```
-- BcdUtilities.Mesa  Edited by Johnsson on April 13, 1978  7:54 AM

DIRECTORY
  AltoDefs: FROM "altodefs",
  BcdControlDefs: FROM "bcdcontroldefs",
  BcdDefs: FROM "bcddefs",
  BcdErrorDefs: FROM "bcderrordefs",
  BcdTabDefs: FROM "bcdtabdefs",
  BcdTreeDefs: FROM "bcdtreedefs",
  BcdUtilDefs: FROM "bcdutildefs",
  InlineDefs: FROM "inlinedefs",
  StringDefs: FROM "stringdefs",
  TableDefs: FROM "tabledefs";

DEFINITIONS FROM BcdUtilDefs, BcdDefs;

BcdUtilities: PROGRAM
  IMPORTS BcdErrorDefs, BcdTabDefs, StringDefs, TableDefs
  EXPORTS BcdControlDefs, BcdUtilDefs = PUBLIC
  BEGIN

  STIndex: TYPE = BcdTabDefs.STIndex;
  STNull: STIndex = BcdTabDefs.STNull;
  HTIndex: TYPE = BcdTabDefs.HTIndex;
  HTNull: HTIndex = BcdTabDefs.HTNull;
  SubStringDescriptor: TYPE = StringDefs.SubStringDescriptor;
  SubString: TYPE = StringDefs.SubString;

  NullVersion: VersionStamp ← [
    zapped: FALSE, net: 0, host: 0, time: [0,0]];

  mtb, sgb, ftb, ctb, itb, etb, ntb, stb, cxb: TableDefs.TableBase;

  Notifier: PRIVATE TableDefs.TableNotifier =
    BEGIN
    mtb ← base[mttype];
    sgb ← base[sgtype];
    ftb ← base[fttype];
    ctb ← base[cttype];
    itb ← base[imptype];
    etb ← base[exptype];
    ntb ← base[nttype];
    stb ← base[sttype];
    cxb ← base[cxtype];
    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: BcdBasePtr, n: NameRecord] RETURNS [NameRecord] =
    BEGIN
    ss: SubStringDescriptor ←
      [base: @bcd.ssb.string, offset: n, length: bcd.ssb.size[n]];
    RETURN[EnterName[@ss]];
    END;

  MapEquivalentName: PRIVATE PROCEDURE [bcd: BcdBasePtr, n: NameRecord] RETURNS [NameRecord] =
    BEGIN
    ss: SubStringDescriptor ←
      [base: @bcd.ssb.string, offset: n, length: bcd.ssb.size[n]];
    found: BOOLEAN;
    hti: HTIndex;
    [found,hti] ← BcdTabDefs.FindString[@ss];
    IF ~found THEN [found,hti] ← BcdTabDefs.FindEquivalentString[@ss];
    RETURN[[IF found THEN NameForHti[hti] ELSE EnterName[@ss]]]
    END;

  HtiForName: PROCEDURE [bcd: BcdBasePtr, n: NameRecord] RETURNS [HTIndex] =
    BEGIN
    ss: SubStringDescriptor ←
      [base: @bcd.ssb.string, offset: n, length: bcd.ssb.size[n]];
```

```
    RETURN[BcdTabDefs.EnterString[@ss]];
    END;

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

  NameForSti: PROCEDURE [sti: BcdTabDefs.STIndex] RETURNS [NameRecord] =
    BEGIN
    RETURN[NameForHti[(stb+sti).hti]];
    END;

  ContextForTree: PROCEDURE [t: BcdTreeDefs.TreeLink]
    RETURNS [BcdTabDefs.CXIndex] =
    BEGIN
    WITH t SELECT FROM
      symbol =>
        WITH stb+index SELECT FROM
          local => RETURN[context];
          ENDCASE => ERROR;
      ENDCASE => ERROR;
    END;

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

  InsertFile: PROCEDURE [fn: NameRecord, version: POINTER TO VersionStamp]
    RETURNS [fti: FTIndex] =
    BEGIN
    ftLimit: FTIndex = LOOPHOLE[TableDefs.TableBounds[fttype].size];
    mismatched: BOOLEAN ← FALSE;
    FOR fti ← FIRST[FTIndex], fti+SIZE[FTRecord]
    UNTIL fti = ftLimit DO
      OPEN new: ftb+fti;
      IF new.name = fn THEN
        BEGIN
        SELECT TRUE FROM
          (new.version = NullVersion) =>
            BEGIN new.version ← version↑; RETURN END;
          EquivalentVersions[@new.version, version],
          (version↑ = NullVersion) =>
            BEGIN
            IF version.zapped THEN new.version.zapped ← TRUE;
            RETURN
            END;
          ENDCASE => mismatched ← TRUE;
        END;
      ENDLOOP;
    fti ← TableDefs.Allocate[fttype, SIZE[FTRecord]];
    (ftb+fti)↑ ← [name: fn, version: version↑];
    IF mismatched THEN
      BcdErrorDefs.ErrorFile[warning, "referenced in different versions", fti];
    RETURN
    END;

  MergeFile: PROCEDURE [bcd: BcdBasePtr, oldfti: FTIndex]
    RETURNS [FTIndex] =
    BEGIN OPEN TableDefs, old: bcd.ftb+oldfti;
    fn: NameRecord;
    IF oldfti = FTSelf OR oldfti = FTNull THEN RETURN[oldfti];
    fn ← MapEquivalentName[bcd, old.name];
    RETURN[InsertFile[fn, @old.version]]
    END;

  EnterFile: PROCEDURE [name: STRING]
    RETURNS [FTIndex] =
    BEGIN OPEN TableDefs;
    ext: STRING ← [4];
    ss: SubStringDescriptor ← [base: name, offset: 0, length: name.length];
    fn: NameRecord;
```

```
    found: BOOLEAN;
    hti: HTIndex;
    i: CARDINAL;
    IF ss.base[ss.offset+ss.length-1] = '. THEN
      ss.length ← ss.length-1;
    IF ss.length > 4 THEN
      BEGIN
      FOR i IN[0..4) DO
        ext[i] ← ss.base[ss.offset+ss.length-4+i];
        ENDLOOP;
      ext.length ← 4;
      IF StringDefs.EquivalentString[ext, ".bcd"L] THEN
        ss.length ← ss.length-4;
      END;
    [found,hti] ← BcdTabDefs.FindString[@ss];
    IF ~found THEN [found,hti] ← BcdTabDefs.FindEquivalentString[@ss];
    fn ← [IF found THEN NameForHti[hti] ELSE EnterName[@ss]];
    RETURN[InsertFile[fn, @NullVersion]]
    END;

  SetFileVersion: PROCEDURE [fti: FTIndex, v: BcdDefs.VersionStamp] =
    BEGIN OPEN file: ftb+fti;
    SELECT TRUE FROM
      (file.version = NullVersion) => file.version ← v;
      EquivalentVersions[@file.version, @v] =>
        IF v.zapped THEN file.version.zapped ← TRUE;
      ENDCASE =>
        BcdErrorDefs.ErrorFile[warning, "referenced in different versions"L, fti];
    RETURN
    END;

  nextGfi: CARDINAL;
  GftOverflow: PUBLIC SIGNAL = CODE;

  GetGfi: PROCEDURE [n: CARDINAL] RETURNS [gfi: GFTIndex] =
    BEGIN
    gfi ← nextGfi;
    nextGfi ← nextGfi + n;
    IF nextGfi > LAST[GFTIndex] THEN ERROR GftOverflow;
    RETURN
    END;

  nextDummyGfi: CARDINAL;

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

  NewContext: PROCEDURE RETURNS [cxi: BcdTabDefs.CXIndex] =
    BEGIN
    cxi ← TableDefs.Allocate[cxtype, SIZE[BcdTabDefs.CXRecord]];
    (cxb+cxi)↑ ← [link: STNull];
    RETURN
    END;

  NewSemanticEntry: PROCEDURE [hti: HTIndex]
    RETURNS [sti: BcdTabDefs.STIndex] =
    BEGIN
    sti ← TableDefs.Allocate[sttype, SIZE[BcdTabDefs.STRecord]];
    (stb+sti)↑ ← [
      filename: FALSE,
      assigned: FALSE,
      imported: FALSE,
      exported: FALSE,
      hti: BcdTabDefs.HTNull,
      link: BcdTabDefs.STNull,
      impi: IMPNull,
      impgfi: 0,
      body: unknown[]];
    (stb+sti).hti ← hti;
    RETURN
    END;
```

```
EnterModule: PROCEDURE [bcd: BcdBasePtr, oldmti: MTIndex, name: HTIndex]
  RETURNS [mti: MTIndex] =
  BEGIN OPEN old: bcd.mtb+oldmti;
  size: CARDINAL = SIZE[MTRecord]+old.frame.length;
  mti ← TableDefs.Allocate[mttype, size];
  InlineDefs.COPY[to: mtb+mti, from: @old, nwords: size];
  (mtb+mti).name ← MapName[bcd, (mtb+mti).name];
  IF name # HTNull THEN
    BEGIN
    (mtb+mti).namedinstance ← TRUE;
    CreateInstanceName[name, [module[mti]]];
    END
  ELSE IF (mtb+mti).namedinstance THEN
    BEGIN
    CopyInstanceName[bcd, [module[oldmti]], [module[mti]]];
    END;
  RETURN
  END;

EnterSegment: PROCEDURE [seg: SGRecord]
  RETURNS [sgi: SGIndex] =
  BEGIN
  sgLimit: SGIndex = LOOPHOLE[TableDefs.TableBounds[sgtype].size];
  FOR sgi ← FIRST[SGIndex], sgi+SIZE[SGRecord]
  UNTIL sgi = sgLimit DO
    IF (sgb+sgi)↑ = seg THEN RETURN;
    ENDLOOP;
  sgi ← TableDefs.Allocate[sgtype, SIZE[SGRecord]];
  (sgb+sgi)↑ ← seg;
  RETURN
  END;

EnterImport: PROCEDURE [bcd: BcdBasePtr, olditi: IMPIndex, name: HTIndex]
  RETURNS [iti: IMPIndex] =
  BEGIN OPEN old: bcd.itb+olditi;
  iti ← TableDefs.Allocate[imptype, SIZE[IMPRecord]];
  (itb+iti)↑ ← old;
  (itb+iti).name ← MapName[bcd, old.name];
  IF name # HTNull THEN
    BEGIN
    (itb+iti).namedinstance ← TRUE;
    CreateInstanceName[name, [import[iti]]];
    END
  ELSE IF (itb+iti).namedinstance THEN
    BEGIN
    CopyInstanceName[bcd, [import[olditi]], [import[iti]]];
    END;
  RETURN
  END;

EnterExport: PROCEDURE [bcd: BcdBasePtr, oldeti: EXPIndex, name: HTIndex]
  RETURNS [eti: EXPIndex] =
  BEGIN OPEN new: etb+eti, old: bcd.etb+oldeti;
  i, size: CARDINAL;
  size ← old.size+SIZE[EXPRecord];
  eti ← TableDefs.Allocate[exptype, size];
  new ← old;
  FOR i IN [0..new.size) DO
    new.links[i] ← NullLink;
    ENDLOOP;
  new.name ← MapName[bcd, old.name];
  IF name # HTNull THEN
    BEGIN
    (etb+eti).namedinstance ← TRUE;
    CreateInstanceName[name, [export[eti]]];
    END
  ELSE IF (etb+eti).namedinstance THEN
    BEGIN
    CopyInstanceName[bcd, [export[oldeti]], [export[eti]]];
    END;
  RETURN
  END;

EnterConfig: PROCEDURE [bcd: BcdBasePtr, oldcti: CTIndex, name: HTIndex]
  RETURNS [cti: CTIndex] =
  BEGIN OPEN old: bcd.ctb+oldcti;
```

```
    cti ← TableDefs.Allocate[cttype, SIZE[CTRecord]];
    (ctb+cti)↑ ← old;
    (ctb+cti).name ← MapName[bcd, old.name];
    IF name # HTNull THEN
      BEGIN
      (ctb+cti).namedinstance ← TRUE;
      CreateInstanceName[name, [config[cti]]];
      END
    ELSE IF (ctb+cti).namedinstance THEN
      BEGIN
      CopyInstanceName[bcd, [config[oldcti]], [config[cti]]];
      END;
    RETURN
    END;

  CreateInstanceName: PROCEDURE [hti: HTIndex, item: Namee] =
    BEGIN
    nti: NTIndex ← TableDefs.Allocate[nttype, SIZE[NTRecord]];
    (ntb+nti).item ← item;
    (ntb+nti).name ← NameForHti[hti];
    RETURN
    END;

  CopyInstanceName: PRIVATE PROCEDURE [bcd: BcdBasePtr, old, new: Namee] =
    BEGIN
    nti: NTIndex ← TableDefs.Allocate[nttype, SIZE[NTRecord]];
    oldnti: NTIndex;
    (ntb+nti).item ← new;
    FOR oldnti ← FIRST[NTIndex], oldnti+SIZE[NTRecord] DO
      IF (bcd.ntb+oldnti).item = old THEN EXIT;
      ENDLOOP;
    (ntb+nti).name ← MapName[bcd, (bcd.ntb+oldnti).name];
    RETURN
    END;

  -- timing procedures

  timer: PRIVATE POINTER TO CARDINAL = LOOPHOLE[430B];

  TimeNow: PROCEDURE RETURNS [CARDINAL] =
    BEGIN
    RETURN[timer↑]
    END;

  TimeSince: PROCEDURE [start: CARDINAL] RETURNS [CARDINAL] =
    BEGIN OPEN InlineDefs;
    RETURN[(LongDiv[LongMult[timer↑-start,2*39],1000]+1)/2]
    END;

  -- Administrative Procedures

  InitUtilities: PROCEDURE =
    BEGIN
    TableDefs.AddNotify[Notifier];
    nextGfi ← nextDummyGfi ← 1;
    END;

  EraseUtilities: PROCEDURE =
    BEGIN
    TableDefs.DropNotify[Notifier];
    END;

  END.
```