```
-- BcdList.Mesa  Edited by Sandman on August 31, 1977  10:21 AM

DIRECTORY
  AltoDefs: FROM "altodefs",
  BcdDefs: FROM "bcddefs",
  CommanderDefs: FROM "commanderdefs",
  ControlDefs: FROM "controldefs",
  IODefs: FROM "iodefs",
  ListerDefs: FROM "listerdefs",
  OutputDefs: FROM "outputdefs",
  SegmentDefs: FROM "segmentdefs",
  StringDefs: FROM "stringdefs",
  SymDefs: FROM "symdefs";

DEFINITIONS FROM OutputDefs, BcdDefs;

BcdList: PROGRAM
  IMPORTS CommanderDefs, IODefs, OutputDefs, SegmentDefs, StringDefs
  EXPORTS ListerDefs =
  BEGIN

  bcd: POINTER TO BCD;
  tb: CARDINAL;
  ssb: STRING;
  ctb: CARDINAL;
  mtb: CARDINAL;
  itb: CARDINAL;
  etb: CARDINAL;
  ftb: CARDINAL;
  ntb: CARDINAL;

  InstallBcd: PROCEDURE [seg: SegmentDefs.FileSegmentHandle] =
    BEGIN OPEN SegmentDefs;
    size: CARDINAL;
    IF ~seg.swappedin THEN SwapIn[seg];
    bcd ← FileSegmentAddress[seg];
    IF (size←bcd.nPages) # seg.pages THEN
      BEGIN
      Unlock[seg];
      MoveFileSegment[seg,seg.base,size];
      SwapIn[seg];
      bcd ← FileSegmentAddress[seg];
      END;
    tb ← LOOPHOLE[bcd];
    ssb ← LOOPHOLE[bcd+bcd.ssOffset];
    ctb ← tb+bcd.ctOffset;
    mtb ← tb+bcd.mtOffset;
    itb ← tb+bcd.impOffset;
    etb ← tb+bcd.expOffset;
    ftb ← tb+bcd.ftOffset;
    ntb ← tb+bcd.ntOffset;
    RETURN
    END;

  UnstallBcd: PROCEDURE [seg: SegmentDefs.FileSegmentHandle] =
    BEGIN OPEN SegmentDefs;
    IF seg.swappedin THEN Unlock[seg];
    SwapOut[seg];
    RETURN
    END;

  WriteBcdID: PROCEDURE [name: STRING, bcd: POINTER TO BCD] =
    BEGIN
    PutString[name];
    PutString[" configured "];
    PutTime[bcd.version.time];
    IF bcd.source # [0,0] THEN
      BEGIN
      PutString[" from "];
      PutName[bcd.source];
      END;
    PutString[" by "];
    PrintMachine[bcd.version];
    IF bcd.versionident # VersionID THEN
      BEGIN PutString["  Obsolete VersionID = "];
      PutDecimal[bcd.versionident] END;
```

```
    PutCR[];
    PutString["  Configured  by "];
    PutTime[bcd.creator.time];
    PutString[" "];
    PrintMachine[bcd.creator];
    PutCR[]; PutCR[];
    RETURN
    END;

  PrintBcd: PUBLIC PROCEDURE =
    BEGIN
    PrintHeader[];  PrintConfigs[];
    PrintImports[];  PrintExports[];
    PrintModules[];  PrintFiles[];
    RETURN
    END;

  PrintHeader: PUBLIC PROCEDURE =
    BEGIN
    PutString["Configurations: "]; PutDecimal[bcd.nConfigs];
    PutString[", Modules: "]; PutDecimal[bcd.nModules];
    PutString[", Imports: "]; PutDecimal[bcd.nImports];
    PutString[", Exports: "]; PutDecimal[bcd.nExports];
    PutString[", Dummy: "]; PutDecimal[bcd.firstdummy];
    PutString[", #Dummies: "]; PutDecimal[bcd.nDummies];
    PutCR[];  PutCR[];
    RETURN
    END;

  PrintConfigs: PUBLIC PROCEDURE =
    BEGIN
    cti: CTIndex ← FIRST[CTIndex];
    PutString["Configurations"];
    PrintIndex[bcd.ctOffset];
    PutChar[':]; PutCR[];
    UNTIL cti = bcd.ctLimit DO
      PrintConfig[cti];
      cti ← cti + SIZE[CTRecord];
      IF LOOPHOLE[cti,CARDINAL] > LOOPHOLE[bcd.ctLimit,CARDINAL] THEN GO TO Bogus;
      REPEAT Bogus => PrintGarbage[];
      ENDLOOP;
    PutCR[];
    RETURN
    END;

  PrintConfig: PUBLIC PROCEDURE [cti: CTIndex] =
    BEGIN OPEN ctb+cti;
    Tab[2];
    PutName[name];  PrintIndex[cti];
    IF namedinstance THEN
      BEGIN
      PutString[", instance: "];
      PutInstanceName[[config[cti]]];
      END;
    PutString[", file: "];
    PrintFileName[file];  PrintIndex[file];
    IF config # CTNull THEN
      BEGIN PutString[", parent: "];
      PutName[(ctb+config).name];
      PrintIndex[config];
      END;
    IF control # MTNull THEN
      BEGIN  PutString[", control: "];
      PutName[(mtb+control).name];
      PrintIndex[control];
      END;
    PutCR[];
    RETURN
    END;

  PrintImports: PUBLIC PROCEDURE =
    BEGIN
    iti: IMPIndex ← FIRST[IMPIndex];
    PutString["Imports"];
    PrintIndex[bcd.impOffset];
    PutChar[':]; PutCR[];
```

```
    UNTIL iti = bcd.impLimit DO
      PrintImport[iti];
      iti ← iti + SIZE[IMPRecord];
      IF LOOPHOLE[iti,CARDINAL] > LOOPHOLE[bcd.impLimit,CARDINAL] THEN GO TO Bogus;
      REPEAT Bogus => PrintGarbage[];
      ENDLOOP;
    PutCR[];
    RETURN
    END;

  PrintImport: PUBLIC PROCEDURE [iti: IMPIndex] =
    BEGIN OPEN itb+iti;
    Tab[2];
    PutName[name];  PrintIndex[iti];
    IF port = module THEN PutString[" (module)"];
    IF namedinstance THEN
      BEGIN
      PutString[", instance: "];
      PutInstanceName[[import[iti]]];
      END;
    PutString[", file: "];
    PrintFileName[file];  PrintIndex[file];
    PutString[", gfi: "];
    PutDecimal[gfi];
    PutString[", ngfi: "];
    PutDecimal[ngfi];
    PutCR[];
    RETURN
    END;

  PrintExports: PUBLIC PROCEDURE =
    BEGIN
    eti: EXPIndex ← FIRST[EXPIndex];
    PutString["Exports"];
    PrintIndex[bcd.expOffset];
    PutChar[':]; PutCR[];
    UNTIL eti = bcd.expLimit DO
      PrintExport[eti];
      eti ← eti + (etb+eti).size + SIZE[EXPRecord]-1;
      IF LOOPHOLE[eti,CARDINAL] > LOOPHOLE[bcd.expLimit,CARDINAL] THEN GO TO Bogus;
      REPEAT Bogus => PrintGarbage[];
      ENDLOOP;
    PutCR[];
    RETURN
    END;

  PrintExport: PUBLIC PROCEDURE [eti: EXPIndex] =
    BEGIN OPEN etb+eti;
    i: CARDINAL;  Tab[2];
    PutName[name];  PrintIndex[eti];
    IF port = module THEN PutString[" (module)"];
    IF namedinstance THEN
      BEGIN
      PutString[", instance: "];
      PutInstanceName[[export[eti]]];
      END;
    PutString[", file: "];
    PrintFileName[file];  PrintIndex[file];
    PutString[", size: "];
    PutDecimal[size];
    IF DumpLinks THEN
      BEGIN
      PutString[", links:"];
      FOR i IN [0..size) DO
        IF i MOD 8 = 0 THEN Tab[4] ELSE PutChar[' ];
        PrintControlLink[links[i]];
        IF i+1 # size THEN PutChar[',];
        ENDLOOP;
      END;
    PutCR[];
    RETURN
    END;

  PrintModules: PUBLIC PROCEDURE =
    BEGIN
    mti: MTIndex ← FIRST[MTIndex];
```

```
    PutString["Modules"];
    PrintIndex[bcd.mtOffset];
    PutChar[':]; PutCR[];
    UNTIL mti = bcd.mtLimit DO
      PrintModule[mti];
      mti ← mti + SIZE[MTRecord]-1+(mtb+mti).frame.length;
      IF LOOPHOLE[mti,CARDINAL] > LOOPHOLE[bcd.mtLimit,CARDINAL] THEN GO TO Bogus;
      REPEAT Bogus => PrintGarbage[];
      ENDLOOP;
    PutCR[];
    RETURN
    END;

  PrintModule: PUBLIC PROCEDURE [mti: MTIndex] =
    BEGIN OPEN mtb+mti;
    i: CARDINAL;  Tab[2];
    PutName[name];  PrintIndex[mti];
    IF namedinstance THEN
      BEGIN
      PutString["instance: "];
      PutInstanceName[[module[mti]]];
      END;
    PutString[", file: "];
    PrintFileName[file];  PrintIndex[file];
    IF config # CTNull THEN
      BEGIN
      PutString[", config: "];
      PutName[(ctb+config).name];
      PrintIndex[config];
      END;
    PutString[", fsi: "];  PutDecimal[IF fsi = ControlDefs.maxallocslot THEN framesize ELSE fsi];
    PutString[", gfi: "];  PutDecimal[gfi];
    PutString[", ngfi: "];  PutDecimal[ngfi];
    Tab[4];
    PutString["code: "];  PrintSegDesc[cseg];
    PutString[", symbols: "];  PrintSegDesc[sseg];
    BEGIN OPEN frame;  Tab[4];
      PutString["frame offset: "];  PutDecimal[offset];
      PutString[", frame length: "];  PutDecimal[length];
      IF DumpLinks THEN
        BEGIN
        PutString[", control links:"];
        FOR i IN [0..length) DO
          IF i MOD 8 = 0 THEN Tab[6] ELSE PutChar[' ];
          PrintControlLink[frag[i]];
          IF i+1 # length THEN PutChar[',];
          ENDLOOP;
        END;
      END;
    PutCR[];
    RETURN
    END;

  PrintSegDesc: PUBLIC PROCEDURE [sd: SegDesc] =
    BEGIN
    PrintFileName[sd.file]; PutString[" [base: "];
    PutDecimal[sd.base]; PutString[", pages: "];
    PutDecimal[sd.pages];
    IF sd.extraPages # 0 THEN
      BEGIN PutChar['+]; PutDecimal[sd.extraPages]; END;
    PutChar[']];
    RETURN
    END;

  PrintFiles: PUBLIC PROCEDURE =
    BEGIN
    fti: FTIndex ← FIRST[FTIndex];
    PutString["Files"];
    PrintIndex[bcd.ftOffset];
    PutChar[':]; PutCR[];
    UNTIL fti = bcd.ftLimit DO
      PrintFile[fti];
      fti ← fti + SIZE[FTRecord];
      IF LOOPHOLE[fti,CARDINAL] > LOOPHOLE[bcd.ftLimit,CARDINAL] THEN GO TO Bogus;
      REPEAT Bogus => PrintGarbage[];
      ENDLOOP;
```

```
    PutCR[];
    RETURN
    END;

  PrintFile: PUBLIC PROCEDURE [fti: FTIndex] =
    BEGIN OPEN ftb+fti;
    Tab[2];
    SELECT fti FROM
      FTNull => PutString["(null)"];
      FTSelf => PutString["(self)"];
      ENDCASE =>
        BEGIN
        PutName[name];  PrintIndex[fti];
        PutString[", time: "];
        PutTime[version.time];
        PutString[", processor: "];
        PrintMachine[version];
        END;
    PutCR[];
    RETURN
    END;


  -- Utility Prints

  PrintControlLink: PROCEDURE [link: ControlLink] =
    BEGIN
    map: ARRAY ControlLinkTag OF CHARACTER = ['0,'1,'2,'3];
    PutChar['[]; PutDecimal[link.gfi];
    PutChar[',]; PutDecimal[link.ep];
    PutChar[',]; PutChar[map[link.tag]];
    PutChar[']]; RETURN
    END;

  PrintMachine: PROCEDURE [stamp: BcdDefs.VersionStamp] =
    BEGIN
    octal: NumberFormat = [8,FALSE,FALSE,1];
    PutNumber[stamp.net, octal];
    PutChar['#];
    PutNumber[stamp.host, octal];
    PutChar['#];
    IF stamp.zapped THEN PutString[" zapped!"];
    RETURN
    END;

  PrintFileName: PROCEDURE [fti: FTIndex] =
    BEGIN
    SELECT fti FROM
      FTNull => PutString["(null)"];
      FTSelf => PutString["(self)"];
      ENDCASE => PutName[(ftb+fti).name];
    RETURN
    END;

  PrintIndex: PROCEDURE [index: UNSPECIFIED] =
    BEGIN
    PutString[" ["]; PutDecimal[index];
    PutChar[']]; RETURN
    END;

  PrintGarbage: PROCEDURE =
    BEGIN Tab[2];
    PutString["? looks like garbage to me ..."];
    PutCR[]; RETURN
    END;

  PrintAnonName: PROCEDURE =
    BEGIN
    PutString[" (anon) "];
    RETURN
    END;


  -- Utility Puts

  PutSubString: PROCEDURE [ss: StringDefs.SubString] =
```

```
    BEGIN  i: CARDINAL;
    FOR i IN [ss.offset..ss.offset+ss.length) DO
      PutChar[ss.base[i]]
      ENDLOOP;
    RETURN
    END;

  PutName: PUBLIC PROCEDURE [n: NameRecord] =
    BEGIN
    ssd: StringDefs.SubStringDescriptor ← [
      base: ssb, offset: n.offset, length: n.length];
    PutSubString[@ssd];
    RETURN
    END;

  Indent: PROCEDURE [n: CARDINAL] =
    BEGIN
    THROUGH [1..n/8] DO PutChar[IODefs.TAB] ENDLOOP;
    THROUGH [1..n MOD 8] DO PutChar[' ] ENDLOOP;
    RETURN
    END;

  Tab: PROCEDURE [n: CARDINAL] =
    BEGIN PutCR[];
    Indent[n];
    RETURN
    END;

  PutInstanceName: PROCEDURE [n: Namee] =
    BEGIN
    FindName: PROCEDURE [ntb: CARDINAL, nti: NTIndex] RETURNS [BOOLEAN] =
      BEGIN
      RETURN[(ntb+nti).item = n];
      END;
    nti: NTIndex;
    IF (nti ← EnumerateNameTable[FindName]) = NTNull
      THEN PrintAnonName[]
      ELSE PutName[(ntb+nti).name];
    END;

  EnumerateNameTable: PROCEDURE [
   proc: PROCEDURE [CARDINAL, NTIndex] RETURNS [BOOLEAN]]
   RETURNS [nti: NTIndex] =
    BEGIN
    FOR nti ← FIRST[NTIndex], nti + SIZE[NTRecord] DO
      IF proc[ntb, nti] THEN RETURN [nti];
      ENDLOOP;
    RETURN [NTNull];
    END;

  -- IncorrectVersion: EXTERNAL SIGNAL;

  Bcd: PROCEDURE [root: STRING] =
    BEGIN
    i: CARDINAL;
    bcdfile: STRING ← [40];
    seg: SegmentDefs.FileSegmentHandle;
    BEGIN OPEN StringDefs;
      AppendString[bcdfile,root];
      FOR i IN [0..bcdfile.length) DO
        IF bcdfile[i] = '. THEN EXIT;
        REPEAT FINISHED => AppendString[bcdfile,".bcd"];
        ENDLOOP;
      END;
    BEGIN OPEN SegmentDefs;
      seg ← NewFileSegment[
        NewFile[bcdfile, Read, DefaultVersion !
          FileNameError => GO TO NoFile],
        1, 1, Read];
      InstallBcd[seg];
      OpenOutput[root,".bl"];
      WriteBcdID[bcdfile,bcd];
      PrintBcd[];  CloseOutput[];
      UnstallBcd[seg];
      EXITS
        NoFile => IODefs.WriteString["File not found"];
```

```
    END;
  RETURN
  END;

BcdLinks: PROCEDURE [root: STRING] =
  BEGIN
  DumpLinks ← TRUE;
  Bcd[root];
  DumpLinks ← FALSE;
  RETURN
  END;

BcdSegment: PROCEDURE [
 root: STRING, base: AltoDefs.PageNumber, pages: AltoDefs.PageCount, links: BOOLEAN] =
  BEGIN
  i: CARDINAL;
  bcdfile: STRING ← [40];
  seg: SegmentDefs.FileSegmentHandle;
  DumpLinks ← links;
  BEGIN OPEN StringDefs;
    AppendString[bcdfile, root];
    FOR i IN [0..bcdfile.length) DO
      IF bcdfile[i] = '. THEN EXIT;
      REPEAT FINISHED => AppendString[bcdfile,".bcd"];
      ENDLOOP;
    END;
  BEGIN OPEN SegmentDefs;
    seg ← NewFileSegment[
      NewFile[bcdfile, Read, DefaultVersion !
        FileNameError => GO TO NoFile],
      base, pages, Read !
        InvalidSegmentSize => GO TO BadSegment];
    InstallBcd[seg ! SwapError, SegmentFault => GO TO BadSegment];
    OpenOutput[root,".bl"];
    WriteBcdID[bcdfile,bcd];
    PrintBcd[];  CloseOutput[];
    UnstallBcd[seg];
    EXITS
      NoFile => IODefs.WriteString["File not found"];
      BadSegment => IODefs.WriteString["Bad Segment"];
    END;
  DumpLinks ← FALSE;
  END;

DumpLinks: BOOLEAN ← FALSE;

command: CommanderDefs.CommandBlockHandle;

command ← CommanderDefs.AddCommand["Bcd", LOOPHOLE[Bcd], 1];
command.params[0] ← [type: string, prompt: "Filename"];

command ← CommanderDefs.AddCommand["BcdLinks", LOOPHOLE[BcdLinks], 1];
command.params[0] ← [type: string, prompt: "Filename"];

command ← CommanderDefs.AddCommand["BcdSegment", LOOPHOLE[BcdSegment], 4];
command.params[0] ← [type: string, prompt: "Filename"];
command.params[1] ← [type: numeric, prompt: "Base"];
command.params[2] ← [type: numeric, prompt: "Pages"];
command.params[3] ← [type: boolean, prompt: "Links"];


END....
```