```
-- Zap.mesa
-- Edited by Sandman on September 12, 1977  9:06 AM

DIRECTORY
  AltoDefs: FROM "altodefs",
  BcdDefs: FROM "bcddefs",
  ImageDefs: FROM "imagedefs",
  InlineDefs: FROM "inlinedefs",
  IODefs: FROM "iodefs",
  SegmentDefs: FROM "segmentdefs",
  StringDefs: FROM "stringdefs",
  SymDefs: FROM "symdefs",
  SystemDefs: FROM "systemdefs",
  TimeDefs: FROM "timedefs";

DEFINITIONS FROM IODefs, SegmentDefs;

Zap: PROGRAM
  IMPORTS IODefs, SegmentDefs, StringDefs, SystemDefs, TimeDefs =

BEGIN

name: STRING ← [40];
filename: STRING ← [40];
file: FileHandle;
c: STRING ← [10];

BcdBase: TYPE = POINTER TO BcdDefs.BCD;

Confirm: PROCEDURE RETURNS [BOOLEAN] =
  BEGIN OPEN IODefs;
  WriteString[" [confirm]"];
  DO
    SELECT ReadChar[] FROM
      CR => RETURN[TRUE];
      DEL =>
        BEGIN
        WriteString[" XXX"];
        RETURN[FALSE];
        END;
      ENDCASE => WriteChar['?];
    ENDLOOP;
  END;

DisplayVersionStamp: PROCEDURE [stamp: BcdDefs.VersionStamp] =
  BEGIN OPEN IODefs;
  date: STRING ← [40];

  TimeDefs.AppendDayTime[date, TimeDefs.UnpackDT[stamp.time]];
  WriteString[date];
  WriteString[", "];
  WriteOctal[stamp.net];
  WriteChar['#];
  WriteOctal[stamp.host];
  WriteChar['#];
  IF stamp.zapped THEN WriteString[" zapped!!"];
  END;

FileZapper: PROCEDURE [bcd: BcdBase] =
  BEGIN OPEN BcdDefs,StringDefs;
  fti: FTIndex;
  ftb: CARDINAL = LOOPHOLE[bcd+bcd.ftOffset];
  filename: STRING ← [40];
  ss: SubStringDescriptor;
  DO
    WriteString["filename:  "];
    IODefs.ReadID[filename];
    IF filename.length = 0 THEN RETURN;
    ss ←[base: filename, offset: 0, length: filename.length];
    FOR fti ← FIRST[FTIndex], fti+SIZE[FTRecord]
      UNTIL fti = bcd.ftLimit DO
        OPEN f: ftb + fti;
        IF SameName[bcd,f.name,@ss] THEN
          BEGIN
          WriteString["  "];
          DisplayVersionStamp[f.version];
```

```
          IF Confirm[] THEN
             BEGIN f.version.zapped ← TRUE;
             WriteString["  . . . is being zapped."]; END;
          EXIT;
          END;
        REPEAT
          FINISHED => WriteString[" . . . can't find file!"];
        ENDLOOP;
  WriteChar[CR];
  ENDLOOP;
  END;

FindBcd: PROCEDURE [file: FileHandle] RETURNS [seg:FileSegmentHandle] =
  BEGIN
  pages: AltoDefs.PageCount;
  bcd: BcdBase;
  seg ← NewFileSegment[file, 1, 1, Read+Write];
  SwapIn[seg];
  bcd ← FileSegmentAddress[seg];
  IF (pages ← bcd.nPages) # 1 THEN
    BEGIN
    Unlock[seg];
    MoveFileSegment[seg, 1, pages];
    SwapIn[seg];
    bcd ← FileSegmentAddress[seg];
    END;
  IF bcd.versionident # BcdDefs.VersionID THEN
    BEGIN
    WriteString["  bad version ID "];
    WriteDecimal[bcd.versionident];
    Unlock[seg];
    DeleteFileSegment[seg];
    RETURN[NIL]
    END;
  END;

FindSymbols: PROCEDURE [file: FileHandle] RETURNS [seg:FileSegmentHandle] =
  BEGIN
  bcd: BcdBase;
  mtb: CARDINAL;
  mti: BcdDefs.MTIndex = FIRST[BcdDefs.MTIndex];
  pages: AltoDefs.PageCount;
  bcdseg: FileSegmentHandle;

  bcdseg ← NewFileSegment[file, 1, 1, Read+Write];
  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 THEN
    BEGIN
    WriteString["  bad version ID "];
    WriteDecimal[bcd.versionident];
    Unlock[bcdseg];
    DeleteFileSegment[bcdseg];
    RETURN[NIL]
    END;
  IF bcd.nModules # 1 THEN
    BEGIN
    WriteString["  too many modules: "];
    WriteDecimal[bcd.nModules];
    Unlock[bcdseg];
    DeleteFileSegment[bcdseg];
    RETURN[NIL]
    END;
  mtb ← LOOPHOLE[bcd,CARDINAL]+bcd.mtOffset;
  seg ← FindSegment[bcdseg, (mtb+mti).sseg, FALSE];
  IF seg # NIL THEN SwapIn[seg];
  Unlock[bcdseg];
  DeleteFileSegment[bcdseg];
  END;
```

```
FindSegment: PROCEDURE [seg: FileSegmentHandle, segdesc: BcdDefs.SegDesc, long: BOOLEAN]
  RETURNS [FileSegmentHandle] =
  BEGIN
  ss: StringDefs.SubStringDescriptor;
  file: SegmentDefs.FileHandle;
  name: STRING;
  bcd: BcdBase ← 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;

SameName: PROCEDURE [bcd: BcdBase, n: BcdDefs.NameRecord, name: StringDefs.SubString]
  RETURNS [BOOLEAN] =
  BEGIN OPEN StringDefs;
  ss: SubStringDescriptor ←
      [base: LOOPHOLE[bcd+bcd.ssOffset], offset: n.offset, length: n.length];
  RETURN[StringDefs.EquivalentSubStrings[@ss,name]];
  END;

ZapFiles: PROCEDURE [file: FileHandle]  =
  BEGIN
  bcdseg: FileSegmentHandle;
  bcd:BcdBase;

  bcdseg ← FindBcd[file];
  IF bcdseg # NIL THEN
    BEGIN
    bcd ← FileSegmentAddress[bcdseg];
    FileZapper[bcd];
    Unlock[bcdseg];
    DeleteFileSegment[bcdseg];
    END;
  END;

ZapHeader: PROCEDURE [file: FileHandle]  =
  BEGIN
  seg: FileSegmentHandle;
  bcd: BcdBase;
  seg ← NewFileSegment[file, 1, 1, Read+Write];
  SwapIn[seg];
  bcd ← FileSegmentAddress[seg];
  WriteString["Header: "];
  SELECT bcd.versionident FROM
   BcdDefs.VersionID =>
      BEGIN
      DisplayVersionStamp[bcd.version];
      IF Confirm[] THEN
        BEGIN bcd.version.zapped ← TRUE;
        WriteString["  . . . is being zapped."] END;
      END;
   ENDCASE =>
      BEGIN
      WriteString[" bad version ID "];
      WriteDecimal[bcd.versionident]
```

```
      END;
  Unlock[seg];
  DeleteFileSegment[seg];
  END;

ZapSymbols: PROCEDURE [file: FileHandle]  =
  BEGIN
  seg: FileSegmentHandle;
  stHeader: POINTER TO SymDefs.STHeader;
  seg ← FindSymbols[file];
  IF seg # NIL THEN
    BEGIN
    stHeader ← FileSegmentAddress[seg];
    WriteString["Symbol Table: "];
    DisplayVersionStamp[stHeader.version];
    IF Confirm[] THEN
      BEGIN stHeader.version.zapped ← TRUE;
      WriteString["  . . . is being zapped."]; END;
    Unlock[seg];
    DeleteFileSegment[seg];
    END;
  END;


-- main program

WriteLine["Mesa Bcd Zapper   Use With Caution!!!!!!"];

DO
  BEGIN ENABLE Rubout =>
    BEGIN
    WriteString[" XXX"];
    GOTO repeatloop;
    END;
  WriteChar[CR];
  WriteString["Zap: "];
  IODefs.ReadID[name];
  IF name.length=0 THEN EXIT;
  StringDefs.AppendString[filename, name];
  CheckForExtension[filename, ".bcd"];
  file ← NewFile[filename, Read+Write, OldFileOnly
    ! FileNameError =>
      BEGIN
      WriteString[" !File not found"];
      GOTO repeatloop;
      END];
  LockFile[file];
  DO ENABLE Rubout =>
      BEGIN
      WriteString[" XXX"];
      CONTINUE
      END;
    WriteChar[CR];
    WriteString["header, symbols, files, or quit? [H,S,F,Q] "];
    IODefs.ReadID[c]; WriteChar[CR];
    IF c.length = 0 THEN EXIT
    ELSE SELECT c[0] FROM
      'h,'H => ZapHeader[file];
      's,'S => ZapSymbols[file];
      'f,'F => ZapFiles[file];
      'q,'Q => EXIT;
      ENDCASE => WriteChar['?];
  ENDLOOP;
  UnlockFile[file];
  ReleaseFile[file];
  EXITS
  repeatloop => NULL;
  END;
  WriteChar[CR];
ENDLOOP;
ImageDefs.StopMesa[];
END.
```