```
-- Includes.Mesa  Edited by Sandman on October 14, 1977  1:57 PM

DIRECTORY
  AltoDefs: FROM "altodefs",
  AltoFileDefs: FROM "altofiledefs",
  BcdDefs: FROM "bcddefs",
  CommanderDefs: FROM "commanderdefs",
  DirectoryDefs: FROM "directorydefs",
  ImageDefs: FROM "imagedefs",
  IODefs: FROM "iodefs",
  Mopcodes: FROM "mopcodes",
  OutputDefs: FROM "outputdefs",
  SymDefs: FROM "SymDefs",
  SegmentDefs: FROM "segmentdefs",
  StringDefs: FROM "stringdefs",
  SymbolTableDefs: FROM "SymbolTableDefs",
  SystemDefs: FROM "SystemDefs",
  TimeDefs: FROM "timedefs";

DEFINITIONS FROM AltoDefs, AltoFileDefs, SegmentDefs, OutputDefs;

Includes: PROGRAM
  IMPORTS CommanderDefs, DirectoryDefs, IODefs, OutputDefs, SegmentDefs, StringDefs,
    SymbolTableDefs, SystemDefs, ImageDefs, TimeDefs = 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: MACHINE CODE [CHARACTER,WORD] RETURNS [CHARACTER] =
      INLINE [Mopcodes.zAND];
    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;

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

```
    DeleteFileSegment[bcdseg];
    RETURN
    END;

  FindSegment: PROCEDURE [seg: FileSegmentHandle, segdesc: BcdDefs.SegDesc, long: BOOLEAN]
    RETURNS [FileSegmentHandle] =
    BEGIN
    ss: StringDefs.SubStringDescriptor;
    file: SegmentDefs.FileHandle;
    name: STRING;
    bcd: POINTER TO BcdDefs.BCD ← FileSegmentAddress[seg];
    IF segdesc.file = BcdDefs.FTNull THEN RETURN[NIL]
    ELSE IF segdesc.file = BcdDefs.FTSelf THEN file ← seg.file
    ELSE
      BEGIN OPEN f: LOOPHOLE[bcd+bcd.ftOffset, CARDINAL]+segdesc.file;
      name ← SystemDefs.AllocateHeapString[f.name.length+4];
      ss ← [LOOPHOLE[bcd+bcd.ssOffset, STRING], f.name.offset, f.name.length];
      StringDefs.AppendSubString[name, @ss];
      CheckForExtension[name, ".bcd"];
      file ← NewFile[name, DefaultAccess, DefaultVersion];
      SystemDefs.FreeHeapString[name];
      END;
    RETURN[NewFileSegment[file, segdesc.base,
      segdesc.pages + (IF long THEN segdesc.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.mdSize];
    mdi: SymDefs.MDIndex;
    includee: FEPtr;
    ss: StringDefs.SubStringDescriptor;
    tname: STRING ← [40];
    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];
      tname.length ← 0;
      FOR i IN [0..ss.length) DO
        IF ss.base[ss.offset+i] = '. THEN EXIT;
        StringDefs.AppendChar[tname, ss.base[ss.offset+i]];
        ENDLOOP;
      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;
    file: FileHandle;
    syms: FileSegmentHandle;
    fe: FEPtr;
    i: CARDINAL;
    tname: STRING ← [40];
    StripExtension[name,ext];
    IF EquivalentString[ext,"bcd"] THEN
      BEGIN
      file ← InsertFile[fp,Read];
      syms ← LoadSymbols[file
        ! InvalidObjectFile => GOTO invalidfile; ANY => GOTO badfile];
      FOR i IN [0..name.length) DO
        IF name[i] = '. THEN EXIT;
        StringDefs.AppendChar[tname, name[i]];
        ENDLOOP;
      fe ← GetFile[tname];
      ProcessIncludes[fe, syms];
      DeleteFileSegment[syms];
      EXITS
        invalidfile => NULL;
        badfile =>
          BEGIN OPEN IODefs;
          WriteString["File "];
          WriteString[name];
          WriteLine[" is a bad file!!"];
          END;
      END;
    RETURN[FALSE]
    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;

  xIncludes: PROCEDURE [root: STRING] =
    BEGIN
    Init[];
    IODefs.WriteChar[IODefs.CR];
    DirectoryDefs.EnumerateDirectory[IsBCD];
    OpenOutput[root,".list"];
    OutputCompileOrder[];
    OutputStats[];
    CloseOutput[];
    END;

  command: CommanderDefs.CommandBlockHandle;

  LoadIncludes: PROCEDURE =
    BEGIN OPEN TimeDefs;
    command: CommanderDefs.CommandBlockHandle;
    herald: STRING ← SystemDefs.AllocateHeapString[50];
    StringDefs.AppendString[herald, "Alto/Mesa Include Checker "];
    START CommanderDefs.Commander[herald];
    command ← CommanderDefs.AddCommand["Includes",LOOPHOLE[xIncludes],1];
    command.params[0] ← [type: string, prompt: "Filename"];
    START IODefs.StreamIO[NIL, NIL];
    ImageDefs.MakeImage["IncludeChecker.image.", FALSE];
    AppendDayTime[herald, UnpackDT[ImageDefs.ImageVersion[].time]];
    herald.length ← herald.length - 3;
    RESTART CommanderDefs.Commander;
    END;

  -- MAIN BODY CODE

  LoadIncludes[];

  END.
```