```
-- file ListInterface.mesa
-- last modified by Sandman, September 7, 1977  4:45 PM

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",
  SystemDefs: FROM "systemdefs";

DEFINITIONS FROM ListerDefs, OutputDefs, SymDefs;

ListInterface: PROGRAM IMPORTS ListerDefs, CommanderDefs, IODefs, OutputDefs, SegmentDefs, StringDefs,
**SymbolTableDefs, SystemDefs
  EXPORTS ListerDefs =
  BEGIN
  FileSegmentHandle: TYPE = SegmentDefs.FileSegmentHandle;
  FileHandle: TYPE = SegmentDefs.FileHandle;

  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;


  PrintInterface: PROCEDURE =
    BEGIN OPEN symbols;
    ss: StringDefs.SubStringDescriptor;
    sei: ISEIndex;

    FOR sei ← firstctxse[stHandle.outerCtx], nextse[sei] UNTIL sei = ISENull DO
      IF typeclass[symtype[sei]] = transfer THEN
        SELECT xfermode[symtype[sei]] FROM
          procedure, signal, error, program =>
            BEGIN
            PutNumber[LOOPHOLE[symaddress[sei]], NumberFormat[10, FALSE, FALSE, 4]];
            PutString[" -- "];
            printsei[sei];
            PutString[": "];
            PutModeName[xfermode[symtype[sei]]];
            PutChar[IODefs.CR];
            END;
        ENDCASE;
      ENDLOOP;
    RETURN
    END;

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

  ModePrintName: ARRAY TransferMode OF STRING = [
    "PROCEDURE", "PORT", "SIGNAL", "ERROR", "PROGRAM", "INLINE", "NONE"];

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

  PrintHeader: PROCEDURE [name: STRING, file:FileHandle] =
    BEGIN OPEN SegmentDefs;
    bcd: POINTER TO BcdDefs.BCD;
    bcdseg: FileSegmentHandle ← NewFileSegment[file,1,1,Read];
    octal3: IODefs.NumberFormat = NumberFormat[8, FALSE, FALSE, 3];
    SwapIn[bcdseg];
    bcd ← FileSegmentAddress[bcdseg];
    PutString[name];
    PutString[" compiled "];
    PutTime[bcd.version.time];
    PutString[" by "];
    PutNumber[bcd.version.net,octal3];
    PutChar['#];
    PutNumber[bcd.version.host,octal3];
    PutChar['#];
    IF bcd.version.zapped THEN PutString[" zapped!!!"];
    PutCR[];
    PutString["    Creator "];
    PutTime[bcd.creator.time];
    PutString[" "];
    PutNumber[bcd.creator.net,octal3];
    PutChar['#];
    PutNumber[bcd.creator.host,octal3];
    PutChar['#];
    IF bcd.creator.zapped THEN PutString[" zapped!!!"];
    PutCR[]; PutCR[];
    Unlock[bcdseg];
    DeleteFileSegment[bcdseg];
    RETURN
    END;

  GetSymbolTable: PROCEDURE [file:FileHandle] RETURNS [symseg:FileSegmentHandle] =
    BEGIN OPEN SegmentDefs;
    pages: PageCount;
    bcd: POINTER TO BcdDefs.BCD;
    mtb: CARDINAL;
    mti: BcdDefs.MTIndex = FIRST[BcdDefs.MTIndex];
    bcdseg: FileSegmentHandle ← NewFileSegment[file,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 BadVersionID;
      IF bcd.nModules # 1 THEN SIGNAL MultipleModules;
      IF ~bcd.definitions THEN SIGNAL NotDefinitionsModule;
    END;
    mtb ← LOOPHOLE[bcd,CARDINAL]+bcd.mtOffset;
    symseg ← FindSegment[bcdseg, (mtb+mti).sseg, FALSE];
    IF symseg # NIL THEN symseg.class ← symbols;
    Unlock[bcdseg];
    DeleteFileSegment[bcdseg];
    RETURN
    END;

  BadVersionID: SIGNAL = CODE;
  MultipleModules: SIGNAL = CODE;
  NotDefinitionsModule: SIGNAL = CODE;
```

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


Interface: PROCEDURE[root: STRING] =
  BEGIN OPEN StringDefs, SegmentDefs;
  i: CARDINAL;
  bcdFile: STRING ← [40];
  file: FileHandle;
  sseg: FileSegmentHandle;
  AppendString[bcdFile,root];
  FOR i IN [0..bcdFile.length) DO
    IF bcdFile[i] = '. THEN EXIT;
    REPEAT FINISHED => AppendString[bcdFile,".bcd"];
    ENDLOOP;
  BEGIN
  file ← NewFile[bcdFile, Read, OldFileOnly !
    FileNameError => GO TO BadName];
  sseg ← GetSymbolTable[file
    !BadVersionID, MultipleModules, NotDefinitionsModule => GOTO BadFormat];
  symbols←SymbolTableDefs.AcquireSymbolTable[SymbolTableDefs.TableForSegment[sseg]];
  OpenOutput[root,".il"];
  PrintHeader[bcdFile, file];
  PrintInterface[];
  SymbolTableDefs.ReleaseSymbolTable[symbols];
  SegmentDefs.DeleteFileSegment[sseg];
  CloseOutput[];
  EXITS
    BadFormat => IODefs.WriteString["Bad Format!"];
    BadName => IODefs.WriteString["File Not Found!"];
  END;
  END;


command: CommanderDefs.CommandBlockHandle;

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

END...
```