```
-- Includes.Mesa  Edited by Sandman on August 3, 1978  11:54 AM

DIRECTORY
  AltoDefs: FROM "altodefs",
  AltoFileDefs: FROM "altofiledefs",
  BcdDefs: FROM "bcddefs",
  ControlDefs: FROM "controldefs",
  DirectoryDefs: FROM "directorydefs",
  ImageDefs: FROM "imagedefs",
  IODefs: FROM "iodefs",
  MiscDefs: FROM "miscdefs",
  Mopcodes: FROM "mopcodes",
  OutputDefs: FROM "outputdefs",
  SymDefs: FROM "SymDefs",
  SegmentDefs: FROM "segmentdefs",
  StringDefs: FROM "stringdefs",
  StreamDefs: FROM "streamdefs",
  SymbolTableDefs: FROM "SymbolTableDefs",
  SystemDefs: FROM "SystemDefs",
  TimeDefs: FROM "timedefs";

DEFINITIONS FROM AltoDefs, AltoFileDefs, SegmentDefs, OutputDefs;

Includes: PROGRAM
  IMPORTS DirectoryDefs, IODefs, MiscDefs, OutputDefs, SegmentDefs,
    StringDefs, SymbolTableDefs, SystemDefs, TimeDefs, StreamDefs = PUBLIC
  BEGIN

  NullTime: TimeDefs.PackedTime = [0,0];
  NullStamp: BcdDefs.VersionStamp = [FALSE, 0, 0, [0,0]];
  SP: CHARACTER = ' ;
  FF: CHARACTER = 14C;

  FileEntry: TYPE = RECORD [
    link: FEPtr,
    name: STRING,
    includes: POINTER TO IncludeItem,
    includedBy: POINTER TO IncludeItem,
    stamp: BcdDefs.VersionStamp,
    mark: BOOLEAN,
    busy: BOOLEAN,
    bad: BOOLEAN];
  FEPtr: TYPE = POINTER TO FileEntry;

  fileList: POINTER TO FileEntry ← NIL;

  IncludeItem: TYPE = RECORD [
    link: POINTER TO IncludeItem,
    includedFile: FEPtr,
    stamp: BcdDefs.VersionStamp];

  TimeLess: PROCEDURE [a,b: TimeDefs.PackedTime] RETURNS [BOOLEAN] =
    BEGIN
    RETURN[IF a.highbits = b.highbits THEN a.lowbits < b.lowbits
      ELSE a.highbits < b.highbits]
    END;

  CompareString: PROCEDURE [a,b: STRING] RETURNS [{less,equal,greater}] =
    BEGIN
    CharAnd: PROCEDURE [CHARACTER,WORD] RETURNS [CHARACTER] =
      MACHINE CODE BEGIN Mopcodes.zAND END;
    l: CARDINAL = MIN[a.length,b.length];
    i: CARDINAL;
    ca, cb: CHARACTER;
    FOR i IN [0..l) DO
      ca ← a[i];
      cb ← b[i];
      IF ca IN ['a..'z] THEN ca ← CharAnd[ca,137B];
      IF cb IN ['a..'z] THEN cb ← CharAnd[cb,137B];
      IF ca < cb THEN RETURN[less];
      IF ca > cb THEN RETURN[greater];
      ENDLOOP;
    RETURN[SELECT a.length FROM
      < b.length => less,
      b.length => equal,
      ENDCASE => greater]
```

```
    END;

  CopyString: PROCEDURE [s: STRING] RETURNS [copy: STRING] =
    BEGIN
    copy ← SystemDefs.AllocateHeapString[s.length];
    StringDefs.AppendString[copy,s];
    RETURN
    END;

  CheckNameInList: PROCEDURE [name: STRING] =
    BEGIN
    p: FEPtr;
    FOR p ← fileList, p.link UNTIL p = NIL DO
      SELECT CompareString[p.name,name] FROM
        equal => BEGIN p.name ← name; RETURN END;
        ENDCASE;
      ENDLOOP;
    END;

  GetFile: PROCEDURE [name: STRING] RETURNS [p: FEPtr] =
    BEGIN
    last: FEPtr ← LOOPHOLE[@fileList]; -- assumes link field is word zero
    FOR p ← fileList, p.link UNTIL p = NIL DO
      SELECT CompareString[p.name,name] FROM
        less => last ← p;
        equal => RETURN;
        ENDCASE => EXIT;
      ENDLOOP;
    p ← SystemDefs.AllocateHeapNode[SIZE[FileEntry]];
    p.link ← last.link;
    last.link ← p;
    p.name ← name;
    p.stamp ← NullStamp;
    p.mark ← p.busy ← p.bad ← FALSE;
    p.includes ← p.includedBy ← NIL;
    RETURN
    END;

  NewIncludeItem: PROCEDURE [link: POINTER TO IncludeItem, fe: FEPtr, stamp: BcdDefs.VersionStamp]
    RETURNS [POINTER TO IncludeItem] =
    BEGIN
    p: POINTER TO IncludeItem;
    last: POINTER TO IncludeItem ← LOOPHOLE[@link]; -- assumes link field is word zero
    FOR p ← link, p.link UNTIL p = NIL DO
      SELECT CompareString[p.includedFile.name,fe.name] FROM
        less => last ← p;
        equal => RETURN[link];
        ENDCASE => EXIT;
      ENDLOOP;
    p ← SystemDefs.AllocateHeapNode[SIZE[IncludeItem]];
    p.link ← last.link;
    last.link ← p;
    p.includedFile ← fe;
    p.stamp ← stamp;
    RETURN[link]
    END;


  InvalidObjectFile: ERROR [file:FileHandle] = CODE;

  LoadSymbols: PROCEDURE [file:FileHandle] RETURNS [symseg:FileSegmentHandle] =
    BEGIN
    bcd: POINTER TO BcdDefs.BCD;
    pages: AltoDefs.PageCount;
    mtb: CARDINAL;
    mti: BcdDefs.MTIndex = FIRST[BcdDefs.MTIndex];
    bcdseg: FileSegmentHandle ← NewFileSegment[file,1,1,Read];
    Cleanup: PROCEDURE =
      BEGIN
      Unlock[bcdseg];
      DeleteFileSegment[bcdseg];
      RETURN
      END;
    SwapIn[bcdseg];
    bcd ← FileSegmentAddress[bcdseg];
    IF (pages ← bcd.nPages) # 1 THEN
```

```
    BEGIN
    Unlock[bcdseg];
    MoveFileSegment[bcdseg, 1, pages];
    SwapIn[bcdseg];
    bcd ← FileSegmentAddress[bcdseg];
    END;
  IF bcd.versionident # BcdDefs.VersionID OR bcd.nModules # 1 THEN
    ERROR InvalidObjectFile[file ! UNWIND => Cleanup[]];
  mtb ← LOOPHOLE[bcd,CARDINAL]+bcd.mtOffset;
  symseg ← FindSegment[bcdseg, (mtb+mti).sseg, FALSE ! UNWIND => Cleanup[]];
  IF symseg = NIL THEN ERROR InvalidObjectFile[file ! UNWIND => Cleanup[]];
  Cleanup[];
  RETURN
  END;

FindSegment: PROCEDURE [seg: FileSegmentHandle, sgi: BcdDefs.SGIndex, long: BOOLEAN]
  RETURNS [FileSegmentHandle] =
  BEGIN
  ss: StringDefs.SubStringDescriptor;
  file: SegmentDefs.FileHandle;
  name: STRING ← [40];
  bcd: POINTER TO BcdDefs.BCD ← FileSegmentAddress[seg];
  sgh: BcdDefs.SGHandle ← LOOPHOLE[bcd+bcd.sgOffset, CARDINAL]+sgi;
  ssb: BcdDefs.NameString ← LOOPHOLE[bcd+bcd.ssOffset];
  IF sgh.file = BcdDefs.FTNull THEN RETURN[NIL]
  ELSE IF sgh.file = BcdDefs.FTSelf THEN file ← seg.file
  ELSE
    BEGIN
    f: BcdDefs.FTHandle =
      LOOPHOLE[bcd+bcd.ftOffset, CARDINAL]+sgh.file;
    ss ← [@ssb.string, f.name, ssb.size[f.name]];
    StringDefs.AppendSubString[name, @ss];
    CheckForExtension[name, ".bcd"];
    file ← NewFile[name, DefaultAccess, DefaultVersion];
    END;
  RETURN[NewFileSegment[file, sgh.base,
    sgh.pages + (IF long THEN sgh.extraPages ELSE 0), Read]];
  END;

CheckForExtension: PROCEDURE [name, ext: STRING] =
  BEGIN
  i: CARDINAL;
  FOR i IN [0..name.length) DO
    IF name[i] = '. THEN RETURN;
    ENDLOOP;
  StringDefs.AppendString[name, ext];
  RETURN
  END;

ProcessIncludes: PROCEDURE [includer: FEPtr, symseg:FileSegmentHandle] =
  BEGIN OPEN SymbolTableDefs;
  symbols: SymbolTableBase = AcquireSymbolTable[TableForSegment[symseg]];
  mdLimit: SymDefs.MDIndex = LOOPHOLE[symbols.stHandle.mdBlock.size];
  mdi: SymDefs.MDIndex;
  includee: FEPtr;
  ss: StringDefs.SubStringDescriptor;
  sname: STRING ← [40];
  tname: STRING;
  i: CARDINAL;
  IODefs.WriteLine[includer.name];
  includer.stamp ← symbols.stHandle.version;
  FOR mdi ← FIRST[SymDefs.MDIndex] + SIZE[SymDefs.MDRecord], mdi + SIZE[SymDefs.MDRecord]
  UNTIL mdi = mdLimit DO
    symbols.SubStringForHash[@ss,(symbols.mdb+mdi).mdhti];
    sname.length ← 0;
    FOR i IN [0..ss.length) DO
      IF ss.base[ss.offset+i] = '. THEN EXIT;
      StringDefs.AppendChar[sname, ss.base[ss.offset+i]];
      ENDLOOP;
    tname ← CopyString[sname];
    includee ← GetFile[tname];
    includer.includes ← NewIncludeItem[
      includer.includes, includee, (symbols.mdb+mdi).mdStamp];
    includee.includedBy ←
      NewIncludeItem[includee.includedBy,includer,NullStamp];
    ENDLOOP;
```

```
    ReleaseSymbolTable[symbols];
    RETURN
    END;

  StripExtension: PROCEDURE [name,ext:STRING] =
    BEGIN OPEN StringDefs;
    i, j: INTEGER;
    ext.length ← 0;
    i ← (j ← name.length) - 1;
    IF name[i]='. THEN i ← (j ← i) - 1;
    UNTIL name[i] = '. DO
      IF name[i] = '! THEN j ← i;
      IF (i ← i-1) < 0 THEN RETURN;
      ENDLOOP;
    i ← i+1;
    UNTIL i=j DO
      AppendChar[ext,name[i]];
      i ← i+1;
      ENDLOOP;
    RETURN
    END;


  ext: STRING ← [FilenameChars+1];

  IsBCD: PROCEDURE [fp:POINTER TO FP, name:STRING]
    RETURNS [BOOLEAN] =
    BEGIN OPEN StringDefs;
    okay: BOOLEAN ← TRUE;
    tname: STRING;
    IF names = NIL THEN
      BEGIN
      StripExtension[name,ext];
      okay ← EquivalentString[ext,"bcd"];
      IF okay THEN tname ← CopyString[name];
      END
    ELSE [okay, tname] ← FindNameInList[name];
    IF okay THEN AddFile[InsertFile[fp,Read], tname];
    RETURN[FALSE]
    END;

  AddFile: PROCEDURE [file: FileHandle, name:STRING] =
    BEGIN OPEN StringDefs;
    syms: FileSegmentHandle;
    fe: FEPtr;
    i: CARDINAL;
      BEGIN
      syms ← LoadSymbols[file
        ! InvalidObjectFile => GOTO invalidfile; ANY => GOTO badfile];
      FOR i IN [0..name.length) DO
        IF name[i] = '. THEN
          BEGIN name.length ← i; EXIT END;
        ENDLOOP;
      fe ← GetFile[name];
      CheckNameInList[name];
      ProcessIncludes[fe, syms];
      DeleteFileSegment[syms];
      EXITS
        invalidfile => NULL;
        badfile =>
          BEGIN OPEN IODefs;
          WriteString["File "];
          WriteString[name];
          WriteLine[" is a bad file!!"];
          END;
      END;
    END;

  NameList: TYPE = POINTER TO NameItem;

  NameItem: TYPE = RECORD [
    link: NameList,
    name: STRING];

  names: NameList ← NIL;
```

```
  AddName: PROCEDURE [name: STRING] =
    BEGIN
    nl: NameList ← SystemDefs.AllocateHeapNode[SIZE[NameItem]];
    nl↑ ← NameItem[link: names, name: CopyAndCheck[name]];
    names ← nl;
    RETURN
    END;

  CopyAndCheck: PROCEDURE [name: STRING] RETURNS [s: STRING] =
    BEGIN
    i: CARDINAL;
    state: {none, ext, dot} ← none;
    FOR i IN [0..name.length) DO
      IF name[i] = '. THEN BEGIN state ← ext; EXIT END;
      ENDLOOP;
    IF name[name.length-1] = '. THEN state ← dot;
    s ← SystemDefs.AllocateHeapString[name.length +
      CARDINAL[(SELECT state FROM dot => 0, ext => 1, ENDCASE => 5)]];
    StringDefs.AppendString[s, name];
    SELECT state FROM
      ext => StringDefs.AppendChar[s, '.];
      none => StringDefs.AppendString[s, ".bcd."L];
      ENDCASE;
    RETURN
    END;

  FindNameInList: PROCEDURE [name: STRING] RETURNS [BOOLEAN, STRING] =
    BEGIN
    nl: NameList;
    FOR nl ← names, nl.link UNTIL nl = NIL DO
      IF StringDefs.EquivalentString[nl.name, name] THEN
        RETURN[TRUE, nl.name];
      ENDLOOP;
    RETURN[FALSE, NIL];
    END;

  PutStamp: PROCEDURE [s: BcdDefs.VersionStamp] =
    BEGIN
    PutTime[s.time];
    PutString[" #"];
    PutNumber[s.net,[8,FALSE,FALSE,1]];
    PutString[" #"];
    PutNumber[s.host,[8,FALSE,FALSE,1]];
    IF s.zapped THEN PutString[" zapped!!!"];
    RETURN
    END;

  printDate: BOOLEAN;

  FileName: PROCEDURE [fe: FEPtr] =
    BEGIN
    PutString[fe.name];
    IF ~printDate OR fe.stamp.time = NullTime THEN RETURN;
    PutString[" ("];
    PutStamp[fe.stamp];
    PutChar[')];
    RETURN
    END;

  OutputStats: PROCEDURE =
    BEGIN
    badfile: BOOLEAN;
    fe: FEPtr;
    i: POINTER TO IncludeItem;
    printDate ← TRUE;
    FOR fe ← fileList, fe.link UNTIL fe = NIL DO
      BEGIN
      IF fe.stamp.time = NullTime THEN GO TO loop
      ELSE FileName[fe];
      PutString[" includes"];
      IF fe.includes = NIL THEN PutString[" nothing"]
      ELSE FOR i ← fe.includes, i.link UNTIL i = NIL DO
        badfile ← i.stamp # i.includedFile.stamp AND i.includedFile.stamp # NullStamp AND i.stamp # Nul
**lStamp;
        PutCR[];
        PutChar[SP];
```

```
        PutChar[IF badfile THEN '* ELSE SP];
        FileName[i.includedFile];
        IF badfile THEN
          BEGIN
          fe.bad ← TRUE;
          PutString[" included version was "];
          PutStamp[i.stamp];
          END
        ELSE IF i.stamp = NullStamp THEN
          PutString[" ** never referenced"];
        ENDLOOP;
      PutCR[];PutCR[];
      EXITS loop => NULL;
      END;
      ENDLOOP;
    PutChar[FF];
    printDate ← FALSE;
    FOR fe ← fileList, fe.link UNTIL fe = NIL DO
      BEGIN
      FileName[fe];
      PutString[" is included by"];
      IF fe.includedBy = NIL THEN PutString[" nothing"]
      ELSE FOR i ← fe.includedBy, i.link UNTIL i = NIL DO
        PutCR[];
        PutString["  "];
        FileName[i.includedFile];
        ENDLOOP;
      PutCR[];PutCR[];
      END;
      ENDLOOP;
    END;

  Compile: PROCEDURE [fe: FEPtr] =
    BEGIN
    i: POINTER TO IncludeItem;
    IF fe.mark THEN RETURN;
    IF fe.busy THEN
      BEGIN
      PutString["
list of included modules from "];
      PutString[fe.name];
      PutString[" forms a circle
"];
      RETURN
      END;
    fe.busy ← TRUE;
    FOR i ← fe.includes, i.link UNTIL i = NIL DO
      Compile[i.includedFile];
      ENDLOOP;
    PutChar[SP];
    PutString[fe.name];
    fe.mark ← TRUE;
    fe.busy ← FALSE;
    END;

  OutputCompileOrder: PROCEDURE =
    BEGIN
    fe: FEPtr;
    PutString["Compilation Order:
"];
    FOR fe ← fileList, fe.link UNTIL fe = NIL DO
      Compile[fe];
      ENDLOOP;
    PutCR[]; PutCR[];
    END;

  Init: PROCEDURE =
    BEGIN OPEN SystemDefs;
    fe: FEPtr;
    i: POINTER TO IncludeItem;
    UNTIL fileList = NIL DO
      UNTIL fileList.includes = NIL DO
        i ← fileList.includes.link;
        FreeHeapNode[fileList.includes];
        fileList.includes ← i;
        ENDLOOP;
```

```
    UNTIL fileList.includedBy = NIL DO
      i ← fileList.includedBy.link;
      FreeHeapNode[fileList.includedBy];
      fileList.includedBy ← i;
      ENDLOOP;
    fe ← fileList.link;
    FreeHeapString[fileList.name];
    FreeHeapNode[fileList];
    fileList ← fe;
    ENDLOOP;
  RETURN
  END;

GetToken: PROCEDURE [name: STRING, in: StreamDefs.StreamHandle] =
  BEGIN
  c: CHARACTER;
  name.length ← 0;
  UNTIL in.endof[in] DO
    c ← in.get[in];
    SELECT c FROM
      IODefs.SP, IODefs.CR => IF name.length # 0 THEN RETURN;
      ENDCASE => StringDefs.AppendChar[name, c];
    ENDLOOP;
  RETURN
  END;

DoWork: PROCEDURE =
  BEGIN
  cfa: POINTER TO AltoFileDefs.CFA;
  in: StreamDefs.StreamHandle;
  root: STRING ← [40];
  name: STRING ← [40];
  cfa ← MiscDefs.CommandLineCFA[];
  in ← StreamDefs.CreateByteStream[
    SegmentDefs.InsertFile[@cfa.fp, StreamDefs.Read], StreamDefs.Read];
  StreamDefs.JumpToFA[in, @cfa.fa];
  GetToken[root, in];
  IF root.length = 0 THEN StringDefs.AppendString[root, "Includes"];
  Init[];
  IODefs.WriteChar[IODefs.CR];
  GetToken[name, in];
  UNTIL name.length = 0 DO
    AddName[name];
    GetToken[name, in];
    ENDLOOP;
  DirectoryDefs.EnumerateDirectory[IsBCD];
  OpenOutput[root,".list"];
  OutputCompileOrder[];
  OutputStats[];
  CloseOutput[];
  END;

MyVersion: PROCEDURE RETURNS [version: BcdDefs.VersionStamp] =
  BEGIN OPEN SegmentDefs;
  self: ControlDefs.GlobalFrameHandle = REGISTER[ControlDefs.Greg];
  seg: FileSegmentHandle ← NewFileSegment[self.codesegment.file,1,1,Read];
  bcd: POINTER TO BcdDefs.BCD;
  SwapIn[seg];
  bcd ← FileSegmentAddress[seg];
  version ← bcd.version;
  Unlock[seg];
  DeleteFileSegment[seg];
  RETURN
  END;

LoadIncludes: PROCEDURE =
  BEGIN OPEN TimeDefs;
  herald: STRING ← [50];
  StringDefs.AppendString[herald, "Alto/Mesa Include Checker "L];
  AppendDayTime[herald, UnpackDT[MyVersion[].time]];
  herald.length ← herald.length - 3;
  IODefs.WriteLine[herald];
  herald.length ← 0;
  AppendDayTime[herald, UnpackDT[DefaultTime]];
  herald.length ← herald.length - 3;
  IODefs.WriteLine[herald];
```

```
    END;

  -- MAIN BODY CODE

  LoadIncludes[];
  DoWork[];
  ImageDefs.StopMesa[];

  END.
```