```
-- BcdSEBuild.mesa; edited by Johnsson on April 12, 1978  5:29 PM

DIRECTORY
  BcdControlDefs: FROM "bcdcontroldefs",
  BcdDefs: FROM "bcddefs",
  BcdTabDefs: FROM "bcdtabdefs",
  BcdTreeDefs: FROM "bcdtreedefs",
  BcdUtilDefs: FROM "bcdutildefs",
  StringDefs: FROM "stringdefs",
  TableDefs: FROM "tabledefs";

DEFINITIONS FROM BcdDefs, BcdTabDefs, BcdTreeDefs;

BcdSEBuild: PROGRAM [data: BcdControlDefs.BinderData]
  IMPORTS BcdTabDefs, BcdTreeDefs, BcdUtilDefs, StringDefs, TableDefs
  EXPORTS BcdControlDefs =
  BEGIN

  BuildSEError: PUBLIC SIGNAL = CODE;

  currentCx, directoryCx: CXIndex;
  currentCti: CTIndex;
  tb, stb, ctb, cxb: TableDefs.TableBase;

  Notifier: TableDefs.TableNotifier =
    BEGIN
    tb ← base[treetype];
    stb ← base[sttype];
    ctb ← base[cttype];
    cxb ← base[cxtype];
    RETURN
    END;

  BuildSemanticEntries: PUBLIC PROCEDURE [root: TreeLink] =
    BEGIN
    TableDefs.AddNotify[Notifier];
    currentCx ← directoryCx ← BcdUtilDefs.NewContext[];
    currentCti ← CTNull;
    WITH root SELECT FROM
      subtree =>
        BEGIN OPEN tb+index;
        SELECT name FROM
          source =>
            BEGIN
            son1 ← updatelist[son1,dirItem];
            currentCx ← BcdUtilDefs.NewContext[];
            son2 ← updatelist[son2, PackId];
            currentCx ← BcdUtilDefs.NewContext[];
            son3 ← TreeWalk[son3];
            END;
          ENDCASE => SIGNAL BuildSEError;
        END;
      ENDCASE => SIGNAL BuildSEError;
    TableDefs.DropNotify[Notifier];
    RETURN
    END;

  TreeWalk: TreeMap =
    BEGIN
    saveIndex: CARDINAL = data.textIndex;
    WITH t SELECT FROM
      hash => t ← twItem[t];
      symbol => t ← twItem[t];
      subtree =>
        BEGIN
        data.textIndex ← (tb+index).sourceindex;
        SELECT (tb+index).name FROM
          list => [] ← updatelist[t, TreeWalk];
          item => [] ← twItem[t];
          config => twConfig[index];
          assign => twAssign[index];
          plus, then => [] ← twExpression[t];
          module => twModule[index];
          ENDCASE => SIGNAL BuildSEError;
        END;
      ENDCASE => SIGNAL BuildSEError;
```

```
    data.textIndex ← saveIndex;
    RETURN[t]
    END;

  PackId: TreeMap =
    BEGIN
    WITH t SELECT FROM
      hash => RETURN[SemanticEntry[t]];
      subtree => RETURN[updatelist[t, PackId]];
      ENDCASE => SIGNAL BuildSEError;
    END;

  ProcessItem: PROCEDURE [t: TreeLink]
    RETURNS [tl: TreeLink, st1, st2: STIndex] =
    BEGIN
    stl: symbol TreeLink;
    st2 ← STNull;
    WITH tt: t SELECT FROM
      symbol => BEGIN tl ← tt; st1 ← tt.index END;
      hash => BEGIN tl ← stl ← SemanticEntry[t]; st1 ← stl.index END;
      subtree =>
        BEGIN OPEN tb+tt.index;
        tl ← t;
        son1 ← stl ← SemanticEntry[son1];
        st1 ← stl.index;
        IF son2 # empty THEN
          BEGIN
          (stb+st1).filename ← FALSE;
          son2 ← stl ← SemanticEntry[son2];
          st2 ← stl.index;
          (stb+st1).body ← external[pointer: instance[st2], map: [unknown[]]];
          END;
        END;
      ENDCASE => SIGNAL BuildSEError;
    RETURN
    END;

  SetFilename: PROCEDURE [sti: STIndex] =
    BEGIN
    OPEN stb+sti;
    IF filename OR type # unknown THEN RETURN;
    filename ← TRUE;
    body ← external[pointer: file[FTNull], map: [unknown[]]];
    RETURN
    END;

  twItem: TreeMap =
    BEGIN
    st1, st2: STIndex;
    [v,st1,st2] ← ProcessItem[t];
    IF st2 = STNull THEN SetFilename[st1]
    ELSE SetFilename[st2];
    RETURN
    END;

  dirItem: TreeMap =
    BEGIN
    sti: STIndex;
    stl: symbol TreeLink;
    filename: STRING ← [40];
    fti: FTIndex;
    name: StringDefs.SubStringDescriptor;
    WITH t SELECT FROM
      subtree =>
        BEGIN
        stl ← SemanticEntry[(tb+index).son1];
        sti ← stl.index;
        WITH s2: (tb+index).son2 SELECT FROM
          hash => BcdTabDefs.SubStringForHash[@name,s2.index];
          ENDCASE;
        END;
      ENDCASE;
    StringDefs.AppendSubString[filename, @name];
    fti ← BcdUtilDefs.EnterFile[filename];
    (stb+sti).body ← external[map: [unknown[]], pointer: file[fti]];
    RETURN
```

```
    END;

  impItem: TreeMap =
    BEGIN
    st1, st2: STIndex;
    [v,st1,st2] ← ProcessItem[t];
    (stb+st1).imported ← TRUE;
    RETURN
    END;

  expItem: TreeMap =
    BEGIN
    st1, st2: STIndex;
    [v,st1,st2] ← ProcessItem[t];
    (stb+st1).exported ← TRUE;
    RETURN
    END;

  twConfig: PROCEDURE [t: TreeIndex] =
    BEGIN OPEN tb+t;
    saveCx: CXIndex = currentCx;
    saveName: NameRecord = data.currentname;
    EnterConfig[t];  -- name
    IF son1 # empty THEN son1 ← updatelist[son1, impItem]; -- IMPORTS
    IF son2 # empty THEN son2 ← updatelist[son2, expItem]; -- EXPORTS
    IF son3 # empty THEN  -- CONTROL
      WITH son3 SELECT FROM
        hash => son3 ← SemanticEntry[son3];
        symbol => NULL;
        ENDCASE => SIGNAL BuildSEError;
    son5 ← TreeWalk[son5];  -- body
    currentCx ← saveCx;
    data.currentname ← saveName;
    RETURN
    END;

  assignItem: TreeMap =
    BEGIN
    st1, st2: STIndex;
    [v,st1,st2] ← ProcessItem[t];
    (stb+st1).assigned ← TRUE;
    IF (stb+st1).filename THEN
      BEGIN OPEN stb+st1;
      filename ← FALSE;
      body ← external[pointer: instance[st2], map: [unknown[]]];
      END;
    IF st2 # STNull THEN
      BEGIN OPEN stb+st2;
      assigned ← TRUE;
      filename ← FALSE;
      body ← external[pointer: instance[STNull], map: [unknown[]]];
      END;
    RETURN
    END;

  twAssign: PROCEDURE [t: TreeIndex] =
    BEGIN OPEN tb+t;
    son1 ← updatelist[son1, assignItem];
    son2 ← twExpression[son2];
    END;

  twExpression: TreeMap =
    BEGIN
    WITH t SELECT FROM
      symbol => t ← ProcessItem[t].tl;
      hash => t ← ProcessItem[t].tl;
      subtree =>
        SELECT (tb+index).name FROM
          item => t ← ProcessItem[t].tl;
          module => twModule[index];
          plus, then =>
            BEGIN OPEN tb+index;
            son1 ← twExpression[son1];
            son2 ← twExpression[son2];
            END;
          ENDCASE => SIGNAL BuildSEError;
```

```
    ENDCASE => SIGNAL BuildSEError;
  RETURN[t]
  END;

modItem: TreeMap =
  BEGIN
  WITH t SELECT FROM
    symbol => NULL;
    hash => RETURN[SemanticEntry[t]];
    ENDCASE => SIGNAL BuildSEError;
  END;

twModule: PROCEDURE [t: TreeIndex] =
  BEGIN OPEN tb+t;
  son1 ← twItem[son1];
  son2 ← updatelist[son2, modItem];
  RETURN
  END;

SemanticEntry: PROCEDURE [tl: TreeLink] RETURNS [symbol TreeLink] =
  BEGIN
  sti, dirsti: STIndex;
  last: STIndex ← STNull;
  WITH t:tl SELECT FROM
    symbol => RETURN[t];
    hash =>
      BEGIN
      FOR sti ← (cxb+currentCx).link, (stb+sti).link UNTIL sti = STNull DO
        IF (stb+sti).hti = t.index THEN
          RETURN[TreeLink[symbol[sti]]];
        last ← sti;
        ENDLOOP;
      FOR dirsti ← (cxb+directoryCx).link, (stb+dirsti).link UNTIL dirsti = STNull DO
        IF (stb+dirsti).hti = t.index THEN
          EXIT;
        ENDLOOP;
      sti ← BcdUtilDefs.NewSemanticEntry[t.index];
      IF last = STNull THEN (cxb+currentCx).link ← sti
      ELSE (stb+last).link ← sti;
      IF dirsti # STNull THEN
        BEGIN
        (stb+sti)↑ ← (stb+dirsti)↑;
        (stb+sti).link ← STNull;
        END;
      RETURN[TreeLink[symbol[sti]]];
      END;
    ENDCASE => SIGNAL BuildSEError
  END;

EnterConfig: PROCEDURE [t: TreeIndex] =
  BEGIN
  stl: symbol TreeLink;
  sti: STIndex;
  stl ← SemanticEntry[(tb+t).son4];
  (tb+t).son4 ← stl;
  (stb+(sti ← stl.index)).filename ← FALSE;
  currentCx ← BcdUtilDefs.NewContext[];
  data.currentname ← BcdUtilDefs.NameForSti[sti];
  WITH stb+sti SELECT FROM
    unknown => (stb+sti).body ← local[info: t, context: currentCx];
    ENDCASE => SIGNAL BuildSEError;
  RETURN
  END;

END...
```