DIRECTORY

AltoDefs: FROM "altodefs",
AltoFileDefs: FROM "altofiledefs",
ControlDefs: FROM "controldefs",
ImageDefs: FROM "imagedefs",
ImageFileInfoDefs: FROM "imagefileinfodefs",
InlineDefs: FROM "inlinedefs",
IODefs: FROM "iodefs",
MiscDefs: FROM "miscdefs",
OutputDefs: FROM "outputdefs",
SegmentDefs: FROM "segmentdefs",
StreamDefs: FROM "streamdefs",
StringDefs: FROM "stringdefs",
SymbolTableDefs: FROM "symboltabledefs",
SymDefs: FROM "symdefs",
SystemDefs: FROM "systemdefs";

DEFINITIONS FROM AltoDefs, AltoFileDefs, SegmentDefs;

FindSigs: PROGRAM

IMPORTS OutputDefs, SegmentDefs, StreamDefs, StringDefs, SymbolTableDefs,
        SystemDefs, ImageFileInfoDefs, MiscDefs

According to the documentation, the code block appears to be
related to handling signals and symbols in a program, possibly
for debugging purposes.

The code includes definitions for signal items and
procedures for manipulating symbols and segment handles.

For a full understanding, the code would need to be read
in its entirety, as it appears to be part of a larger
program or system that deals with file definitions and
symbol handling.

The stage is set for further exploration and
implementation, with the potential for significant
technical insights and applications.
desc + (symbols.seb+sei).idvalue;
t + nsigs;
WHILE t > 0 DO
  IF sigdata[t-1].desc < desc THEN EXIT;
  sigdata[t] + sigdata[t-1];
  t ~ t-1;
ENDLOOP;
nsigs ~ nsigs+1;
sigdata[t]~ [SystemDefs.AllocateHeapString[tname.length], desc];
StringDefs.AppendString[sigdata[t].name, tname];
END;
ENDCASE;
ENDLOOP;

IF nsigs > 0 THEN
  BEGIN
    OPEN OutputDefs;
    PutCR[];
    PutNumber[gframe, [8,FALSE,TRUE,6]];
    PutString["B "];
    PutString[modname];
    PutCR[];
    FOR t IN [0..nsigs) DO
      PutString[" "];
      PutNumber[sigdata[t].desc+gfimask, [8,FALSE,TRUE,6]];
      PutString["B "];
      PutString[sigdata[t].name];
      PutCR[];
    ENDO;
    ReleaseSymbolTable[symbols];
    RETURN
  END;
END;

ListSignals: PROCEDURE =
BEGIN OPEN ImageFileInfoDefs;
MungeModule: PROCEDURE [f: GlobalFrameHandle] RETURNS [BOOLEAN] =
BEGIN
  BadName: PROCEDURE =
  BEGIN
    OPEN OutputDefs;
    PutString[" (problems encountered)"]L;
    END;
  BadFrame: PROCEDURE =
  BEGIN
    OPEN OutputDefs;
    PutOctal[f];
    PutString[" (problems encountered)"]L;
    END;
  seg: FileSegmentHandle;
  name. length ~ 0;
  BEGIN
    FrameToModuleName[f, name ! ANY => BEGIN BadFrame[]: GOTO ret END];
    seg + SymbolSegForFrame[f ! ANY => BEGIN BadName[]: GOTO ret END];
    PrintSignals[seg, name, LOOPHOLE[f], VirtualGlobalFrame[f].gfi !
      ANY => BEGIN BadName[]; CONTINUE END];
    EXITs ret => NULL;
    END;
    RETURN[FALSE]
  END;
  name: STRING ~ [40];
  [] + ImageFileInfoDefs.EnumerateGlobalFrames[MungeModule];
  OutputDefs.CloseOutput[];
END;

CheckForExtension: PROCEDURE [name, ext: STRING] =
BEGIN
  i: CARDINAL;
  FOR i IN [0..name.length) DO
    IF name[i] = '. THEN RETURN;
  ENDO;
  StringDefs.AppendString[name, ext];
  RETURN
END;

ProcessImage: PROCEDURE =
BEGIN
  infile: STRING ~ [40];
  root: STRING ~ [40];
  i: CARDINAL;
  GetToken(infile];
  IF infile.length = 0 THEN SIGNAL Done;
FindSigs.mesa

FOR i IN [0..infile.length) DO
  IF infile[i] = '.' THEN EXIT;
  StringDefs.AppendChar[root, infile[i]]
ENDLOOP;
ImageFileInfoDefs.SetImage(infile];
ImageFileInfoDefs.FindAllSymbols();
OutputDefs.OpenOutput[root, "signals", L];
WriteHerald[infile];
ListSignals[];
RETURN
END:

WriteHerald: PROCEDURE [name: STRING] =
BEGIN OPEN OutputDefs;
PutString[name];
PutString['--']L;
PutTime[ImageFileInfoDefs.Version[.].time];
PutCR[]; PutCR[]; PutCR[];
RETURN
END:

GetToken: PROCEDURE [token: STRING] =
BEGIN c: CHARACTER;
token.length = 0;
UNTIL constr.endof[constr] DO
  SELECT c + constr.get[constr] FROM
   IODefs.SP, IODefs.CR
=)
IF token.length # 0 THEN RETURN;
ENDCASE => StringDefs.AppendChar[token, c];
ENDLOOP;
RETURN
END;

GetCommandLineStream: PROCEDURE RETURNS [s: StreamDefs.StreamHandle] =
BEGIN OPEN StreamDefs;
cfa: POINTER TO AltoFileDefs.CFA = MiscDefs.CommandLineCFA[];
s = CreateByteStream[SegmentDefs.InsertFile[@cfa.fp, Read], Read];
JumpToFAs, @cfa.fa];
RETURN
END;

Done: SIGNAL = CODE;

-- Main Body
name: STRING + [80];
constr: StreamDefs.StreamHandle = GetCommandLineStream[];
DO
  ProcessImage[] Done => EXIT];
ENDLOOP;
ImageDefs.StopMesa[];
END...