi: PROCEDURE [iti: IMPIndex] RETURNS [CARDINAL] =
  BEGIN
  IF iti = IMPNull THEN RETURN[0];
  RETURN[relMap[LOOPHOLE[iti,CARDINAL]/SIZE[IMPRecord]]]
  END;

ReleaseGFMap: PROCEDURE =
  BEGIN
  IF LENGTH[gfMap]#0 THEN BcdHeapDefs.FreeSpace[BASE[gfMap]];
  IF LENGTH[relMap]#0 THEN BcdHeapDefs.FreeSpace[BASE[relMap]];
  END;


NameToHti: PROCEDURE [name: NameRecord] RETURNS [hti: HTIndex] =
  BEGIN OPEN BcdTabDefs;
  found: BOOLEAN;
  ss: StringDefs.SubStringDescriptor ←
    [base: @ssb.string, offset: name, length: ssb.size[name]];
  [found,hti] ← FindString[@ss];
  IF ~found THEN error[];
  RETURN
  END;

ExpiForSti: PROCEDURE [sti: STIndex] RETURNS [EXPIndex] =
  BEGIN OPEN BcdTabDefs;
  IF sti = STNull THEN RETURN[EXPNull];
  WITH s:stb+sti SELECT FROM
    external =>
      WITH m:s.map SELECT FROM
        interface => RETURN[m.expi];
        ENDCASE;
    ENDCASE;
  RETURN[EXPNull]
  END;

CopyVariantPartOfSTRecord: PROCEDURE [to, from: STIndex] =
  BEGIN   -- implements (stb+to).body ← (stb+from).body;
  -- assignment to dummy assures that this procedure will be noticed
  -- when STRecords change
  dummy: STRecord ← [
    filename:, assigned:, hti:, imported:, exported:,
    link:, impi:, impgfi:, body:];
  s: POINTER TO STRecord = stb+to;
  dummy ← s↑;
  s↑ ← (stb+from)↑;
  s.filename ← dummy.filename;
  s.assigned ← dummy.assigned;
```

```
    s.hti ← dummy.hti;
    s.imported ← dummy.imported;
    s.exported ← dummy.exported;
    s.link ← dummy.link;
    s.impi ← dummy.impi;
    s.impgfi ← dummy.impgfi;
    END;

  AssignImports: PROCEDURE =
    BEGIN
    saveIndex: CARDINAL = data.textIndex;
    saveName: NameRecord = data.currentname;
    iti, import: IMPIndex;
    export: EXPIndex;
    defgfi: CARDINAL;
    sti, parentsti: STIndex;
    FOR rel ← relocationHead, rel.link UNTIL rel = NIL DO
      data.textIndex ← rel.textIndex;
      data.currentname ← BcdUtilDefs.NameForSti[StiForContext[rel.context]];
      IF rel.parameters # empty THEN AssignByParameters[rel]
      ELSE FOR iti ← FIRST[IMPIndex]+rel.import, iti+SIZE[IMPRecord]
        UNTIL iti = rel.importLimit DO
        BEGIN
        OPEN imp: itb+iti;
        IF (sti ← Lookup[NameToHti[imp.name], rel.context]) = STNull THEN
          GOTO loop;
        defgfi ← (stb+sti).impgfi;
        IF (stb+sti).impi # IMPNull THEN
          BEGIN OPEN s:stb+sti, imp: itb+(stb+sti).impi;
          SELECT rel.type FROM
            outer =>
              BEGIN
              s.impgfi ← imp.gfi ← BcdUtilDefs.GetGfi[imp.ngfi];
              GOTO loop;
              END;
            inner =>
              BEGIN
              parentsti ← LookupImport[iti,rel.parentcx];
              import ← (stb+parentsti).impi;
              defgfi ← (stb+parentsti).impgfi;
              export ← ExpiForSti[parentsti];
              sti ← parentsti;
              END;
            ENDCASE =>
              BEGIN import ← s.impi; export ← ExpiForSti[sti] END;
          END
        ELSE BEGIN import ← IMPNull; export ← ExpiForSti[sti] END;
        WITH s: stb+sti SELECT FROM
          external =>
            WITH m:s.map SELECT FROM
              module => AssignModule[defgfi, m.mti, iti];
              interface =>
                AssignInterface[defgfi, import, export, iti];
              unknown => AssignImport[defgfi, import, iti];
              ENDCASE => error[];
          unknown => AssignImport[defgfi, import, iti];
          ENDCASE => error[];
        EXITS loop=> NULL;
        END;
        ENDLOOP;
      ENDLOOP;
    data.textIndex ← saveIndex;
    data.currentname ← saveName;
    RETURN
    END;

  LookupImport: PROCEDURE [iti: IMPIndex, cxi: CXIndex]
    RETURNS [sti: STIndex] =
    BEGIN OPEN s: stb+sti;
    sti ← STNull;
    IF cxi = CXNull THEN RETURN;
    sti ← Lookup[NameToHti[(itb+iti).name],cxi];
    RETURN
    END;
```

```
AssignByParameters: PROCEDURE[rel: POINTER TO BcdBindDefs.BcdRelocations] =
  BEGIN
  iti: IMPIndex;
  msg: STRING ← "has too many parameters"L;
  TooManyParameters: SIGNAL = CODE;
  AssignOne: BcdTreeDefs.TreeScan =
    BEGIN
    import: IMPIndex;
    export: EXPIndex;
    sti: STIndex;
    defgfi: CARDINAL;
    IF iti = rel.importLimit THEN SIGNAL TooManyParameters;
    WITH t SELECT FROM
      symbol => sti ← index;
      ENDCASE => error[];
    defgfi ← (stb+sti).impgfi;
    import ← (stb+sti).impi;
    export ← ExpiForSti[sti];
    WITH s: stb+sti SELECT FROM
      external =>
        WITH m:s.map SELECT FROM
          module => AssignModule[defgfi, m.mti, iti];
          interface =>
            AssignInterface[defgfi, import, export, iti];
          ENDCASE => AssignImport[defgfi, import, iti];
      ENDCASE => BcdErrorDefs.ErrorSti[error,"is undeclared"L,sti];
    iti ← iti+SIZE[BcdDefs.IMPRecord];
    END;

  iti ← FIRST[IMPIndex] + rel.import;
  BEGIN
  BcdTreeDefs.scanlist[rel.parameters,AssignOne
    ! TooManyParameters => GOTO wrongnumber];
  IF iti # rel.importLimit THEN
    BEGIN msg ← "has too few parameters"L; GOTO wrongnumber END;
  EXITS wrongnumber =>
    BcdErrorDefs.ErrorHti[error,msg,HtiForRelocation[rel]];
  END;
  RETURN;
  END;

MakeLink: PROCEDURE [defgfi: CARDINAL, import: IMPIndex, offset: CARDINAL]
  RETURNS [LinkType] =
  BEGIN
  IF defgfi # 0 THEN RETURN[[gfi[defgfi+offset]]];
  IF import = IMPNull THEN RETURN[[gfi[0]]];
  RETURN[[import[import]]]
  END;

AssignModule: PROCEDURE [defgfi: GFTIndex, mti: MTIndex, iti: IMPIndex] =
  BEGIN OPEN imp: itb+iti;
  gfi: CARDINAL = RelocatedGfi[iti];
  IF imp.port # module OR imp.file # (mtb+mti).file THEN
    BcdErrorDefs.Error2Files[error, "cannot be imported as"L, imp.file, (mtb+mti).file];
  gfMap[gfi] ← [
    linkitem: [gfi[IF defgfi # 0 THEN defgfi ELSE (mtb+mti).gfi]],
    expi: EXPNull, offset: 0];
  END;

AssignInterface: PROCEDURE [defgfi: GFTIndex, import: IMPIndex, expi: EXPIndex, iti: IMPIndex] =
  BEGIN OPEN exp: etb+expi, imp: itb+iti;
  i: [0..4);
  gfi: CARDINAL = RelocatedGfi[iti];
  IF imp.port # exp.port OR imp.file # exp.file THEN
    BcdErrorDefs.Error2Files[error, "cannot be imported as"L, imp.file, (etb+expi).file];
  IF exp.port = module THEN
    gfMap[gfi] ← [
      linkitem: [gfi[(etb+expi).links[0].gfi]],
      expi: EXPNull, offset: 0]
  ELSE FOR i IN [0..imp.ngfi) DO
    gfMap[gfi+i] ← [
      linkitem: MakeLink[defgfi, import, i],
      expi: expi, offset: i];
    ENDLOOP;
  END;
```

```
  AssignImport: PROCEDURE [defgfi: GFTIndex, import: IMPIndex, iti: IMPIndex] =
    BEGIN OPEN imp: itb+iti;
    i: [0..4);
    gfi: CARDINAL = RelocatedGfi[iti];
    FOR i IN [0..imp.ngfi) DO
      gfMap[gfi+i] ← [
        linkitem: MakeLink[defgfi, import, i],
        expi: EXPNull, offset: i];
      ENDLOOP;
    END;

  Lookup: PROCEDURE [hti: HTIndex, cxi: CXIndex] RETURNS [sti: STIndex] =
    BEGIN
    FOR sti ← (cxb+cxi).link, (stb+sti).link UNTIL sti = STNull DO
      IF (stb+sti).hti = hti THEN RETURN;
      ENDLOOP;
    BcdErrorDefs.ErrorHti[error, "is undeclared"L, hti
      ! BcdErrorDefs.GetSti => RESUME[StiForContext[cxi]]];
    RETURN[STNull]
    END;

  StiForContext: PROCEDURE [cxi: CXIndex] RETURNS [sti: STIndex] =
    BEGIN
    stLimit: STIndex = LOOPHOLE[TableDefs.TableBounds[sttype].size];
    FOR sti ← FIRST[STIndex], sti+SIZE[STRecord] UNTIL sti = stLimit DO
      WITH s:stb+sti SELECT FROM
        local => IF s.context = cxi THEN RETURN;
        ENDCASE;
      ENDLOOP;
    RETURN[STNull]
    END;

  HtiForRelocation: PROCEDURE [rel: POINTER TO BcdRelocations] RETURNS [HTIndex] =
    BEGIN
    sti: STIndex;
    mti: MTIndex;
    cti: CTIndex;
    IF rel.type # file THEN
      BEGIN
      sti ← StiForContext[rel.context];
      RETURN[(stb+sti).hti];
      END;
    mti ← FIRST[MTIndex] + rel.module;
    cti ← FIRST[CTIndex] + rel.config;
    RETURN[NameToHti[IF (mtb+mti).config = cti THEN (ctb+cti).name ELSE (mtb+mti).name]];
    END;

  BcdRelocations: TYPE = BcdBindDefs.BcdRelocations;
  relocationHead: POINTER TO BcdRelocations;
  rel: POINTER TO BcdRelocations;

  BindModules: PROCEDURE =
    BEGIN
    saveIndex: CARDINAL = data.textIndex;
    saveName: NameRecord = data.currentname;
    mti: MTIndex;
    mtLimit: MTIndex = LOOPHOLE[TableDefs.TableBounds[mttype].size];
    i: CARDINAL;
    FOR mti ← FIRST[MTIndex], mti+SIZE[MTRecord]+(mtb+mti).frame.length
      UNTIL mti = mtLimit DO
      SetRelocationForModule[mti];
      FOR i IN [0..(mtb+mti).frame.length) DO
        (mtb+mti).frame.frag[i] ← RelocateLink[(mtb+mti).frame.frag[i]
          ! BcdErrorDefs.GetModule => RESUME[mti]];
        ENDLOOP;
      ENDLOOP;
    data.textIndex ← saveIndex;
    data.currentname ← saveName;
    RETURN
    END;

  SetRelocationForModule: PROCEDURE [mti: MTIndex] =
    BEGIN
    gfi: GFTIndex = (mtb+mti).gfi;
    BEGIN
    IF rel # NIL THEN
```

```
      BEGIN
      IF gfi IN [rel.firstgfi..rel.lastgfi] THEN RETURN;
      FOR rel ← rel.link, rel.link UNTIL rel = NIL DO
        IF gfi IN [rel.firstgfi..rel.lastgfi] THEN GOTO found;
        ENDLOOP;
      END;
    FOR rel ← relocationHead, rel.link UNTIL rel = NIL DO
      IF gfi IN [rel.firstgfi..rel.lastgfi] THEN GOTO found;
      ENDLOOP;
    EXITS found =>
      BEGIN
      data.textIndex ← rel.textIndex;
      data.currentname ← BcdUtilDefs.NameForSti[StiForContext[rel.context]];
      RETURN
      END;
    END;
    error[];
    RETURN
    END;

  RelocateLink: PROCEDURE[cl: ControlLink] RETURNS [ControlLink] =
    BEGIN
    newcl: ControlLink;
    gfi: CARDINAL;
    expi: EXPIndex;
    name: NameRecord;
    offset: CARDINAL ← 0;
    map: POINTER TO GFMapItem;
    IF cl.gfi = 0 THEN RETURN[cl];
    IF cl.gfi < rel.originalfirstdummy THEN
      BEGIN cl.gfi ← cl.gfi + rel.firstgfi-1; RETURN[cl] END;
    gfi ← cl.gfi-rel.originalfirstdummy+rel.dummygfi;
    DO
      map ← @gfMap[gfi];
      IF (expi←map.expi) # EXPNull THEN
        BEGIN
        newcl ← (etb+expi).links[cl.ep+map.offset*EPLimit];
        IF newcl # NullLink THEN RETURN[newcl];
        END;
      WITH m:map.linkitem SELECT FROM
        gfi =>
          IF (gfi←m.gfi) # 0 THEN EXIT
          ELSE GOTO unbindable;
        import =>
          gfi ← RelocatedGfi[m.impi]+map.offset;
        ENDCASE;
      REPEAT
        unbindable =>
          BEGIN
          [name,offset] ← ImportName[cl.gfi];
          BcdErrorDefs.ErrorInterface[
            warning, "is unbindable"L,name,cl.ep+offset];
          RETURN[IF cl.tag = frame THEN NullLink ELSE UnboundLink];
          END;
      ENDLOOP;
    cl.gfi ← gfi;
    RETURN[cl]
    END;

  ImportName: PROCEDURE [gfi: GFTIndex] RETURNS [NameRecord, CARDINAL] =
    BEGIN
    iti: IMPIndex;
    FOR iti ← FIRST[IMPIndex]+rel.import, iti+SIZE[IMPRecord] UNTIL iti=rel.importLimit DO
      OPEN imp: itb+iti;
      IF gfi IN [imp.gfi..imp.gfi+imp.ngfi) THEN
        RETURN[imp.name, (gfi-imp.gfi)*EPLimit];
      ENDLOOP;
    RETURN[NullName,0]
    END;

  END...
```

```
-- BcdBindDefs.Mesa  Edited by Johnsson on April 12, 1978  4:29 PM

DIRECTORY
  BcdDefs: FROM "bcddefs",
  BcdTabDefs: FROM "bcdtabdefs",
  BcdTreeDefs: FROM "bcdtreedefs";

DEFINITIONS FROM BcdDefs;

BcdBindDefs: DEFINITIONS =
  BEGIN

  RelocationType: TYPE = {outer, inner, file};

  BcdRelocations: TYPE = RECORD [
    link: POINTER TO BcdRelocations,
    type: RelocationType,
    firstgfi, originalfirstdummy: GFTIndex,
    dummygfi: CARDINAL,
    lastgfi: GFTIndex,
    import, module, config: CARDINAL,
    importLimit: IMPIndex,
    context, parentcx: BcdTabDefs.CXIndex,
    textIndex: CARDINAL,
    parameters: BcdTreeDefs.TreeLink];

  GetRelocationHead: PROCEDURE RETURNS [POINTER TO BcdRelocations];
  END.
```