```
-- file listsymbols.mesa
-- last modified by Sandman, October 17, 1977  9:10 AM

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

DEFINITIONS FROM ListerDefs, OutputDefs, SymDefs;

ListSymbols: PROGRAM IMPORTS ListerDefs, CommanderDefs, IODefs, OutputDefs, SegmentDefs, StringDefs, Sy
**mbolTableDefs
  EXPORTS ListerDefs =
  BEGIN
  FileSegmentHandle: TYPE = SegmentDefs.FileSegmentHandle;

  symbols: SymbolTableDefs.SymbolTableBase;

  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;

  PrintSymbols: PROCEDURE =
    BEGIN
    ctx: CTXIndex;
    ctx ← FIRST[CTXIndex];
    UNTIL ctx = LOOPHOLE[symbols.stHandle.ctxSize,CTXIndex]
      DO
      PutCR;  PrintContext[ctx];
      ctx ← ctx + (WITH (symbols.ctxb+ctx) SELECT FROM
        included => SIZE [included CTXRecord],
        imported => SIZE [imported CTXRecord],
        ENDCASE => SIZE [simple CTXRecord]);
      ENDLOOP;
    PutCR;  RETURN
    END;

  PrintContext: PROCEDURE [ctx: CTXIndex] =
    BEGIN
    sei, root: ISEIndex;
    typesei: SEIndex;
    addr: bitaddress;
    PutCR;
    PutString["Context: "];  printindex[ctx];
    IF (symbols.ctxb+ctx).ctxlevel # 1Z THEN
      BEGIN  PutString[", static level: "];
      PutDecimal[(symbols.ctxb+ctx).ctxlevel];
      END;
    WITH (symbols.ctxb+ctx) SELECT FROM
      included =>
        BEGIN  PutString[", copied from [file: "];
        printhti[(symbols.mdb+ctxmodule).mdhti];
        PutString[", context: "];  printindex[ctxmap];
        PutChar[']];
        END;
      imported =>
        BEGIN  PutString[", imported from file: "];
        printhti[(symbols.mdb+(symbols.ctxb+includeLink).ctxmodule).mdhti];
        END;
      ENDCASE;
    root ← sei ← (symbols.ctxb+ctx).selist;
    WHILE sei # SENull DO
      OPEN (symbols.seb+sei);
```

```
      indent[2];
      printsei[sei];
      PutString[" ["];  printindex[sei];  PutChar[']];
      IF public THEN PutString[" [public]"];
      IF external THEN PutString[" [external]"];
      IF mark3
        THEN
          BEGIN
          PutString[", type = "];
          IF idtype = typeTYPE
            THEN
              BEGIN  typesei ← idinfo;
              PutString["TYPE, equated to: "];
              printtype[typesei];
              IF typelink[sei] # SENull
                THEN
                  BEGIN  PutString[", tag code: "];
                  PutDecimal[idvalue];
                  END;
              END
            ELSE
              BEGIN  printtype[idtype];  typesei ← idtype;
              IF writeonce THEN PutString[" [read only]"];
              IF constant THEN PutString[" [constant]"];
              IF ~mark4
                THEN
                  BEGIN  PutString[", references: "];
                  PutDecimal[idinfo];
                  END
                ELSE
                  BEGIN
                  SELECT TRUE FROM
                    constant =>
                      BEGIN  PutString[", value: "];
                      IF ABS[idvalue] <= 1000
                        THEN  PutDecimal[idvalue]
                        ELSE  PutOctal[idvalue];
                      END;
                    (external AND symbols.stHandle.definitionsFile) =>
                      BEGIN  PutString[", index: "];
                      PutDecimal[idvalue];
                      END;

                    ENDCASE =>
                      BEGIN  addr ← idvalue;
                      PutString[", address: "];
                      PutOctal[addr.wd];  PutChar[' ];
                      PutChar['[];  PutOctal[addr.bd];
                      PutChar[':];  PutOctal[idinfo];
                      PutChar[']];
                      END;
                  END;
              END;
          printtypeinfo[typesei, 4];
          END;
      IF (sei ← symbols.nextse[sei]) = root THEN EXIT;
      ENDLOOP;
    RETURN
    END;


  printhti: PROCEDURE [hti: HTIndex] =
    BEGIN
    desc: StringDefs.SubStringDescriptor;
    s: StringDefs.SubString = @desc;
    IF hti = HTNull
      THEN PutString["(anonymous)"]
      ELSE
        BEGIN
        symbols.SubStringForHash[s, hti];  PutSubString[s];
        END;
    RETURN
    END;

  printsei: PROCEDURE [sei: ISEIndex] =
    BEGIN
```

```
    printhti[IF sei=SENull THEN HTNull ELSE (symbols.seb+sei).htptr];
    RETURN
    END;

  TypePrintName: ARRAY TypeClass OF STRING = [
    "mode", "basic", "enumerated", "record", "pointer", "array",
    "arraydesc", "transfer", "definition", "union", "subrange"];

  PutTypeName: PROCEDURE[n: TypeClass] =
    BEGIN
    PutString[TypePrintName[n]];  RETURN
    END;

  ModePrintName: ARRAY TransferMode OF STRING = [
    "procedure", "port", "signal", "error", "program", "inline", "none"];

  PutModeName: PROCEDURE[n: TransferMode] =
    BEGIN
    PutString[ModePrintName[n]];  RETURN
    END;

  typelink: PROCEDURE [type: SEIndex] RETURNS [SEIndex] =
    BEGIN
    sei: CSEIndex = symbols.undertype[type];
    RETURN [WITH se: (symbols.seb+sei) SELECT FROM
      record =>
        WITH se SELECT FROM
          linked => linktype,
          ENDCASE => SENull,
      ENDCASE => SENull]
    END;

  printtype: PROCEDURE [sei: SEIndex] =
    BEGIN
    tsei: SEIndex;
    IF sei = SENull
      THEN PutString["?"]
      ELSE
        WITH t: (symbols.seb+sei) SELECT FROM
          constructor =>
            WITH t SELECT FROM
              transfer => PutModeName[mode];
              ENDCASE =>  PutTypeName[t.typetag];
          id =>
            BEGIN
            printsei[LOOPHOLE[sei, ISEIndex]];  tsei ← sei;
            UNTIL (tsei ← typelink[tsei]) = SENull
              DO
              WITH (symbols.seb+tsei) SELECT FROM
                id =>
                  BEGIN  PutChar[' ];
                  printsei[LOOPHOLE[tsei, ISEIndex]];
                  END;
              ENDCASE;
            ENDLOOP;
          END;
        ENDCASE;
    PutString[" ["];  printindex[sei];  PutChar[']];
    RETURN
    END;

  printtypeinfo: PROCEDURE [sei: SEIndex, nblanks: INTEGER] =
    BEGIN
    IF sei # SENull
      THEN
        WITH s: (symbols.seb+sei) SELECT FROM
          constructor =>
            BEGIN  indent[nblanks];
            PutChar['[];  printindex[sei];  PutString["] "];
            WITH s SELECT FROM
              transfer => PutModeName[mode];
              ENDCASE => PutTypeName[s.typetag];
            WITH t: s SELECT FROM
              basic =>
                BEGIN
                IF t.ordered THEN PutString[" (ordered)"];
```

```
      PutString[", code: "];
      PutDecimal[t.code];
      PutString[", length: "];  PutOctal[t.length];
      END;
    enumerated =>
      BEGIN
      IF t.ordered THEN PutString[" (ordered)"];
      PutString[", value ctx: "];
      printindex[t.valuectx];
      PutString[", nvalues: "];
      PutDecimal[t.nvalues];
      END;
    record =>
      BEGIN
      IF t.machineDep THEN PutString[" (Machine Dependent)"];
      IF t.unifield THEN PutString[" (unifield)"];
      IF t.variant THEN PutString[" (variant)"];
      outrecordctx[", field ctx: ", LOOPHOLE[sei, recordCSEIndex]];
      WITH (symbols.ctxb+t.fieldctx) SELECT FROM
        included =>
          IF ~ctxcomplete THEN PutString[" [partial]"];
        imported => PutString[" [partial]"];
        ENDCASE;
      WITH t SELECT FROM
        linked =>
          BEGIN  PutString[", link: "];
          printtype[linktype];
          END;
        ENDCASE;
      END;
    pointer =>
      BEGIN
      IF t.ordered THEN PutString[" (ordered)"];
      IF t.readonly THEN PutString[" (readonly)"];
      PutString[", pointing to: "];
      printtype[t.pointedtotype];
      printtypeinfo[t.pointedtotype, nblanks+2];
      END;
    array, arraydesc =>
      BEGIN
      IF t.packed THEN PutString[" (packed)"];
      PutString[", index type: "];
      printtype[t.indextype];
      PutString[", component type: "];
      printtype[t.componenttype];
      printtypeinfo[t.indextype, nblanks+2];
      printtypeinfo[t.componenttype, nblanks+2];
      END;
    transfer =>
      BEGIN
      outrecordctx[", input ctx: ", t.inrecord];
      outrecordctx[", output ctx: ", t.outrecord];
      END;
    definition =>
      BEGIN
      PutString[", ctx: "];  printindex[t.defCtx];
      PutString[", number of gfi's: "];
      PutDecimal[t.nGfi];
      END;
    union =>
      BEGIN
      IF t.overlayed THEN PutString[" (overlayed) "];
      IF t.controlled
        THEN
          BEGIN PutString[", tag: "]; printsei[t.tagsei];
          END;
      PutString[", tag type: "];
      printtype[(symbols.seb+t.tagsei).idtype];
      PutString[", case ctx: "];  printindex[t.casectx];
      END;
    subrange =>
      BEGIN
      PutString[" of: "];  printtype[t.rangetype];
      IF t.empty THEN PutString[" (empty) "];
      IF t.filled
        THEN
```

```
                    BEGIN
                    PutString[" origin: "];  PutDecimal[t.origin];
                    PutString[", range: "];
                    IF t.flexible
                      THEN PutChar['*]
                      ELSE PutDecimal[t.range];
                    END;
                END;
              ENDCASE;
            END;
          ENDCASE;
    RETURN
    END;

  outrecordctx: PROCEDURE [message: STRING, sei: recordCSEIndex] =
    BEGIN
    PutString[message];
    IF sei = SENull
      THEN PutString["NIL"]
      ELSE printindex[(symbols.seb+sei).fieldctx];
    RETURN
    END;

  printindex: PROCEDURE [v: UNSPECIFIED] =
    BEGIN
    PutDecimal[v];  RETURN
    END;

  indent: PROCEDURE [n: CARDINAL] =
    BEGIN
    PutCR;
    THROUGH [1..n/8] DO PutTab[] ENDLOOP;
    THROUGH [1..n MOD 8] DO PutChar[' ] ENDLOOP;
    RETURN
    END;


Symbols: PROCEDURE[root: STRING] =
  BEGIN OPEN StringDefs;
  i: CARDINAL;
  bcdFile: STRING ← [40];
  sseg: FileSegmentHandle;
  AppendString[bcdFile,root];
  FOR i IN [0..bcdFile.length) DO
    IF bcdFile[i] = '. THEN EXIT;
    REPEAT FINISHED => AppendString[bcdFile,".bcd"];
    ENDLOOP;
  BEGIN
  sseg ← Load[bcdFile
    !NoCode => RESUME;
    NoFGT => RESUME;
    NoSymbols, IncorrectVersion =>
      BEGIN IODefs.WriteString["Bad format"]; GOTO giveup END;
    SegmentDefs.FileNameError =>
      BEGIN IODefs.WriteString["File not found"]; GOTO giveup END
    ].symbolseg;
  symbols←SymbolTableDefs.AcquireSymbolTable[
    SymbolTableDefs.TableForSegment[sseg]];
  -- this kludge because ctxLimit not in SymbolTable frame
  OpenOutput[root,".sl"];
  WriteFileID[bcdFile];
  PrintSymbols[];
  CloseOutput[];
  EXITS giveup => NULL;
  END;
  END;

SymbolSegment: PROCEDURE[root: STRING, base: AltoDefs.PageNumber, pages: AltoDefs.PageCount] =
  BEGIN OPEN StringDefs;
  i: CARDINAL;
  bcdFile: STRING ← [40];
  sseg: FileSegmentHandle;
  AppendString[bcdFile,root];
  FOR i IN [0..bcdFile.length) DO
    IF bcdFile[i] = '. THEN EXIT;
    REPEAT FINISHED => AppendString[bcdFile,".bcd"];
```

```
    ENDLOOP;
  BEGIN OPEN SegmentDefs;
    sseg ← NewFileSegment[
      NewFile[bcdFile, Read, DefaultVersion !
        FileNameError => GO TO NoFile],
      base, pages, Read !
        InvalidSegmentSize => GO TO BadSegment];
    sseg.class ← symbols;
    symbols ← SymbolTableDefs.AcquireSymbolTable[SymbolTableDefs.TableForSegment[sseg !
        InvalidSegmentSize => GO TO BadSegment]];
    -- this kludge because ctxLimit not in SymbolTable frame
    OpenOutput[root,".sl"];
    PutString["Symbol Table in file: "];
    PutString[root];
    PutString[", base: "];
    PutDecimal[base];
    PutString[", pages: "];
    PutDecimal[pages];
    PutCR[];
    WriteSymbolID[];
    PrintSymbols[];
    CloseOutput[];
    EXITS
      NoFile => IODefs.WriteString["File not found"];
      BadSegment => IODefs.WriteString["Bad Segment"];
    END;
  END;

WriteSymbolID: PROCEDURE =
  BEGIN OPEN symbols.stHandle;
  octal3: NumberFormat =
    NumberFormat[base: 8, columns: 3, zerofill: FALSE, unsigned: TRUE];
  PutString["  Created "];
  PutTime[version.time];
  PutString[" by "];
  PutNumber[version.net,octal3];
  PutChar['#];
  PutNumber[version.host,octal3];
  PutChar['#];
  IF version.zapped THEN PutString[" zapped!!!"];
  PutCR[];
  PutString["  Creator "];
  PutTime[creator.time];
  PutString[" "];
  PutNumber[creator.net,octal3];
  PutChar['#];
  PutNumber[creator.host,octal3];
  PutChar['#];
  IF creator.zapped THEN PutString[" zapped!!!"];
  PutCR[]; PutCR[];
  RETURN
  END;

command: CommanderDefs.CommandBlockHandle;

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

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

END...
```