```
-- ListerRoutines.mesa; edited by Johnsson; July 20, 1978  12:10 PM

DIRECTORY
    AllocDefs: FROM "allocdefs",
    AltoDefs: FROM "altodefs",
    BcdDefs: FROM "bcddefs",
    BinaryDefs: FROM "binarydefs",
    ControlDefs: FROM "controldefs",
    ErrorTabDefs: FROM "errortabdefs",
    InlineDefs: FROM "inlinedefs",
    IODefs: FROM "iodefs",
    ListerDefs: FROM "listerdefs",
    Mopcodes: FROM "mopcodes",
    MiscDefs: FROM "miscdefs",
    OpTableDefs: FROM "optabledefs",
    OutputDefs: FROM "outputdefs",
    SegmentDefs: FROM "segmentdefs",
    StringDefs: FROM "stringdefs",
    SymbolTableDefs: FROM "symboltabledefs",
    SymDefs: FROM "symdefs",
    SymSegDefs: FROM "symsegdefs",
    SystemDefs: FROM "systemdefs",
    TableDefs: FROM "tabledefs",
    TimeDefs: FROM "timedefs",
    TreeDefs: FROM "treedefs";

DEFINITIONS FROM OutputDefs;

ListerRoutines: PROGRAM
  IMPORTS AllocDefs, BinaryDefs, MiscDefs, OutputDefs, SegmentDefs, SystemDefs
  EXPORTS ListerDefs, TableDefs SHARES SymbolTableDefs = PUBLIC
  BEGIN

  BYTE: TYPE = AltoDefs.BYTE;
  FileSegmentHandle: TYPE = SegmentDefs.FileSegmentHandle;
  FrameHandle: TYPE = ControlDefs.FrameHandle;
  NumberFormat: TYPE = IODefs.NumberFormat;
  opcode: TYPE = BYTE;
  PageCount: TYPE = AltoDefs.PageCount;
  WordPC: TYPE = ControlDefs.WordPC;



IncorrectVersion: SIGNAL = CODE;
NoFGT: SIGNAL = CODE;
NoCode: SIGNAL = CODE;
NoSymbols: SIGNAL = CODE;
MultipleModules: SIGNAL = CODE;
version, creator: BcdDefs.VersionStamp;
Dstar: BOOLEAN;
filename: STRING;

symbols: SymbolTableDefs.SymbolTableBase;
base: ARRAY [0..15] OF TableDefs.TableBase;

SetRoutineSymbols: PROCEDURE [s: SymbolTableDefs.SymbolTableBase] =
  BEGIN
  symbase: TableDefs.TableBase ← LOOPHOLE[s.stHandle];

  symbols ← s;
  BEGIN OPEN s.stHandle;
  base[SymDefs.httype] ← symbase + htBlock.offset;
  base[SymDefs.sstype] ← symbase + ssBlock.offset;
  base[SymDefs.setype] ← symbase + seBlock.offset;
  base[SymDefs.ctxtype] ← symbase + ctxBlock.offset;
  base[SymDefs.mdtype] ← symbase + mdBlock.offset;
  base[SymDefs.bodytype] ← symbase + bodyBlock.offset;
  base[SymSegDefs.exttype] ← symbase + extBlock.offset;
  base[SymSegDefs.treetype] ← symbase + treeBlock.offset;
  base[SymSegDefs.lttype] ← symbase + litBlock.offset;
  UpdateBases[];
  END;
  END;

  -- communication
```

```
  NotifyNode: TYPE = RECORD [
    notifier: TableDefs.TableNotifier,
    link: POINTER TO NotifyNode];

  notifyList: POINTER TO NotifyNode ← NIL;

  AddNotify: PUBLIC PROCEDURE [proc: TableDefs.TableNotifier] =
    BEGIN
    p: POINTER TO NotifyNode = SystemDefs.AllocateHeapNode[SIZE[NotifyNode]];
    p↑ ← [notifier:proc, link:notifyList];
    notifyList ← p;
    proc[DESCRIPTOR[base]];  RETURN
    END;

  DropNotify: PUBLIC PROCEDURE [proc: TableDefs.TableNotifier] =
    BEGIN
    p, q: POINTER TO NotifyNode;
    IF notifyList = NIL THEN RETURN;
    p ← notifyList;
    IF p.notifier = proc
      THEN notifyList ← p.link
      ELSE
        BEGIN
          DO
          q ← p;  p ← p.link;
          IF p = NIL THEN RETURN;
          IF p.notifier = proc THEN EXIT
          ENDLOOP;
        q.link ← p.link;
        END;
    SystemDefs.FreeHeapNode[p];  RETURN
    END;

  UpdateBases: PROCEDURE =
    BEGIN
    p: POINTER TO NotifyNode;
    FOR p ← notifyList, p.link UNTIL p = NIL DO  p.notifier[DESCRIPTOR[base]]  ENDLOOP;
    RETURN
    END;

-- to make TreeInit happy

GetChunk: PROCEDURE [size: CARDINAL] RETURNS [TableDefs.TableIndex] =
  BEGIN
  IF size # TreeDefs.TreeNodeSize THEN ERROR; -- called to reserve empty
  RETURN [LOOPHOLE[0]];
  END;


Load: PROCEDURE [name: STRING] RETURNS [code, symbols: FileSegmentHandle] =
  BEGIN OPEN SegmentDefs;
  bcdseg: FileSegmentHandle;
  bcd: POINTER TO BcdDefs.BCD;
  sgb: CARDINAL;
  pages: AltoDefs.PageCount;
  codefile: FileHandle;
  mh: BcdDefs.MTHandle;

  code ← symbols ← NIL;
  Dstar ← FALSE;
  filename ← name;
  codefile ← NewFile[name, Read, DefaultVersion];
  bcdseg ← NewFileSegment[codefile, 1, 1, Read];
  SwapIn[bcdseg];
  bcd ← FileSegmentAddress[bcdseg];
  IF (pages ← bcd.nPages) # 1 THEN BEGIN
    Unlock[bcdseg];
    MoveFileSegment[bcdseg, 1, pages];
    SwapIn[bcdseg];
    bcd ← FileSegmentAddress[bcdseg];
    END;
  BEGIN
  ENABLE
    UNWIND => BEGIN Unlock[bcdseg]; DeleteFileSegment[bcdseg] END;
  IF bcd.versionident # BcdDefs.VersionID THEN SIGNAL IncorrectVersion;
  version ← bcd.version;
```

```
    creator ← bcd.creator;
    mh ← LOOPHOLE[bcd, CARDINAL]+bcd.mtOffset+FIRST[BcdDefs.MTIndex];
    sgb ← LOOPHOLE[bcd, CARDINAL]+bcd.sgOffset;
    IF bcd.nModules # 1 THEN SIGNAL MultipleModules;
    IF bcd.definitions THEN SIGNAL NoCode
    ELSE
      BEGIN
      code ← NewFileSegment[codefile,
        (sgb+mh.code.sgi).base, (sgb+mh.code.sgi).pages, Read];
      code.class ← code;
      END;
    IF (sgb+mh.sseg).pages = 0 THEN SIGNAL NoSymbols
    ELSE
      BEGIN
      IF (sgb+mh.sseg).extraPages = 0 THEN SIGNAL NoFGT;
      symbols ← NewFileSegment[codefile,
        (sgb+mh.sseg).base, (sgb+mh.sseg).pages+(sgb+mh.sseg).extraPages, Read];
      END;
    END;
    IF code # NIL THEN
      BEGIN
      p: POINTER TO ControlDefs.CSegPrefix;
      SwapIn[code];
      p ← FileSegmentAddress[code];
      Dstar ← p.fill = 1;
      Unlock[code];
      END;
    Unlock[bcdseg];  DeleteFileSegment[bcdseg];
    RETURN
    END;

  WriteVersions: PROCEDURE [version, creator: POINTER TO BcdDefs.VersionStamp] =
    BEGIN OPEN OutputDefs;
    PutString[" created "L];
    PutTime[version.time];
    PutString["  on "L];
    PrintMachine[version↑];
    PutCR[];
    PutString["  creator "L];
    PutTime[creator.time];
    PutString["  on "L];
    PrintMachine[creator↑];
    PutCR[]; PutCR[];
    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!"L];
      RETURN
      END;

  WriteFileID: PROCEDURE =
    BEGIN
    PutString[filename];
    IF Dstar THEN PutString[" (/-A)"L];
    Dstar ← FALSE;
    WriteVersions[@version, @creator];
    RETURN
    END;

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

```
    END;

  PrintSei: PROCEDURE [sei: SymDefs.ISEIndex] =
    BEGIN
    PrintHti[IF sei=SymDefs.SENull THEN SymDefs.HTNull ELSE (symbols.seb+sei).htptr];
    RETURN
    END;

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

  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;

  -- csrP and desc.base are set by SwapInStringTab

  stringTableSeg: SegmentDefs.FileSegmentHandle;
  csrP: ErrorTabDefs.CSRptr;
  desc: StringDefs.SubStringDescriptor;
  ss: StringDefs.SubString = @desc;

  SwapInStringTab: PROCEDURE =
    BEGIN OPEN AllocDefs;
    info: AllocInfo = [0,hard,bottomup,initial,other,TRUE,FALSE];
    [stringTableSeg,] ← MiscDefs.DestroyFakeModule[LOOPHOLE[BinaryDefs.ErrorTab]];
    MakeSwappedIn[stringTableSeg, SegmentDefs.DefaultBase, info];
    csrP ← LOOPHOLE[SegmentDefs.FileSegmentAddress[stringTableSeg]];
    ss.base ← LOOPHOLE[csrP + csrP.relativebase, STRING];
    RETURN
    END;

  PutNodeName: PROCEDURE[n: TreeDefs.NodeName] =
    BEGIN
    ss.offset ← csrP.NodePrintName[n].offset;
    ss.length ← csrP.NodePrintName[n].length;
    PutSubString[ss];  RETURN
    END;

SwapInStringTab[];

END..
```