```
-- ListSymbols.mesa; modified by Johnsson, July 20, 1978  1:42 PM

DIRECTORY
  AltoDefs: FROM "altodefs",
  BcdDefs: FROM "bcddefs",
  CommanderDefs: FROM "commanderdefs",
  IODefs: FROM "iodefs",
  ListerDefs: FROM "listerdefs",
  LitDefs: FROM "litdefs",
  OutputDefs: FROM "outputdefs",
  SegmentDefs: FROM "segmentdefs",
  StringDefs: FROM "stringdefs",
  SymbolTableDefs: FROM "symboltabledefs",
  SymDefs: FROM "symdefs",
  SymSegDefs: FROM "symsegdefs",
  SymTabDefs: FROM "symtabdefs",
  TableDefs: FROM "tabledefs",
  TreeDefs: FROM "treedefs";

DEFINITIONS FROM ListerDefs, OutputDefs, SymDefs;

ListSymbols: PROGRAM IMPORTS CommanderDefs, IODefs, ListerDefs, LitDefs, OutputDefs, SegmentDefs, Strin
**gDefs, SymbolTableDefs, SymSegDefs, TableDefs, TreeDefs
  EXPORTS ListerDefs =
  BEGIN
  FileSegmentHandle: TYPE = SegmentDefs.FileSegmentHandle;
  CR: CHARACTER = IODefs.CR;

  symbols: SymbolTableDefs.SymbolTableBase;

  -- utilities

  seb, ctxb, htb, ltb, tb: TableDefs.TableBase;

  ListSymbolsNotify: TableDefs.TableNotifier =
    BEGIN
    tb ← base[TreeDefs.treetype];
    ltb ← base[LitDefs.lttype];
    seb ← base[SymDefs.setype];
    ctxb ← base[SymDefs.ctxtype];
    htb ← base[SymDefs.httype];
    END;

  -- tree printing

  PrintLiteral: PROCEDURE[t: literal TreeDefs.TreeLink] =
    BEGIN  OPEN LitDefs;
    desc: LitDescriptor;
    i: CARDINAL;
    v: WORD;
    WITH t.info SELECT FROM
      string =>
        BEGIN  PutChar['"];
        PutString[StringLiteralValue[index]];
        PutChar['"];
        IF index # MasterString[index] THEN PutChar['L];
        END;
      word =>
        BEGIN
        desc ← LitDescriptorValue[index];
        IF desc.length # 1 THEN PutChar['[];
        FOR i IN [0 .. desc.length)
          DO
          IF (v ← (ltb+desc.offset)[i]) < 1000
            THEN PutDecimal[v]
            ELSE PutOctal[v];
          IF i+1 # desc.length THEN PutChar[',];
          ENDLOOP;
        IF desc.length # 1 THEN PutChar[']];
        END;
      ENDCASE;
    END;

  PrintSubTree: PROCEDURE [t: TreeDefs.TreeLink, nBlanks: CARDINAL] =
    BEGIN  OPEN TreeDefs;
```

```
    Printer: TreeMap =
      BEGIN
      node: TreeIndex;
      Indent[nBlanks];
      WITH s: t SELECT FROM
        hash =>  PrintHti[s.index];
        symbol =>  PrintSei[s.index];
        literal =>  PrintLiteral[s];
        subtree =>
          BEGIN  node ← s.index;
          SELECT node FROM
            nullTreeIndex =>  PutString["<empty>"];
            ENDCASE =>
              BEGIN  OPEN (tb+node);
              PutNodeName[name];
              PutChar['[];  PrintIndex[node];  PutString["] "];
              IF info # 0
                THEN  BEGIN  PutString["  info="];  PrintIndex[info]  END;
              IF attr1 OR attr2
                THEN
                  BEGIN
                  IF info = 0 THEN PutChar[' ];
                  PutChar['(];
                  IF attr1 THEN PutChar['1];
                  IF attr2 THEN PutChar['2];
                  PutChar[')];
                  END;
              nBlanks ← nBlanks + 2;
              [] ← TreeDefs.UpdateTree[s, Printer];
              nBlanks ← nBlanks - 2;
              END;
          END;
        ENDCASE;
      RETURN [t]
      END;

    [] ← Printer[t];  RETURN
    END;


  PrintTree: PUBLIC PROCEDURE [t: TreeDefs.TreeLink] =
    BEGIN
    PrintSubTree[t, 0];  PutChar[CR];
    END;

  PrintBodies: PUBLIC PROCEDURE =
    BEGIN OPEN symbols;
    bti, prev: BTIndex;
    bti ← FIRST[BTIndex];
      DO
      PrintBody[bti];
      IF (bb+bti).firstSon # BTNull
        THEN bti ← (bb+bti).firstSon
        ELSE
          DO
          prev ← bti;  bti ← (bb+bti).link.index;
          IF bti = BTNull THEN GO TO Done;
          IF (bb+prev).link.which # parent THEN EXIT;
          ENDLOOP;
      REPEAT
        Done => NULL;
      ENDLOOP;
    PutChar[CR];
    END;

  PrintBody: PROCEDURE [bti: BTIndex] =
    BEGIN
    OPEN body: (symbols.bb+bti);
    PutChar[CR];  PutChar[CR];  PutString["Body["];
    PrintIndex[bti];  PutString["]: "];
    WITH b: body SELECT FROM
      Callable =>
        BEGIN
        PrintSei[b.id];
        PutString[", ep: "];  PutDecimal[b.entryIndex];
        WITH b SELECT FROM
```

```
        Inner =>
          BEGIN
          PutString[", frame address: "];  PutDecimal[frameOffset];
          END;
        ENDCASE;
      END;
    ENDCASE =>  PutString["(anon)"];
  PutChar[CR];
  PutString["  context: "];  PrintIndex[body.localCtx];
  PutString[", static level: "];  PutDecimal[body.level];
  WITH body.info SELECT FROM
    External =>
      BEGIN
      PutString[", Origin: "];  PrintIndex[origin];
      PutString[", Bytes: "];  PutDecimal[bytes];
      PutString[", startIndex: "];  PutDecimal[startIndex];
      PutString[", indexLength: "];  PutDecimal[indexLength];
      END;
    ENDCASE;
  PutChar[CR];  RETURN
  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 ← symbols.TypeLink[tSei]) = SENull
            DO
            WITH (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: CARDINAL] =
  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.lengthUsed THEN PutChar['*];
    IF t.machineDep THEN PutString[" (md)"];
    IF t.monitored THEN PutString[" (monitored)"];
    IF t.variant THEN PutString[" (variant)"];
    OutRecordCtx[", field ctx: ", LOOPHOLE[sei, recordCSEIndex]];
    WITH (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.dereferenced THEN PutChar['*];
    IF t.ordered THEN PutString[" (ordered)"];
    IF t.readonly THEN PutString[" (readonly)"];
    IF t.basing THEN PutString[" (base)"];
    PutString[", pointing to: "];
    PrintType[t.pointedtotype];
    PrintTypeInfo[t.pointedtotype, nBlanks+2];
    END;
  array =>
    BEGIN
    IF ~t.lengthUsed THEN PutChar['*];
    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;
  arraydesc =>
    BEGIN
    PutString[", described type: "]; PrintType[t.describedType];
    PrintTypeInfo[t.describedType, nBlanks+2];
    END;
  transfer =>
    BEGIN
    OutRecordCtx[", input ctx: ", t.inrecord];
    OutRecordCtx[", output ctx: ", t.outrecord];
    END;
  definition =>
    BEGIN
    PutString[", ctx: "];  PrintIndex[t.defCtx];
    PutString[", ngfi: "];  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[(seb+t.tagsei).idtype];
    PutString[", case ctx: "];  PrintIndex[t.casectx];
    IF t.controlled
      THEN PrintSE[t.tagsei, nBlanks+2];
    END;
  relative =>
    BEGIN
    PutString[", base type: "]; PrintType[t.baseType];
    PutString[", offset type: "]; PrintType[t.offsetType];
    PutString[", result type: "]; PrintType[t.resultType];
    PrintTypeInfo[t.baseType, nBlanks+2];
    PrintTypeInfo[t.offsetType, nBlanks+2];
    PrintTypeInfo[t.resultType, nBlanks+2];
    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;
            PrintTypeInfo[t.rangetype, nBlanks+2];
            END;
          long, real =>
            BEGIN
            PutString[" of: "];  PrintType[t.rangetype];
            PrintTypeInfo[t.rangetype, nBlanks+2];
            END;
          ENDCASE;
        END;
      ENDCASE;
  RETURN
  END;

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

PrintIndex: PROCEDURE [v: UNSPECIFIED] = LOOPHOLE[PutDecimal];

PrintSymbols: PROCEDURE =
  BEGIN
  ctx: CTXIndex;
  ctx ← FIRST[CTXIndex];
  UNTIL ctx = LOOPHOLE[symbols.stHandle.ctxBlock.size, 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;
  IF ~ symbols.stHandle.definitionsFile THEN PrintBodies[];
  RETURN
  END;

PrintContext: PROCEDURE [ctx: CTXIndex] =
  BEGIN
  sei, root: ISEIndex;
  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
    PrintSE[sei, 2];
```

```
    IF (sei ← symbols.NextSe[sei]) = root THEN EXIT;
    ENDLOOP;
  RETURN
  END;

PrintSE: PROCEDURE [sei: ISEIndex, nBlanks: CARDINAL] =
  BEGIN  OPEN (seb+sei);
  typeSei: SEIndex;
  addr: BitAddress;
  link: BcdDefs.ControlLink;
  Indent[nBlanks];
  PrintSei[sei];
  PutString[" ["];  PrintIndex[sei];  PutChar[']];
  IF public THEN PutString[" [public]"];
  IF mark3 THEN
    BEGIN
    PutString[", type = "];
    IF idtype = typeTYPE THEN
      BEGIN  typeSei ← idinfo;
      PutString["TYPE, equated to: "];
      PrintType[typeSei];
      IF symbols.TypeLink[sei] # SENull THEN
        BEGIN PutString[", tag code: "]; PutDecimal[idvalue] END;
      END
    ELSE
      BEGIN  typeSei ← idtype;  PrintType[typeSei];
      SELECT TRUE FROM
        constant => PutString[" [const]"];
        writeonce => PutString[" [init only]"];
        ENDCASE;
      IF ~mark4 THEN
        BEGIN PutString[", # refs: "]; PutDecimal[idinfo] END
      ELSE
        SELECT TRUE FROM
          constant =>
            IF ~ extended THEN
              BEGIN  PutString[", value: "];
              SELECT symbols.XferMode[typeSei] FROM
                procedure, program, signal, error =>
                  BEGIN  link ← idvalue;
                  PutChar['[];
                  PutDecimal[link.gfi];  PutChar[',];
                  PutDecimal[link.ep];  PutChar[',];
                  PutDecimal[LOOPHOLE[link.tag]];  PutChar[']];
                  END;
                ENDCASE =>
                  IF LOOPHOLE[idvalue, CARDINAL] < 1000
                    THEN  PutDecimal[idvalue]
                    ELSE  PutOctal[idvalue];
              END;
          (symbols.stHandle.definitionsFile AND (ctxb+ctxnum).ctxlevel = 1G) =>
            BEGIN PutString[", index: "]; PutDecimal[idvalue] END;
          ENDCASE =>
            BEGIN  addr ← idvalue;
            PutString[", address: "];
            PutDecimal[addr.wd];  PutChar[' ];
            PutChar['[];  PutDecimal[addr.bd];
            PutChar[':];  PutDecimal[idinfo];
            PutChar[']];
            IF linkSpace THEN PutChar['*];
            END;
          END;
        PrintTypeInfo[typeSei, nBlanks+2];
        IF extended THEN PrintSubTree[SymSegDefs.FindExtension[sei], nBlanks+4];
        END;
  RETURN
  END;

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

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

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

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

symbolsIn: BOOLEAN ← FALSE;

GetSymbolTable: PROCEDURE [seg: FileSegmentHandle] =
  BEGIN
  IF symbolsIn THEN TableDefs.DropNotify[ListSymbolsNotify];
  symbols←SymbolTableDefs.AcquireSymbolTable[
    SymbolTableDefs.TableForSegment[seg]];
  ListerDefs.SetRoutineSymbols[symbols];
  TableDefs.AddNotify[ListSymbolsNotify];
  TreeDefs.TreeInit[];
  LitDefs.LitTabInit[];
  symbolsIn ← TRUE;
  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
  [symbols: sseg] ← Load[bcdFile ! NoCode => RESUME;
    NoFGT => RESUME;
    NoSymbols, IncorrectVersion, MultipleModules =>
      BEGIN IODefs.WriteString["Bad format"]; GOTO giveup END;
    SegmentDefs.FileNameError =>
      BEGIN IODefs.WriteString["File not found"]; GOTO giveup END];
  GetSymbolTable[sseg];
  OpenOutput[root,".sl"];
  WriteFileID[];
  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];
    GetSymbolTable[sseg];
    OpenOutput[root,".sl"];
    PutString["Symbol Table in file: "];
    PutString[root];
    PutString[", base: "];
    PutDecimal[base];
    PutString[", pages: "];
    PutDecimal[pages];
    PutCR[];
    WriteVersions[@symbols.stHandle.version, @symbols.stHandle.creator];
```

```
    PrintSymbols[];
    CloseOutput[];
    EXITS
      NoFile => IODefs.WriteString["File not found"];
      BadSegment => IODefs.WriteString["Bad Segment"];
    END;
  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...
```