```
-- MakeModule.Mesa  Edited by Sandman on April 28, 1978  8:00 AM

DIRECTORY
  AltoDefs: FROM "altodefs",
  AltoFileDefs: FROM "altofiledefs",
  BcdDefs: FROM "bcddefs",
  ControlDefs: FROM "ControlDefs",
  MiscDefs: FROM "miscdefs",
  ImageDefs: FROM "imagedefs",
  IODefs: FROM "iodefs",
  OsStaticDefs: FROM "osstaticdefs",
  SegmentDefs: FROM "segmentdefs",
  StreamDefs: FROM "streamdefs",
  StringDefs: FROM "stringdefs",
  SystemDefs: FROM "systemdefs",
  TimeDefs: FROM "timedefs";

DEFINITIONS FROM BcdDefs, SegmentDefs;

MakeModule: PROGRAM
  IMPORTS IODefs, MiscDefs, SegmentDefs, StreamDefs, StringDefs, SystemDefs =

  PUBLIC BEGIN

  bcdheader: BcdDefs.BCD;
  module: BcdDefs.MTRecord;
  export: POINTER TO BcdDefs.EXPRecord;
  defsfile: BcdDefs.FTRecord;
  codeseg, symbolseg: BcdDefs.SGRecord;
  ssbstring: STRING ← [60];
  ssb: BcdDefs.NameString ← LOOPHOLE[ssbstring];
  segmentname: STRING ← [40];
  modulename: STRING ← [40];
  defsfilename: STRING ← [40];
  outfile: STRING ← [40];
  in, out: StreamDefs.StreamHandle;
  file: SegmentDefs.FileHandle;
  interfacesize, modulenumber: CARDINAL;
  dname: BcdDefs.NameRecord;
  brFile: BOOLEAN;
  count: CARDINAL;

  WriteBcd: PROCEDURE [out: StreamDefs.StreamHandle] =
    BEGIN OPEN StreamDefs;
    [] ← WriteBlock[out, @bcdheader, SIZE[BcdDefs.BCD]];
    [] ← WriteBlock[out, ssb, StringDefs.WordsForString[ssb.string.length]];
    [] ← WriteBlock[out, @module, SIZE[BcdDefs.MTRecord]];
    [] ← WriteBlock[out, @codeseg, SIZE[BcdDefs.SGRecord]];
    [] ← WriteBlock[out, @symbolseg, SIZE[BcdDefs.SGRecord]];
    [] ← WriteBlock[out, @defsfile, SIZE[BcdDefs.FTRecord]];
    [] ← WriteBlock[out, export, SIZE[BcdDefs.EXPRecord]+interfacesize];
    END;

  FillInHeader: PROCEDURE =
    BEGIN OPEN bcdheader;
    time: AltoFileDefs.TIME ← MiscDefs.DAYTIME[];
    net: CARDINAL ← MiscDefs.GetNetworkNumber[];
    MiscDefs.Zero[@bcdheader, SIZE[BcdDefs.BCD]];
    versionident ← BcdDefs.VersionID;
    version ← BcdDefs.VersionStamp[
      time: TimeDefs.PackedTime[lowbits: time.low, highbits: time.high],
      zapped: FALSE,
      net: net,
      host: OsStaticDefs.OsStatics.SerialNumber];
    creator ← BcdDefs.VersionStamp[
      time: TimeDefs.PackedTime[lowbits: 1, highbits: 2],
      zapped: FALSE,
      net: net,
      host: OsStaticDefs.OsStatics.SerialNumber];
    nPages ← 1;
    nModules ← nExports ← 1;
    definitions ← FALSE;
    firstdummy ← 2;
    nDummies ← 0;
    ssOffset ← ctOffset ← impOffset ← ntOffset ← SIZE[BcdDefs.BCD];
    ssLimit ← StringDefs.WordsForString[ssb.string.length];
```

```
    mtOffset ← ssOffset + LOOPHOLE[ssLimit, CARDINAL];
    mtLimit ← LOOPHOLE[SIZE[MTRecord]];
    sgOffset ← mtOffset + LOOPHOLE[mtLimit, CARDINAL];
    sgLimit ← LOOPHOLE[2*SIZE[SGRecord]];
    ftOffset ← sgOffset + LOOPHOLE[sgLimit, CARDINAL];
    ftLimit ← LOOPHOLE[SIZE[FTRecord]];
    expOffset ← ftOffset + LOOPHOLE[ftLimit, CARDINAL];
    expLimit ← LOOPHOLE[SIZE[EXPRecord]+interfacesize];
    source ← BcdDefs.NullName;
    -- all other fields are set to zero
    RETURN
    END;

  FillInModule: PROCEDURE [name: BcdDefs.NameRecord, segmentSize: CARDINAL] =
    BEGIN
    MiscDefs.Zero[@module, SIZE[MTRecord]];
    module ← MTRecord[name: name, namedinstance: FALSE, initial: FALSE,
      file: FTSelf, links: frame, config: CTNull, code: [sgi: FIRST[SGIndex],
      linkspace: FALSE, packed: FALSE, offset: 0, length: 2*segmentSize],
      sseg: FIRST[SGIndex]+SIZE[SGRecord], framesize: 4, fsi: 0, gfi: 1,
      ngfi: 1, frame: [length: 0, frag: ]];
    codeseg ← SGRecord[class: code, file: FTSelf, base: 2,
      pages: SystemDefs.PagesForWords[segmentSize], extraPages: 0];
    symbolseg ← SGRecord[class: symbols, file: FTNull, base: 0,
      pages: 0, extraPages: 0];
    RETURN
    END;

  FillInExport: PROCEDURE [name: BcdDefs.NameRecord] =
    BEGIN
    export ← SystemDefs.AllocateHeapNode[
      SIZE[BcdDefs.EXPRecord]+interfacesize];
    MiscDefs.Zero[export, SIZE[BcdDefs.EXPRecord]+interfacesize];
    export↑ ← EXPRecord[name: name, size: interfacesize, port: interface,
      namedinstance: FALSE, file: FIRST[FTIndex], links:];
    export.links[modulenumber] ←
      ControlDefs.ControlLink[procedure[gfi: 1, ep: 0, tag: frame]];
    -- all other fields are set to zero
    RETURN
    END;

  FillInDefsFile: PROCEDURE [name: BcdDefs.NameRecord] =
    BEGIN OPEN SegmentDefs;
    defsseg: FileSegmentHandle;
    defsbcd: POINTER TO BcdDefs.BCD;
    s: STRING ← [40];
    StringDefs.AppendString[s, defsfilename];
    StringDefs.AppendString[s, ".bcd."];
    defsseg ← NewFileSegment[NewFile[s, Read, OldFileOnly], 1, 1, Read];
    SwapIn[defsseg];
    defsbcd ← FileSegmentAddress[defsseg];
    defsfile.version ← defsbcd.version;
    Unlock[defsseg];
    DeleteFileSegment[defsseg];
    defsfile.name ← name;
    -- all other fields are set to zero
    RETURN
    END;

  InitializePackedString: PROCEDURE =
    BEGIN
    ssb.string.length ← 1;
    ssb.size[1] ← 0;
    END;

  AddName: PROCEDURE [n: STRING] RETURNS [name: BcdDefs.NameRecord] =
    BEGIN
    StringDefs.AppendChar[@ssb.string, LOOPHOLE[n.length]];
    name ← BcdDefs.NameRecord[ssb.string.length];
    StringDefs.AppendString[@ssb.string, n];
    RETURN
    END;

  CheckForBr: PROCEDURE [name: STRING] RETURNS [BOOLEAN] =
    BEGIN
    i: CARDINAL;
```

```
    FOR i IN [0..name.length) DO
      IF name[i] = '. THEN GOTO extension;
      REPEAT
        extension =>
          BEGIN
          IF name.length = i+3 AND
            (name[i+1] = 'b OR name[i+1] = 'B) AND
            (name[i+2] = 'r OR name[i+2] = 'R) THEN RETURN[TRUE];
          END;
      ENDLOOP;
    RETURN[FALSE];
    END;

  FindSegmentSize: PROCEDURE [br: BOOLEAN, seg: FileHandle]
    RETURNS [count: CARDINAL, in: StreamDefs.StreamHandle] =
    BEGIN OPEN StreamDefs;
    in ← CreateWordStream[file, Read];
    IF br THEN
      BEGIN
      SetIndex[in, [0, 7*AltoDefs.BytesPerWord]];
      count ← in.get[in];
      SetIndex[in, [0, count*AltoDefs.BytesPerWord]];
      count ← in.get[in];
      END
    ELSE
      BEGIN
      eof: StreamIndex ← FileLength[in];
      count ←
        (eof.page*AltoDefs.BytesPerPage + eof.byte)/AltoDefs.BytesPerWord;
      in.reset[in];
      END;
    RETURN
    END;

  WriteSegment: PROCEDURE [in, out: StreamDefs.StreamHandle, count: CARDINAL] =
    BEGIN OPEN StreamDefs;
    p: POINTER;
    p ← SystemDefs.AllocateSegment[count];
    count ← ReadBlock[in, p, count];
    IF WriteBlock[out, p, count] # count THEN ERROR;
    in.destroy[in];
    out.destroy[out];
    RETURN
    END;


  DO OPEN IODefs, SegmentDefs;
    ENABLE Rubout => BEGIN WriteLine["XXX"]; RETRY END;
    WriteString["Module Name: "];
    ReadID[modulename];
    IF modulename.length = 0 THEN EXIT;
    WriteChar[CR];
    WriteString["Segment File: "];
    ReadID[segmentname];
    WriteChar[CR];
    WriteString["Defs File: "];
    ReadID[defsfilename];
    WriteChar[CR];
    WriteString["Interface Size: "];
    interfacesize ← ReadDecimal[];
    WriteChar[CR];
    WriteString["Module Interface Number: "];
    modulenumber ← ReadDecimal[];
    WriteChar[CR];

    brFile ← CheckForBr[segmentname];
    file ← NewFile[segmentname, Read, OldFileOnly];
    outfile.length ← 0;
    StringDefs.AppendString[outfile, modulename];
    StringDefs.AppendString[outfile, ".bcd."];
    out ← StreamDefs.NewWordStream[outfile, Write+Append];
    InitializePackedString[];
    dname ← AddName[defsfilename];
    [count, in] ← FindSegmentSize[brFile, file];
    FillInModule[AddName[modulename], count];
    FillInExport[dname];
```

```
    FillInDefsFile[dname];
    FillInHeader[]; -- Do this after all strings entered
    WriteBcd[out];
    StreamDefs.SetIndex[out, [1, 0]];
    WriteSegment[in, out, count];
    ENDLOOP;

  ImageDefs.StopMesa[];

  END.
```