```
-- StmtMap.Mesa
-- Edited by:
--            Sandman on April 15, 1978  3:44 PM
--            Barbara on July 13, 1978  10:38 AM

DIRECTORY
  ControlDefs: FROM "controldefs" USING [BytePC, GlobalFrameHandle],
  DebugBreakptDefs: FROM "debugbreakptdefs",
  DebugData: FROM "debugdata" USING [textwindow],
  DebuggerDefs: FROM "debuggerdefs" USING [MainBTI, PcToBTI],
  DebugSymbolDefs: FROM "debugsymboldefs" USING [
    DAcquireSymbolTable, DReleaseSymbolTable, SymbolsForGFrame],
  DebugUtilityDefs: FROM "debugutilitydefs" USING [CacheNewFile],
  ImageDefs: FROM "imagedefs" USING [AddCleanupProcedure, CleanupItem,
    CleanupMask, CleanupProcedure],
  IODefs: FROM "iodefs" USING [CR, WriteChar],
  SegmentDefs: FROM "segmentdefs" USING [
    FileHandle, FileNameError, FileSegmentHandle, Read],
  StreamDefs: FROM "streamdefs" USING [
    CreateByteStream, GetIndex, ModifyIndex, SetIndex, StreamError,
    StreamHandle, StreamIndex],
  SymbolTableDefs: FROM "symboltabledefs" USING [SymbolTableBase],
  SymDefs: FROM "symdefs" USING [BTIndex, BTNull, FGTEntry],
  SystemDefs: FROM "systemdefs" USING [AllocateHeapNode, FreeHeapNode],
  WindowDefs: FROM "windowdefs" USING [
    SetFileHandleForWindow, SetIndexForWindow];

StmtMap: PROGRAM
  IMPORTS DDptr: DebugData, DebuggerDefs, DebugSymbolDefs, DebugUtilityDefs,
    ImageDefs, IODefs, SegmentDefs, StreamDefs, SystemDefs, WindowDefs
  EXPORTS DebugBreakptDefs
  SHARES SegmentDefs =

BEGIN

BytePC: TYPE = ControlDefs.BytePC;
BTIndex: TYPE = SymDefs.BTIndex;
SymbolTableBase: TYPE = SymbolTableDefs.SymbolTableBase;
GlobalFrameHandle: TYPE = ControlDefs.GlobalFrameHandle;

-- variables describing the file

nullframe: GlobalFrameHandle = NIL;
textframe: GlobalFrameHandle ← nullframe;
symbase: SymbolTableBase;
sourceavailable: BOOLEAN ← FALSE;
sourceFGTavailable: BOOLEAN ← FALSE;

-- fine-grain table utilities

CodeMappingFailure: SIGNAL = CODE;
SourceToObjectMappingNotAvailable: SIGNAL = CODE;

FindCodeIndex: PROCEDURE [pc: BytePC] RETURNS [CARDINAL] =
  BEGIN OPEN symbase;
  -- finds i such that pc is in [fgt[i].cindex..fgt[i+1].cindex)
  bti: BTIndex ← DebuggerDefs.PcToBTI[symbase, pc];
  i, s, l: CARDINAL;
  IF ~sourceFGTavailable THEN ERROR SourceToObjectMappingNotAvailable;
  IF bti = SymDefs.BTNull THEN ERROR CodeMappingFailure;
  WITH (bb+bti).info SELECT FROM
    External =>
      BEGIN s ← startIndex; l ← indexLength; END;
    ENDCASE => ERROR;
  IF l = 1 THEN RETURN [s];
  FOR i IN (s..s+l) DO
    IF pc < fgTable[i].cindex THEN RETURN [i-1];
    ENDLOOP;
  RETURN[i]
  END;

FindFileIndex: PROCEDURE [index: CARDINAL] RETURNS [besti: CARDINAL] =
  BEGIN OPEN symbase;
  -- finds least i such that index >= fgt[i].findex
  i, delta, x: CARDINAL;
  IF ~sourceFGTavailable THEN ERROR SourceToObjectMappingNotAvailable;
```

```
  besti ← delta ← 0;
  FOR i IN [0..LENGTH[fgTable]) DO
    IF (x ← fgTable[i].findex) > delta THEN
      BEGIN
      delta ← x;
      besti ← i;
      END;
    ENDLOOP;
  FOR i IN [0..LENGTH[fgTable]) DO
    IF index >= (x ← fgTable[i].findex) AND (x ← index-x) < delta THEN
      BEGIN
      delta ← x;  besti ← i;
      IF delta = 0 THEN RETURN[besti];
      END;
    ENDLOOP;
  RETURN[besti]
  END;

EntryToBTI: PUBLIC PROCEDURE [sbase: SymbolTableBase, e: CARDINAL]
  RETURNS [BTIndex] =
  BEGIN OPEN sbase;
  bti, prev: BTIndex ← DebuggerDefs.MainBTI;
  DO
    WITH b:(bb+bti) SELECT FROM
      Callable => IF b.entryIndex = e THEN RETURN[bti];
      ENDCASE;
    IF (bb+bti).firstSon # SymDefs.BTNull THEN bti ← (bb+bti).firstSon
        ELSE DO
          prev ← bti; bti ← (bb+bti).link.index;
          IF bti = SymDefs.BTNull THEN GOTO Done;
          IF (bb+prev).link.which # parent THEN EXIT;
          ENDLOOP;
    REPEAT
          Done => NULL;
    ENDLOOP;
  RETURN[bti]
  END;

-- interface routines

BodyEntryPc: PUBLIC PROCEDURE [frame: GlobalFrameHandle, e: CARDINAL,
  noSym: BOOLEAN] RETURNS [pc: BytePC] =
  BEGIN OPEN DebugSymbolDefs;
  bti: BTIndex;
  symFSH: SegmentDefs.FileSegmentHandle;
  sbase: SymbolTableBase ← DAcquireSymbolTable[
    LOOPHOLE[symFSH ← LOOPHOLE[SymbolsForGFrame[frame]]]];
  bti ← EntryToBTI[sbase,e];
  BEGIN OPEN sbase;
  FGTAvailable[symFSH,sbase];
  WITH (bb+bti).info SELECT FROM
    External =>
      pc ← IF sourceFile = NIL OR indexLength <= 1 OR noSym THEN [origin]
        ELSE [fgTable[startIndex+1].cindex];
    ENDCASE => ERROR;
  DReleaseSymbolTable[sbase];
  END;
  RETURN
  END;

BodyExitPc: PUBLIC PROCEDURE [frame: GlobalFrameHandle, e: CARDINAL]
  RETURNS [pc: BytePC] =
  BEGIN OPEN DebugSymbolDefs;
  bti: BTIndex;
  sbase: SymbolTableBase ← DAcquireSymbolTable[SymbolsForGFrame[frame]];
  bti ← EntryToBTI[sbase,e];
  BEGIN OPEN sbase;
  WITH (bb+bti).info SELECT FROM
    External =>  pc ← [origin + bytes-1];
    ENDCASE => ERROR;
  DReleaseSymbolTable[sbase];
  END;
  RETURN
  END;

CodeToSourceIndex: PUBLIC PROCEDURE [frame: GlobalFrameHandle, pc: BytePC]
```

```
  RETURNS [index: CARDINAL] =
  BEGIN OPEN symbase;
  SetUpSourceFile[frame];
  index ← fgTable[FindCodeIndex[pc]].findex;
  DebugSymbolDefs.DReleaseSymbolTable[symbase];
  RETURN
  END;

StringToFGTEntry: PUBLIC PROCEDURE [
  frame: GlobalFrameHandle, entryindex: CARDINAL, s: STRING]
  RETURNS [fgte: SymDefs.FGTEntry] =
  BEGIN OPEN symbase;
  bti: BTIndex;
  SetUpSourceFile[frame];
  bti ← EntryToBTI[symbase, entryindex];
  WITH (bb+bti).info SELECT FROM
    External =>
      fgte ← fgTable[FindFileIndex[TextStringSearch[
        s, fgTable[FindCodeIndex[[origin]]].findex]]];
    ENDCASE => ERROR;
  DebugSymbolDefs.DReleaseSymbolTable[symbase];
  RETURN
  END;

PrintLocation: PUBLIC PROCEDURE [frame: GlobalFrameHandle, index: CARDINAL,
  scroll:  BOOLEAN] =
  BEGIN
  pos: StreamDefs.StreamIndex;
  SetUpSourceFile[frame];
  DebugSymbolDefs.DReleaseSymbolTable[symbase];
  pos ← NextTextLine[index, IODefs.WriteChar];
  IF scroll THEN  WindowDefs.SetIndexForWindow[DDptr.textwindow, pos];
  RETURN
  END;

-- source file manipulation

SourceFileMissing: PUBLIC SIGNAL [sourcename: STRING] = CODE;

SetUpSourceFile: PROCEDURE [frame: GlobalFrameHandle] =
  BEGIN OPEN DebugSymbolDefs;
  symFSH: SegmentDefs.FileSegmentHandle;
  file: SegmentDefs.FileHandle;
  IF frame # textframe THEN ReleaseSourceFile[];
  symbase ← DAcquireSymbolTable[
    LOOPHOLE[symFSH ← LOOPHOLE[SymbolsForGFrame[frame]]]];
  BEGIN OPEN symbase;
    ENABLE UNWIND => DReleaseSymbolTable[symbase];
    IF frame # textframe OR ~sourceavailable THEN
      BEGIN
      textframe ← frame;
      FGTAvailable[symFSH,symbase];
      IF sourceFile # NIL THEN
        BEGIN
        file ← OpenTextFile[sourceFile ! NoSource =>
          BEGIN sourceavailable ← FALSE; file ← NIL; CONTINUE END];
        IF sourceavailable THEN WindowDefs.SetFileHandleForWindow[
            DDptr.textwindow, file, sourceFile];
        END;
      END;
    IF ~sourceavailable THEN ERROR SourceFileMissing[
      IF sourceFile # NIL THEN sourceFile ELSE "-- Compressed Symbols --"L];
    END;
  RETURN
  END;

ReleaseSourceFile: PROCEDURE =
  BEGIN
  IF textframe # nullframe AND sourceavailable THEN CloseTextFile[];
  textframe ← nullframe;
  RETURN
  END;

CleanupSourceFile: ImageDefs.CleanupProcedure =
  BEGIN
  IF why = OutLd THEN ReleaseSourceFile[];
```

```
  RETURN
  END;

FGTAvailable: PROCEDURE [fsh: SegmentDefs.FileSegmentHandle,
  symbase: SymbolTableBase] =
  BEGIN OPEN symbase;
  IF fsh.pages # (stHandle.fgRelPgBase+stHandle.fgPgCount)
    THEN sourceFile ← NIL;
  IF sourceFile = NIL
    THEN sourceFGTavailable ← sourceavailable ← FALSE
  ELSE sourceavailable ← sourceFGTavailable ← TRUE;
  END;

-- text searching routines

window: StreamDefs.StreamHandle;

NoSource: ERROR = CODE;

OpenTextFile: PROCEDURE [name: STRING] RETURNS [SegmentDefs.FileHandle] =
  BEGIN OPEN SegmentDefs;
  file: FileHandle ← DebugUtilityDefs.CacheNewFile[name, Read
    ! FileNameError => ERROR NoSource];
  window ← StreamDefs.CreateByteStream[file, Read];
  RETURN[file];
  END;

CloseTextFile: PROCEDURE =
  BEGIN
  IF window = NIL THEN RETURN;
  window.destroy[window];
  window ← NIL;
  RETURN
  END;

NextTextLine: PROCEDURE [index: CARDINAL, outchar: PROCEDURE [CHARACTER]]
  RETURNS [StreamDefs.StreamIndex] =
  BEGIN OPEN StreamDefs;
  ControlZ: CHARACTER = 32C;
  linestart, pos: StreamIndex;
  c: CHARACTER;
  SetIndex[window,StreamIndex[0,index]];
  linestart ← pos ← GetIndex[window];
  linestart ← ModifyIndex[linestart, -1];
  SetIndex[window, linestart];
  c ← window.get[window];
  UNTIL c = IODefs.CR DO
    IF GetIndex[window] = StreamIndex[0,1] THEN
      BEGIN window.reset[window]; EXIT; END;
    linestart ← ModifyIndex[linestart, -1];
    SetIndex[window, linestart];
    c ← window.get[window];
    ENDLOOP;
  linestart ← GetIndex[window];
  UNTIL window.endof[window] DO
    IF GetIndex[window] = pos THEN
      BEGIN outchar['<]; outchar['>]; END;
    c ← window.get[window];
    SELECT c FROM
    ControlZ =>
        BEGIN outchar[IODefs.CR]; EXIT END;
    ENDCASE => BEGIN outchar[c];
      IF c = IODefs.CR THEN EXIT;
      END;
    ENDLOOP;
  RETURN[linestart]
  END;

StringTooLong: PUBLIC SIGNAL [STRING] = CODE;
StringMatchFailure: PUBLIC SIGNAL [STRING] = CODE;

TextStringSearch: PROCEDURE [s: STRING, index: CARDINAL] RETURNS [CARDINAL] =
  -- searches for s using the Knuth, Pratt, Morris algorithm.
  -- returns byte index of the beginning of s.
  BEGIN OPEN StreamDefs;
  i, j, intL: INTEGER;
```

```
  offset: CARDINAL ← 0;
  char: CHARACTER;
  l: CARDINAL ← s.length;
  ff: DESCRIPTOR FOR ARRAY OF INTEGER; -- failure function
  IF l = 0 THEN RETURN[index];  -- empty string
  IF l > 60 THEN ERROR StringTooLong[s]; -- too big
  ff ← DESCRIPTOR[SystemDefs.AllocateHeapNode[l],l];
  -- set up failure function
  intL ← l;  j ← 0; i ← ff[0] ← -1;
  WHILE j < intL-1 DO
    WHILE i >= 0 AND s[j] # s[i] DO i ← ff[i] ENDLOOP;
    i ← i+1; j ← j+1;
    ff[j] ← IF s[j] = s[i] THEN ff[i] ELSE i;
    ENDLOOP;
  SetIndex[window,[page:0,byte:index]];
  j ← 0;  -- j = pattern index
  DO ENABLE UNWIND => SystemDefs.FreeHeapNode[BASE[ff]];
    IF j >= intL THEN EXIT;
    char ← window.get[window ! StreamError => GOTO fail];
    offset ← offset+1;
    WHILE j >= 0 AND char # s[j] DO j ← ff[j] ENDLOOP;
    j ← j + 1;
    REPEAT
      fail => ERROR StringMatchFailure[s];
    ENDLOOP;
  SystemDefs.FreeHeapNode[BASE[ff]];
  RETURN[index+offset-l]
  END;

-- Main Body

cleanupSource: ImageDefs.CleanupItem ← ImageDefs.CleanupItem [
  link:, mask: ImageDefs.CleanupMask[OutLd], proc: CleanupSourceFile];

ImageDefs.AddCleanupProcedure[@cleanupSource];

END.
```