```
-- DebugSymbols.Mesa
-- Edited by:
--             Sandman April 24, 1978  4:34 PM
--             Barbara June 19, 1978  1:43 PM
--             Johnsson August 28, 1978  10:08 PM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [PageCount, PageNumber],
  ControlDefs: FROM "controldefs" USING [FrameHandle, GlobalFrameHandle],
  DebugContextDefs: FROM "debugcontextdefs" USING [
    InvalidGlobalFrame, SymbolSegForFrame],
  DebugData: FROM "debugdata" USING [level],
  DebuggerDefs: FROM "debuggerdefs" USING [ClobberedFrame],
  DebugSymbolDefs: FROM "debugsymboldefs",
  DebugUtilityDefs: FROM "debugutilitydefs" USING [
    CheckFrame, FindSymbolTable, InvalidateFileCache, MREAD,
    ValidGlobalFrame],
  SegmentDefs: FROM "segmentdefs" USING [
    DeleteFileSegment, FileError, FileHandle, FileSegmentHandle, InvalidFP,
    NewFileSegment, PageCount, PageNumber, Read, SwapError],
  StringDefs: FROM "stringdefs" USING [AppendString],
  SymbolTableDefs: FROM "symboltabledefs" USING [
    AcquireSymbolTable, IllegalSymbolBase, NoSymbolTable, ReleaseSymbolTable,
    SegmentForTable, SetSymbolCacheSize, SymbolCacheSize, SymbolTableBase,
    SymbolTableHandle, TableForSegment],
  SystemDefs: FROM "systemdefs" USING [AllocateHeapNode, FreeHeapNode];

DEFINITIONS FROM DebugUtilityDefs;

DebugSymbols: PROGRAM
  IMPORTS DDptr: DebugData, DebugContextDefs, DebugUtilityDefs, SegmentDefs,
    DebuggerDefs, StringDefs, SymbolTableDefs, SystemDefs
  EXPORTS DebugSymbolDefs =

BEGIN

FrameHandle: TYPE = ControlDefs.FrameHandle;
GlobalFrameHandle: TYPE = ControlDefs.GlobalFrameHandle;
FileSegmentHandle: TYPE = SegmentDefs.FileSegmentHandle;
FileHandle: TYPE = SegmentDefs.FileHandle;
SymbolTableHandle: TYPE = SymbolTableDefs.SymbolTableHandle;
SymbolTableBase: TYPE = SymbolTableDefs.SymbolTableBase;

DCacheList: POINTER TO DCacheItem ← NIL;

DCacheItem: TYPE = RECORD [
  next: POINTER TO DCacheItem,
  handle: SymbolTableHandle,
  stbase: SymbolTableBase,
  count: CARDINAL,
  level: INTEGER];

DAcquireSymbolTable: PUBLIC PROCEDURE [h: SymbolTableHandle] RETURNS [b: SymbolTableBase] =
  BEGIN OPEN SegmentDefs;
  dl: POINTER TO DCacheItem;

  FOR dl ← DCacheList, dl.next UNTIL dl = NIL DO
    IF h = dl.handle AND dl.level = DDptr.level THEN
      BEGIN dl.count ← dl.count + 1; RETURN[dl.stbase]; END;
    ENDLOOP;
  b ← SymbolTableDefs.AcquireSymbolTable[h !
    SwapError, FileError, InvalidFP =>
      ERROR SymbolTableDefs.NoSymbolTable[NIL]];
  dl ← SystemDefs.AllocateHeapNode[SIZE[DCacheItem]];
  dl↑ ← DCacheItem[
    next: DCacheList, level: DDptr.level, handle: h, stbase: b, count: 1];
  DCacheList ← dl;
  RETURN
  END;

DReleaseSymbolTable: PUBLIC PROCEDURE [b: SymbolTableBase] =
  BEGIN
  pdl, dl: POINTER TO DCacheItem;
  pdl ← NIL;
  dl ← DCacheList;
  FOR dl ← DCacheList, dl.next UNTIL dl = NIL DO
```

```
    IF dl.stbase = b AND dl.level = DDptr.level THEN
      BEGIN
      IF (dl.count ← dl.count - 1) = 0 THEN
        BEGIN
        IF pdl = NIL THEN DCacheList ← dl.next ELSE pdl.next ← dl.next;
        SymbolTableDefs.ReleaseSymbolTable[dl.stbase];
        SystemDefs.FreeHeapNode[dl];
        END;
      RETURN
      END
    ELSE pdl ← dl;
    ENDLOOP;
  RETURN
  END;

HandleForBase: PUBLIC PROCEDURE [b: SymbolTableBase] RETURNS [SymbolTableHandle] =
  BEGIN
  dl: POINTER TO DCacheItem;
  FOR dl ← DCacheList, dl.next UNTIL dl = NIL DO
    IF dl.stbase = b THEN RETURN [dl.handle];
    ENDLOOP;
  ERROR
  END;

CheckDCache: PUBLIC PROCEDURE =
  BEGIN
  dl: POINTER TO DCacheItem ← DCacheList;
  ndl: POINTER TO DCacheItem;
  FOR dl ← DCacheList, ndl UNTIL dl = NIL DO
    ndl ← dl.next;
    IF dl.level >= DDptr.level THEN
      BEGIN
      dl.count ← 1;  -- force entry to be released
      DReleaseSymbolTable[dl.stbase !
        SymbolTableDefs.IllegalSymbolBase => CONTINUE];
      END;
    ENDLOOP;
  RETURN
  END;

TableForString: PUBLIC PROCEDURE [name: STRING] RETURNS [SymbolTableHandle] =
  BEGIN OPEN SegmentDefs;
  i: CARDINAL;
  file: FileHandle;
  base: PageNumber;
  pages: PageCount;
  symseg: FileSegmentHandle;
  FOR i IN [0..name.length) DO
    IF name[i] = '. THEN EXIT;
    REPEAT FINISHED => StringDefs.AppendString[name, ".bcd"];
    ENDLOOP;
  [file, base, pages] ← FindSymbolTable[name];
  symseg ← NewSymbolSegment[file, base, pages];
  RETURN[LOOPHOLE[symseg]]
  END;

DSymbolItem: TYPE = RECORD [
  next: POINTER TO DSymbolItem,
  frame: GlobalFrameHandle,
  table: FileSegmentHandle];

DSymbols: POINTER TO DSymbolItem ← NIL;

SymbolsForFrame: PUBLIC PROCEDURE [frame: FrameHandle]
  RETURNS [SymbolTableHandle] =
  BEGIN
  IF ~DebugUtilityDefs.CheckFrame[frame] THEN
    ERROR DebuggerDefs.ClobberedFrame[frame];
  RETURN[SymbolsForGFrame[MREAD[@frame.accesslink]]]
  END;

SymbolsForGFrame: PUBLIC PROCEDURE [gframe: GlobalFrameHandle]
  RETURNS [SymbolTableHandle] =
  BEGIN OPEN SymbolTableDefs;
  seg: FileSegmentHandle;
  syms: POINTER TO DSymbolItem;
```

```
  IF ~DebugUtilityDefs.ValidGlobalFrame[gframe] THEN
    ERROR DebugContextDefs.InvalidGlobalFrame[gframe];
  FOR syms ← DSymbols, syms.next UNTIL syms = NIL DO
    IF gframe = syms.frame THEN RETURN[TableForSegment[syms.table]];
    ENDLOOP;
  seg ← DebugContextDefs.SymbolSegForFrame[gframe];
  syms ← SystemDefs.AllocateHeapNode[SIZE[DSymbolItem]];
  syms↑ ← DSymbolItem[next: DSymbols, frame: gframe, table: seg];
  DSymbols ← syms;
  RETURN[TableForSegment[syms.table]]
  END;

DCheckSymbolItems: PUBLIC PROCEDURE =
  BEGIN
  syms: POINTER TO DSymbolItem;
  FOR syms ← DSymbols, DSymbols UNTIL syms = NIL DO
    DSymbols ← syms.next;
    IF syms.table # NIL THEN
      SegmentDefs.DeleteFileSegment[syms.table ! ANY => CONTINUE];
    SystemDefs.FreeHeapNode[syms];
    ENDLOOP;
  END;

DCleanSymbolItems: PUBLIC PROCEDURE =
  BEGIN
  syms, prev, next: POINTER TO DSymbolItem;
  prev ← NIL;
  FOR syms ← DSymbols, next UNTIL syms = NIL DO
    next ← syms.next;
    IF syms.table = NIL THEN
      BEGIN
      IF prev = NIL THEN DSymbols ← next ELSE prev.next ← next;
      SystemDefs.FreeHeapNode[syms];
      END
    ELSE prev ← syms;
    ENDLOOP;
  END;

AttachSymbols: PUBLIC PROCEDURE [frame: GlobalFrameHandle, file: STRING] =
  BEGIN OPEN SymbolTableDefs;
  syms: POINTER TO DSymbolItem;
  table: SymbolTableHandle;
  table ← TableForString[file];
  FOR syms ← DSymbols, syms.next UNTIL syms = NIL DO
    IF frame = syms.frame THEN
      BEGIN
      syms.table ← SegmentForTable[table];
      RETURN
      END;
    ENDLOOP;
  syms ← SystemDefs.AllocateHeapNode[SIZE[DSymbolItem]];
  syms↑ ← DSymbolItem[next: DSymbols, frame: frame, table: SegmentForTable[table]];
  DSymbols ← syms;
  DebugUtilityDefs.InvalidateFileCache[];
  RETURN
  END;

-- Lists of user symbol table segments

SymbolListItem: TYPE = RECORD [
  link: POINTER TO SymbolListItem,
  segment: FileSegmentHandle];

UserSymbolList: POINTER TO SymbolListItem ← NIL;

AddToUserSymbolList: PROCEDURE [seg: FileSegmentHandle] =
  BEGIN
  item: POINTER TO SymbolListItem =
    SystemDefs.AllocateHeapNode[SIZE[SymbolListItem]];
  item.segment ← seg;
  item.link ← UserSymbolList;
  UserSymbolList ← item;
  END;

EnumerateUserSymbolList: PROCEDURE [
  proc: PROCEDURE [FileSegmentHandle] RETURNS [BOOLEAN]]
```

```
  RETURNS [FileSegmentHandle] =
  BEGIN
  item: POINTER TO SymbolListItem;
  FOR item ← UserSymbolList, item.link UNTIL item = NIL DO
    IF proc[item.segment] THEN RETURN[item.segment];
    ENDLOOP;
  RETURN[NIL];
  END;

NewSymbolSegment: PROCEDURE [
  file: FileHandle, base: AltoDefs.PageNumber, pages: AltoDefs.PageCount]
  RETURNS [s: FileSegmentHandle] =
  BEGIN
  FindSegment: PROCEDURE [test: FileSegmentHandle] RETURNS [BOOLEAN] =
    BEGIN
    RETURN [test.file = file AND test.base = base AND test.pages = pages]
    END;
  IF (s ← EnumerateUserSymbolList[FindSegment]) = NIL THEN
    BEGIN
    s ← SegmentDefs.NewFileSegment[file, base, pages, SegmentDefs.Read];
    s.class ← other;
    AddToUserSymbolList[s];
    END;
  RETURN
  END;

PurgeUserSymbols: PUBLIC PROCEDURE RETURNS [did: BOOLEAN] =
  BEGIN
  this, prev: POINTER TO SymbolListItem;
  cachesize: INTEGER ← SymbolTableDefs.SymbolCacheSize[];

  SymbolTableDefs.SetSymbolCacheSize[0];
  SymbolTableDefs.SetSymbolCacheSize[cachesize];
  did ← FALSE;
  prev ← NIL; this ← UserSymbolList;
  UNTIL this=NIL DO
    IF this.segment.swappedin THEN
      BEGIN
      prev ← this;
      this ← this.link;
      END
    ELSE
      BEGIN
      IF prev=NIL THEN UserSymbolList ← this.link
      ELSE prev.link ← this.link;
      SegmentDefs.DeleteFileSegment[this.segment
        ! SegmentDefs.InvalidFP => CONTINUE];
      did ← TRUE;
      SystemDefs.FreeHeapNode[this];
      IF prev=NIL THEN this ← UserSymbolList
      ELSE this ← prev.link;
      END;
    ENDLOOP;
  RETURN
  END;

END...
```