```
-- Version.Mesa  Edited by Johnsson on April 21, 1978  9:44 AM

DIRECTORY
  AltoDefs: FROM "altodefs",
  AltoFileDefs: FROM "altofiledefs",
  BcdDefs: FROM "bcddefs",
  DirectoryDefs: FROM "directorydefs",
  ImageDefs: FROM "imagedefs",
  InlineDefs: FROM "inlinedefs",
  IODefs: FROM "iodefs",
  SegmentDefs: FROM "segmentdefs",
  StreamDefs: FROM "streamdefs",
  StringDefs: FROM "StringDefs",
  SystemDefs: FROM "SystemDefs",
  TimeDefs: FROM "timedefs";

DEFINITIONS FROM IODefs, StreamDefs, StringDefs;

Version: PROGRAM
  IMPORTS DirectoryDefs, IODefs, SegmentDefs, StreamDefs, StringDefs, SystemDefs, TimeDefs
  SHARES ImageDefs =
  BEGIN

  FP: TYPE = AltoFileDefs.FP;
  BcdBase: TYPE = POINTER TO BcdDefs.BCD;
  UnpackedTime: TYPE = TimeDefs.UnpackedTime;

  -- convert characters to upper case
  CharMask: PROCEDURE[CHARACTER,CARDINAL] RETURNS [CARDINAL] =
    LOOPHOLE[InlineDefs.BITAND];
  upcase: CARDINAL = 137B;


  BinaryVersion: VersionProc =
    BEGIN OPEN SegmentDefs;
    bcd: BcdBase;
    seg: FileSegmentHandle ←
      NewFileSegment[InsertFile[@info.fp,Read],1,1,Read];
    SwapIn[seg];  bcd ← FileSegmentAddress[seg];
    SELECT bcd.versionident FROM
      BcdDefs.VersionID =>
        BEGIN
        PrintVersion[bcd.version];
        WriteChar[CR];
        IF type=bcd AND verbose THEN
          BEGIN
          IF bcd.nPages > 1 THEN
            BEGIN
            Unlock[seg];
            MoveFileSegment[seg,1,bcd.nPages];
            SwapIn[seg];
            bcd ← FileSegmentAddress[seg];
            END;
          PrintFileVersions[bcd];
          END;
        END;
      ImageDefs.VersionID =>
        BEGIN
        PrintVersion[bcd.version];
        WriteChar[CR];
        END;
      ENDCASE =>
        BEGIN
        WriteString[" unknown version ID "L];
        WriteDecimal[bcd.versionident];
        WriteChar[CR];
        END;
    Unlock[seg];
    DeleteFileSegment[seg];
    RETURN
    END;

  BufferObject: TYPE = RECORD [
    p: CARDINAL,
    s: STRING];
  Buffer: TYPE = POINTER TO BufferObject;
```

```
SourceDate: VersionProc =
  BEGIN OPEN AltoDefs;
  LI: TYPE = LONG INTEGER;
  date, tdate: TimeDefs.PackedTime;
  b: BufferObject;
  stream: StreamHandle;
  found: BOOLEAN ← FALSE;
  stream ← CreateByteStream[
    SegmentDefs.InsertFile[@info.fp,Read],Read];
  b.s ← SystemDefs.AllocatePages[1];
  b.s↑ ← [
    length:0, maxlength:(PageSize-SIZE[StringBody])*BytesPerWord, text:];
  b.s.length ← ReadBlock[
    stream,@b.s.text,b.s.maxlength/BytesPerWord]*BytesPerWord;
  b.p ← 0;
  [found,date] ← GetDate[@b ! TimeDefs.InvalidTime => CONTINUE];
  IF found THEN
    BEGIN
    DO
      [found,tdate] ← GetDate[@b ! TimeDefs.InvalidTime => EXIT];
      IF ~found THEN EXIT;
      IF LOOPHOLE[tdate,LI] > LOOPHOLE[date,LI] THEN date ← tdate;
      ENDLOOP;
    WriteChar[' ]; PrintDate[TimeDefs.UnpackDT[date]];
    END
  ELSE WriteString[" ?"L];
  WriteChar[CR];
  SystemDefs.FreePages[b.s];
  stream.destroy[stream];
  RETURN
  END;

GetDate: PROCEDURE [b: Buffer] RETURNS [BOOLEAN,TimeDefs.PackedTime] =
  BEGIN
  token: STRING ← [20];
  found: BOOLEAN ← FALSE;
  index: CARDINAL[0..12);
  u: UnpackedTime ← [0,0,0,0,0,0,0,0,FALSE];
  NextItem[b,token];
  WHILE token.length # 0 DO
    [found,index] ← IsMonth[token];
    IF found THEN
      BEGIN
      NextItem[b,token];
      IF IsNumber[token] THEN
        BEGIN
        u.month ← index;
        u.day ← StringToDecimal[token];
        END
      ELSE LOOP;
      END
    ELSE IF IsNumber[token] THEN -- possible Day-Mo-Yr form
      BEGIN
      u.day ← StringToDecimal[token];
      NextItem[b,token];
      [found,index] ← IsMonth[token];
      IF found THEN u.month ← index ELSE LOOP;
      END;
    IF found THEN
      BEGIN
      NextItem[b,token];
      IF IsNumber[token] THEN
        BEGIN
        u.year ← StringToDecimal[token];
        IF u.year < 100 THEN u.year ← u.year+1900;
        END
      ELSE LOOP;
      NextItem[b,token];
      IF IsNumber[token] THEN
        BEGIN
        u.hour ← StringToDecimal[token];
        NextItem[b,token];
        IF IsNumber[token] THEN
          u.minute ← StringToDecimal[token];
        NextItem[b,token];
```

```
        IF EquivalentString[token,"PM"L] AND u.hour < 12 THEN
          u.hour ← u.hour+12;
        END;
      RETURN[TRUE,TimeDefs.PackDT[u,TRUE]]
      END;
    NextItem[b,token];
    ENDLOOP;
  RETURN[FALSE,TimeDefs.DefaultTime]
  END;

IsMonth: PROCEDURE [candidate: STRING] RETURNS [BOOLEAN,CARDINAL] =
  BEGIN
  Months: ARRAY [0..12) OF STRING = [
    "JANUARY"L,"FEBRUARY"L,"MARCH"L,"APRIL"L,"MAY"L,"JUNE"L,"JULY"L,
    "AUGUST"L,"SEPTEMBER"L,"OCTOBER"L,"NOVEMBER"L,"DECEMBER"L];
  i,j: CARDINAL;
  test: STRING;
  IF candidate.length >= 3 THEN
    FOR i IN [0..12) DO
      test ← Months[i];
      IF candidate.length <= test.length THEN
        FOR j IN [0..candidate.length) DO
          IF CharMask[candidate[j],upcase] # LOOPHOLE[test[j],CARDINAL] THEN EXIT;
          REPEAT FINISHED => RETURN[TRUE,i];
          ENDLOOP;
      ENDLOOP;
  RETURN[FALSE,0]
  END;

IsNumber: PROCEDURE [s: STRING] RETURNS [BOOLEAN] =
  BEGIN
  i: CARDINAL;
  FOR i IN [0..s.length) DO
    IF s[i] ~IN ['0..'9] THEN RETURN[FALSE];
    ENDLOOP;
  RETURN[TRUE]
  END;

NextItem: PROCEDURE [b: Buffer, token: STRING] =
  BEGIN
  c: CHARACTER;
  token.length ← 0;
  WHILE b.p < b.s.length DO
     c ← b.s[b.p];
     b.p ← b.p+1;
     SELECT c FROM
       IN ['a..'z], IN ['A..'Z], IN ['0..'9] =>
         IF token.length < token.maxlength THEN AppendChar[token,c];
       ENDCASE => IF token.length # 0 THEN EXIT;
     ENDLOOP;
  RETURN
  END;

PrintFileVersions: PROCEDURE [bcd: BcdBase] =
  BEGIN OPEN BcdDefs;
  line: STRING ← [40];
  filename: SubStringDescriptor;
  fti: FTIndex;
  ftb: CARDINAL = LOOPHOLE[bcd+bcd.ftOffset];
  stb: NameString = LOOPHOLE[bcd+bcd.ssOffset];
  FOR fti ← FIRST[FTIndex], fti+SIZE[FTRecord]
  UNTIL fti = bcd.ftLimit DO
    OPEN f: ftb + fti;
    filename ← SubStringDescriptor[
      @stb.string, f.name, stb.size[f.name]];
    line.length ← 0;  WriteChar[' ];
    PrintVersion[f.version]; WriteChar[' ];
    AppendSubString[line,@filename];
    WriteLine[line];
    ENDLOOP;
  RETURN
  END;

PrintVersion: PROCEDURE [stamp: BcdDefs.VersionStamp] =
  BEGIN
  WriteChar[IF stamp.zapped THEN '* ELSE ' ];
```

```
    PrintDate[TimeDefs.UnpackDT[stamp.time]];
    WriteChar[' ];
    WriteOctal[stamp.net];
    WriteChar['#];
    WriteOctal[stamp.host];
    WriteChar['#];
    END;

  PrintDate: PROCEDURE [dt: UnpackedTime] =
    BEGIN
    tmp: STRING ← [40];
    TimeDefs.AppendDayTime[tmp,dt];
    WriteString[tmp];
    END;


  -- command line stuff

  NextFile: PROCEDURE[token: STRING] RETURNS [BOOLEAN];

  ReadFromCmdFile: PROCEDURE[token: STRING] RETURNS [BOOLEAN] =
    BEGIN
    RETURN[ReadCmdStream[cmdstream,token]]
    END;

  ReadFromKeyboard: PROCEDURE[token: STRING] RETURNS [BOOLEAN] =
    BEGIN
    WriteString["File: "L];
    ReadID[token];  WriteChar[CR];
    RETURN[token.length#0];
    END;

  ReadCmdStream: PROCEDURE [
    stream: StreamHandle, token: STRING]
    RETURNS [BOOLEAN] =
    BEGIN
    c: CHARACTER;
    token.length ← 0;
    WHILE ~stream.endof[stream] DO
       c ← stream.get[stream];
       SELECT c FROM
         SP => IF token.length # 0 THEN EXIT;
         CR => EXIT;
         ENDCASE => AppendChar[token,c];
       ENDLOOP;
    RETURN[token.length#0];
    END;

  cmdstream: StreamHandle;

  SetCommandInput: PROCEDURE =
    BEGIN
    cmdstream ← NewByteStream["Com.Cm"L,Read];
    [] ← ReadCmdStream[cmdstream, name];
    StripSwitches[name];  -- mesa.image
    [] ← ReadCmdStream[cmdstream, name];
    IF interactive ← cmdstream.endof[cmdstream] THEN
      BEGIN
      cmdstream.destroy[cmdstream];
      NextFile ← ReadFromKeyboard;
      END
    ELSE NextFile ← ReadFromCmdFile;
    RETURN
    END;

  StripSwitches: PROCEDURE [token: STRING] =
    BEGIN
    i, j: CARDINAL;
    option: BOOLEAN ← TRUE;
    FOR i IN [0..token.length) DO
      IF token[i] = '/ THEN
        BEGIN
        FOR j IN (i..token.length) DO
          SELECT token[j] FROM
            '~, '- => option ← ~option;
            'p, 'P => BEGIN pause ← option; option ← TRUE END;
```

```
          'v, 'V => BEGIN verbose ← option; option ← TRUE END;
          ENDCASE;
        ENDLOOP;
      token.length ← i;
      END;
    ENDLOOP;
  RETURN
  END;

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

-- presearching the directory

HashVal: TYPE = [0..1000);

Hash: PROCEDURE [s: STRING] RETURNS [HashVal] =
  BEGIN
  RETURN[(CharMask[s[0],upcase]*s.length+CharMask[s[s.length/2],upcase]) MOD (LAST[HashVal]+1)];
  END;

StringItem: TYPE = RECORD [
  link: POINTER TO StringItem,
  hash: HashVal,
  s: STRING];

StringHead: POINTER TO StringItem ← NIL;

CopyString: PROCEDURE [s: STRING] RETURNS [STRING] =
  BEGIN
  p: POINTER TO StringItem;
  hash: HashVal = Hash[s];
  FOR p ← StringHead, p.link UNTIL p = NIL DO
    IF p.hash = hash AND StringDefs.EquivalentString[s,p.s] THEN RETURN[p.s];
    ENDLOOP;
  p ← SystemDefs.AllocateHeapNode[SIZE[Item]];
  p.link ← StringHead;
  StringHead ← p;
  p.hash ← hash;
  p.s ← SystemDefs.AllocateHeapString[s.length];
  StringDefs.AppendString[p.s,s];
  RETURN[p.s]
  END;

Item: TYPE = RECORD [
  link: POINTER TO Item,
  hash: WORD,
  name: STRING,
  fp: AltoFileDefs.FP];

ItemHead: ARRAY FileType OF POINTER TO Item;

NewItem: PROCEDURE [s: STRING] RETURNS [p: POINTER TO Item] =
  BEGIN
  p ← SystemDefs.AllocateHeapNode[SIZE[Item]];
  p.hash ← Hash[s];
  p.name ← CopyString[s];
  RETURN
  END;

LookupItem: PROCEDURE [s: STRING, t: FileType] RETURNS [p: POINTER TO Item] =
```

```
    BEGIN
    hash: HashVal = Hash[s];
    FOR p ← ItemHead[t], p.link UNTIL p = NIL DO
      IF p.hash = hash AND StringDefs.EquivalentString[s,p.name] THEN EXIT;
      ENDLOOP;
    RETURN
    END;

  EnterItem: PROCEDURE [s: STRING, t: FileType] RETURNS [p: POINTER TO Item] =
    BEGIN
    IF (p←LookupItem[s,t]) = NIL THEN
      BEGIN
      p ← NewItem[s];
      p.link ← ItemHead[t];
      ItemHead[t] ← p;
      END;
    RETURN
    END;

  GetFiles: PROCEDURE [fp: POINTER TO FP, fn: STRING] RETURNS [BOOLEAN] =
    BEGIN
    p: POINTER TO Item;
    ext: STRING ← [40];
    StripExtension[fn,ext];
    FOR i IN FileType DO
      IF EquivalentString[ext,extensions[i]] THEN
        BEGIN
        p ← EnterItem[fn,i];
        p.fp ← fp↑;
        EXIT;
        END;
    ENDLOOP;
    RETURN [FALSE]
    END;


  -- main program

  FileType: TYPE = {source, config, bcd, symbols, code, image};

  extensions: ARRAY FileType OF STRING = ["mesa", "config", "bcd", "symbols", "code", "image"];

  VersionProc: TYPE = PROCEDURE [info: POINTER TO Item, type: FileType];

  dateproc: ARRAY FileType OF VersionProc = [SourceDate, SourceDate, BinaryVersion, BinaryVersion, Bina
**ryVersion, BinaryVersion];

  i: FileType;
  name: STRING ← [100];

  interactive: BOOLEAN;
  pause, pauseDflt: BOOLEAN ← FALSE;
  verbose, verboseDflt: BOOLEAN ← FALSE;
  item: POINTER TO Item;

  SetCommandInput[];
  FOR i IN FileType DO ItemHead[i] ← NIL ENDLOOP;
  DirectoryDefs.EnumerateDirectory[GetFiles];
  pauseDflt ← pause;
  verboseDflt ← verbose;
  WriteChar[CR];
  WHILE NextFile[name] DO
    pause ← pauseDflt;
    verbose ← verboseDflt;
    StripSwitches[name];
    IF name.length = 0 THEN
      BEGIN
      pauseDflt ← pause;
      verboseDflt ← verbose;
      IF interactive THEN WriteChar[CR];
      END
    ELSE
      BEGIN
      StripExtension[name, NIL];
      IF ~interactive THEN WriteLine[name];
      FOR i IN FileType DO
```

```
      IF (item←LookupItem[name,i]) # NIL THEN
        BEGIN
        WriteChar[' ];
        WriteString[extensions[i]];
        WriteChar[':];
        dateproc[i][item,i];
        END;
      ENDLOOP;
    WriteChar[CR];
    IF pause THEN [] ← ReadChar[];
    END;
  ENDLOOP;
ImageDefs.StopMesa[];

END.
```