```
-- Debugger.Mesa
-- Edited by:
--          Sandman on April 24, 1978  3:27 PM
--          Barbara on August 3, 1978  11:06 AM

DIRECTORY
  ControlDefs: FROM "controldefs" USING [
    Frame, FrameHandle, GlobalFrameHandle, localbase, NullFrame,
    NullGlobalFrame],
  DebugContextDefs: FROM "debugContextdefs" USING [
    EnumerateConfiguration, InvalidGlobalFrame, ModuleNameToFrame],
  DebugData: FROM "debugdata" USING [caseignoring, gContext, lContext],
  DebuggerDefs: FROM "debuggerdefs" USING [
    CatchFrame, ClobberedFrame, EquivalentVersions, FrameRelBPC,
    GetValue, InitSOP, MainBodyFrame, MainBTI, NoPreviousFrame, PcToBTI,
    PreviousFrame, SA, SearchType, SOPointer, SymbolObject, SymFrameHandle,
    VersionStamp],
  DebugMiscDefs: FROM "debugmiscdefs" USING [ControlDEL, LookupFail],
  DebugSymbolDefs: FROM "debugsymboldefs" USING [
    DAcquireSymbolTable, DReleaseSymbolTable, SymbolsForFrame,
    SymbolsForGFrame, TableForString],
  DebugUtilityDefs: FROM "debugutilitydefs" USING [
    CheckFrame, MREAD, ValidGlobalFrame],
  StreamDefs: FROM "streamdefs" USING [ControlDELtyped],
  StringDefs: FROM "stringdefs" USING [
    AppendSubString, EqualSubStrings, EquivalentSubStrings, SubString,
    SubStringDescriptor],
  SymbolTableDefs: FROM "symboltabledefs" USING [
    NoSymbolTable, SymbolTableBase],
  SymDefs: FROM "symdefs" USING [
    BTIndex, BTNull, CBTIndex, CSEIndex, CTXIndex, CTXNull, CTXRecord,
    HTIndex, HTNull, includedCTXIndex, ISEIndex, ISENull, lG, MDIndex,
    MDRecord, SEIndex, SENull],
  SystemDefs: FROM "systemdefs" USING [AllocateHeapNode, FreeHeapNode];

Debugger: PROGRAM
IMPORTS DDptr: DebugData, DebugContextDefs, DebuggerDefs, DebugMiscDefs,
  DebugSymbolDefs, DebugUtilityDefs, StreamDefs, StringDefs, SymbolTableDefs,
  SystemDefs
EXPORTS DebuggerDefs =
BEGIN

FrameHandle: TYPE = ControlDefs.FrameHandle;
GlobalFrameHandle: TYPE = ControlDefs.GlobalFrameHandle;
SymbolTableBase: TYPE = SymbolTableDefs.SymbolTableBase;
ISEIndex: TYPE = SymDefs.ISEIndex;
SEIndex: TYPE = SymDefs.SEIndex;
BTIndex: TYPE = SymDefs.BTIndex;
SymbolObject: TYPE = DebuggerDefs.SymbolObject;
SOPointer: TYPE = DebuggerDefs.SOPointer;

Lookup: PUBLIC PROCEDURE [s: STRING, publiconly: BOOLEAN, sop: SOPointer,
  type: BOOLEAN, search: DebuggerDefs.SearchType]
  RETURNS [found:BOOLEAN] =
  BEGIN
  frame: FrameHandle;
  SearchFrame: PROCEDURE [frame: FrameHandle] RETURNS [BOOLEAN] =
    BEGIN
    RETURN[found ← SearchFrameForSym[frame, s, publiconly, sop, type]]
    END;
  SearchGlobalFrame: PROCEDURE [gframe: GlobalFrameHandle] RETURNS [BOOLEAN] =
    BEGIN
    RETURN[found ← IF gframe = ControlDefs.NullGlobalFrame THEN FALSE
      ELSE SearchGFrameForSym[gframe, s, publiconly, sop, type]];
    END;
  found ← FALSE;
  SELECT search FROM
    mod =>
      BEGIN OPEN DebugMiscDefs, DebuggerDefs;
      frame ← DDptr.lContext;
      IF frame # NIL THEN
        WHILE frame # ControlDefs.NullFrame AND ~SearchFrame[frame] DO
          IF StreamDefs.ControlDELtyped[] THEN SIGNAL ControlDEL;
          frame ← PreviousFrame[frame ! NoPreviousFrame => EXIT];
          ENDLOOP
      ELSE found ← SearchGlobalFrame[DDptr.gContext];
```

```
      END;
    config => DebugContextDefs.EnumerateConfiguration[SearchGlobalFrame];
    ENDCASE => ERROR;
  RETURN
  END;

LookupProc: PUBLIC PROCEDURE [proc: STRING]
  RETURNS [found: BOOLEAN, gframe: GlobalFrameHandle, e: CARDINAL] =
  BEGIN OPEN DebugSymbolDefs; -- looks up proc at current context
  pdesc: StringDefs.SubStringDescriptor ←
    [base: proc, offset: 0, length: proc.length];
  pss: StringDefs.SubString ← @pdesc;
  thlp, hlp: HashListPtr;
  bti, prev: BTIndex ← DebuggerDefs.MainBTI;
  mainhti: SymDefs.HTIndex;
  stbase: SymbolTableDefs.SymbolTableBase;
  BEGIN OPEN stbase;
  found ← FALSE;
  gframe ← DDptr.gContext;
  stbase ← DAcquireSymbolTable[SymbolsForGFrame[gframe
      !SymbolTableDefs.NoSymbolTable => GOTO nosym]
    !SymbolTableDefs.NoSymbolTable => GOTO nosym];
  thlp ← hlp ← DFindString[stbase, pss];
  IF hlp # NIL THEN
    BEGIN
    IF (bb+bti).firstSon = SymDefs.BTNull THEN RETURN;
    DO
      WITH b:(bb+bti) SELECT FROM
        Callable =>
          BEGIN
          mainhti ← ((b.id)+seb).htptr;
          thlp ← hlp;
          UNTIL thlp = NIL DO
            IF mainhti = thlp.hti THEN
              BEGIN
              IF bti # DebuggerDefs.MainBTI THEN
                BEGIN found ← TRUE; e ← b.entryIndex; END
              ELSE found ← FALSE;
              GOTO done;
              END
            ELSE thlp ← thlp.link;
            ENDLOOP;
          END;
        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;
    FreeHashList[hlp];
    END;
  DReleaseSymbolTable[stbase];
  EXITS
    nosym => NULL;
  END;
  RETURN
  END;

LookupProg: PUBLIC PROCEDURE [prog: STRING]
RETURNS [found: BOOLEAN, gframe: GlobalFrameHandle, stbase: SymbolTableBase] =
  BEGIN OPEN DebugSymbolDefs;
  gfdesc: StringDefs.SubStringDescriptor ←
    [base: prog, offset: 0, length: prog.length];
  gfss: StringDefs.SubString ← @gfdesc;
  thlp, hlp: HashListPtr;
  mainhti: SymDefs.HTIndex;

  BEGIN OPEN stbase;
  found ← FALSE;
  gframe ← DDptr.gContext;
  stbase ← DAcquireSymbolTable[SymbolsForGFrame[gframe !
      SymbolTableDefs.NoSymbolTable => GOTO nosym] !
```

```
    SymbolTableDefs.NoSymbolTable => GOTO nosym];
  thlp ← hlp ← DFindString[stbase, gfss];
  IF hlp # NIL THEN
    BEGIN
    mainhti ← (((bb+DebuggerDefs.MainBTI).id)+seb).htptr;
    UNTIL thlp = NIL DO
      IF mainhti = thlp.hti THEN
        BEGIN found ← TRUE; EXIT; END
      ELSE thlp ← thlp.link;
      ENDLOOP;
    FreeHashList[hlp];
    END;
  IF ~found THEN DReleaseSymbolTable[stbase];
  EXITS
    nosym => NULL;
  END;
  RETURN
  END;

LookupLocals: PUBLIC PROCEDURE [sop: SOPointer, ss: StringDefs.SubString]
  RETURNS [BOOLEAN] =
  BEGIN
  hlp: HashListPtr;
  cbti: SymDefs.CBTIndex;
  BEGIN OPEN sop.stbase;
  hlp ← DFindString[sop.stbase, ss];
  IF hlp # NIL THEN
    BEGIN
    cbti ← (seb+sop.sei).idinfo;
    IF SearchCtxForSym[(bb+cbti).localCtx, hlp, sop, FALSE, FALSE]
      THEN GOTO foundit;
    IF SearchIOCtx[(seb+sop.sei).idtype, hlp, sop, FALSE] THEN GOTO foundit;
    FreeHashList[hlp];
    END;
  DebugSymbolDefs.DReleaseSymbolTable[sop.stbase];
  EXITS
    foundit => BEGIN FreeHashList[hlp]; RETURN[TRUE] END;
  END;
  RETURN[FALSE]
  END;

ClobberedAccessLink: PUBLIC ERROR [f: FrameHandle] = CODE;

SearchFrameForSym: PUBLIC PROCEDURE [frame: FrameHandle, s: STRING,
  publiconly: BOOLEAN, sop: SOPointer, type: BOOLEAN]
  RETURNS [BOOLEAN] =
  BEGIN  -- searches all variables of procedure and program frame
  OPEN DebugSymbolDefs, DebuggerDefs, ControlDefs, SymDefs, DebugUtilityDefs, sop.stbase;
  desc: StringDefs.SubStringDescriptor;
  hlp: HashListPtr;
  bti: BTIndex;
  gframe: ControlDefs.GlobalFrameHandle;
  CatchHandle: TYPE = POINTER TO catch Frame;

  BEGIN
  IF ~CheckFrame[frame] THEN ERROR ClobberedFrame[frame];
  sop.stbase ← DAcquireSymbolTable[SymbolsForFrame[frame !
      SymbolTableDefs.NoSymbolTable => GOTO nosym] !
    SymbolTableDefs.NoSymbolTable => GOTO nosym];
  IF CatchFrame[frame] THEN frame ← LOOPHOLE[MREAD[@LOOPHOLE[frame,
    CatchHandle].staticlink] - localbase];
  sop.baddr.wd ← short[shortAddr:IF ~MainBodyFrame[
    SymFrameHandle[faddr: frame, stbase: sop.stbase]] THEN LOOPHOLE[frame, SA]
    ELSE LOOPHOLE[MREAD[@frame.accesslink], SA]];
  desc ← StringDefs.SubStringDescriptor[base: s, offset: 0, length: s.length];
  hlp ← DFindString[sop.stbase, @desc];
  IF hlp # NIL THEN
    BEGIN
    bti ← PcToBTI[sop.stbase, FrameRelBPC[frame]];
    DO
      IF SearchCtxForSym[(bb+bti).localCtx, hlp, sop, publiconly, type]
        THEN GOTO foundit;
      WITH b:(bb+bti) SELECT FROM
        Callable =>
          BEGIN
          IF ~type AND SearchIOCtx[b.ioType, hlp, sop, publiconly] THEN
```

```
            GOTO foundit;
          WITH b SELECT FROM
            Inner => WITH s:sop.baddr SELECT FROM
              short => sop.baddr.wd ←short[shortAddr:LOOPHOLE[MREAD[@LOOPHOLE[
              s.shortAddr, FrameHandle].staticlink] - localbase]];
              ENDCASE;
            ENDCASE => EXIT;
          END;
        ENDCASE;
      UNTIL (bb+bti).link.which = parent DO
        bti ← (bb+bti).link.index;
        ENDLOOP;
      bti ← (bb+bti).link.index;
      IF bti = BTNull THEN EXIT;
      ENDLOOP;
    DReleaseSymbolTable[sop.stbase];
    FreeHashList[hlp];
    END;
  IF ~ValidGlobalFrame[gframe ← MREAD[@frame.accesslink]]
    THEN ERROR ClobberedAccessLink[frame];
  IF SearchGFrameForSym[gframe, s, publiconly, sop, type] THEN RETURN[TRUE];
  EXITS
    nosym => RETURN[FALSE];
    foundit => BEGIN FreeHashList[hlp]; RETURN[TRUE] END;
  END;
  RETURN[FALSE]
  END;

SearchIOCtx: PROCEDURE [ioType: SymDefs.SEIndex, hlp: HashListPtr,
  sop: SOPointer, publiconly: BOOLEAN]
  RETURNS [BOOLEAN] =
  BEGIN OPEN sop.stbase;
  typein, typeout: SymDefs.SEIndex;
  [typein, typeout] ← TransferTypes[ioType];
  IF typeout # SymDefs.SENull AND SearchCtxForSym[
    FieldContext[sop.stbase, typeout], hlp, sop, publiconly, FALSE] THEN
    RETURN[TRUE];
  IF typein # SymDefs.SENull AND SearchCtxForSym[
    FieldContext[sop.stbase, typein], hlp, sop, publiconly, FALSE] THEN
    RETURN[TRUE];
  RETURN[FALSE];
  END;

SearchGFrameForSym: PUBLIC PROCEDURE [gframe: GlobalFrameHandle, s: STRING,
  publiconly: BOOLEAN, sop: SOPointer, type: BOOLEAN]
  RETURNS [BOOLEAN] =
  BEGIN OPEN DebugSymbolDefs;
  IF ~DebugUtilityDefs.ValidGlobalFrame[gframe]
    THEN ERROR DebugContextDefs.InvalidGlobalFrame[gframe];
  sop.stbase ← DAcquireSymbolTable[SymbolsForGFrame[gframe !
      SymbolTableDefs.NoSymbolTable => GOTO nosym] !
    SymbolTableDefs.NoSymbolTable => GOTO nosym];
  sop.baddr.wd ← short[shortAddr:LOOPHOLE[gframe, DebuggerDefs.SA]];
  RETURN[SearchForGSym[s, publiconly, sop, type]]
  EXITS
    nosym => RETURN[FALSE];
  END;

SearchForModuleSym: PUBLIC PROCEDURE [mod: STRING, s: STRING,
  publiconly: BOOLEAN, sop: SOPointer, type: BOOLEAN] RETURNS [BOOLEAN] =
  BEGIN OPEN DebugSymbolDefs;
  gframe: GlobalFrameHandle;
  gframe ← DebugContextDefs.ModuleNameToFrame[mod !
    DebugMiscDefs.LookupFail =>
      BEGIN gframe ← ControlDefs.NullGlobalFrame; CONTINUE END];
  IF gframe = ControlDefs.NullGlobalFrame THEN
    sop.stbase ← DAcquireSymbolTable[TableForString[mod !
        SymbolTableDefs.NoSymbolTable => GOTO nosym] !
      SymbolTableDefs.NoSymbolTable => GOTO nosym]
  ELSE IF ~DebugUtilityDefs.ValidGlobalFrame[gframe]
    THEN ERROR DebugContextDefs.InvalidGlobalFrame[gframe]
  ELSE sop.stbase ← DAcquireSymbolTable[SymbolsForGFrame[gframe !
      SymbolTableDefs.NoSymbolTable => GOTO nosym] !
    SymbolTableDefs.NoSymbolTable => GOTO nosym];
  sop.baddr.wd ← short[shortAddr:LOOPHOLE[gframe, DebuggerDefs.SA]];
  RETURN[SearchForGSym[s, publiconly, sop, type]]
```

```
  EXITS
    nosym => RETURN[FALSE];
  END;

SearchForGSym: PROCEDURE [sym: STRING, publiconly: BOOLEAN, sop: SOPointer,
  type: BOOLEAN]
  RETURNS [BOOLEAN] = -- searches all variables of program frame
  BEGIN OPEN DebuggerDefs,DebugSymbolDefs;
  desc: StringDefs.SubStringDescriptor;
  ss: StringDefs.SubString = @desc;
  hlp: HashListPtr;

  BEGIN OPEN sop.stbase;
  desc ← StringDefs.SubStringDescriptor[base: sym, offset: 0,
    length: sym.length];
  hlp ← DFindString[sop.stbase, ss];
  IF hlp # NIL THEN
    BEGIN
    IF SearchCtxForSym[stHandle.outerCtx, hlp, sop, publiconly, type]
      THEN GOTO foundit;
    IF SearchImportCtxForSym[hlp, publiconly, sop, type] THEN GOTO foundit;
    IF ~type AND
      SearchIOCtx[(seb+(bb+MainBTI).id).idtype, hlp, sop, publiconly] THEN
      GOTO foundit
    ELSE IF SearchIncludeCtxForSym[hlp, publiconly, sop] OR
      SearchFieldCtxForSym[hlp, publiconly, sop] THEN GOTO foundit;
    IF SearchCtxForSym[stHandle.directoryCtx, hlp, sop, publiconly, type]
      THEN GOTO foundit;
    FreeHashList[hlp];
    END;
  DReleaseSymbolTable[sop.stbase];
  EXITS
    foundit => BEGIN FreeHashList[hlp]; RETURN[TRUE] END;
  END;
  RETURN[FALSE]
  END;

SearchIncludeCtxForSym: PROCEDURE [hlp: HashListPtr, publiconly: BOOLEAN, sop: SOPointer] RETURNS [BOOL
**EAN]=
  BEGIN OPEN SymDefs, sop.stbase;
  mdLimit: MDIndex;
  mdi: MDIndex;
  ctx: includedCTXIndex;
  mdLimit ← LOOPHOLE[stHandle.mdBlock.size];
  FOR mdi ← FIRST[MDIndex], mdi + SIZE[MDRecord] UNTIL mdi = mdLimit DO
    FOR ctx ← (mdb+mdi).mdctx, (ctxb+ctx).ctxchain UNTIL ctx = CTXNull DO
      IF (ctxb+ctx).ctxlevel = 1G THEN EXIT;
      ENDLOOP;
    IF ctx # CTXNull AND SearchCtxForSym[ctx, hlp, sop, publiconly, TRUE]
      THEN RETURN[TRUE];
    ENDLOOP;
  RETURN[FALSE]
  END;

SearchFieldCtxForSym: PROCEDURE [hlp: HashListPtr, publiconly: BOOLEAN, sop: SOPointer] RETURNS [BOOLEA
**N]=
  BEGIN OPEN sop.stbase;
  sei: SymDefs.ISEIndex;
  FOR sei ← FirstCtxSe[stHandle.directoryCtx], NextSe[sei]
  UNTIL sei = SymDefs.ISENull DO
    WITH (seb+UnderType[(seb+sei).idtype]) SELECT FROM
      transfer => IF mode = program THEN
        BEGIN
        IF SearchCtxForSym[FieldContext[sop.stbase, inrecord], hlp, sop,
          publiconly, TRUE] THEN RETURN[TRUE];
        IF SearchCtxForSym[FieldContext[sop.stbase, outrecord], hlp, sop,
          publiconly, TRUE] THEN RETURN[TRUE];
        EXIT
        END;
      ENDCASE;
    ENDLOOP;
  RETURN[FALSE]
  END;

SearchImportCtxForSym: PROCEDURE [hlp: HashListPtr, publiconly: BOOLEAN, sop: SOPointer, type: BOOLEAN]
** RETURNS [BOOLEAN]=
```

```
  BEGIN OPEN sop.stbase;
  sei: SymDefs.ISEIndex;
  FOR sei ← FirstCtxSe[stHandle.importCtx], NextSe[sei]
  UNTIL sei = SymDefs.ISENull DO
    WITH (seb+UnderType[(seb+sei).idtype]) SELECT FROM
      definition => WITH (ctxb+defCtx) SELECT FROM
        imported => IF SearchCtxForSym[defCtx, hlp, sop, publiconly, type]
          THEN RETURN[TRUE];
        ENDCASE;
      ENDCASE;
    ENDLOOP;
  RETURN[FALSE]
  END;

SearchForBasicSym: PUBLIC PROCEDURE[s: STRING, sop: SOPointer]
  RETURNS [BOOLEAN] =
  BEGIN OPEN DebugSymbolDefs, SymDefs, sop.stbase;
  desc: StringDefs.SubStringDescriptor;
  ss: StringDefs.SubString = @desc;
  hlp: HashListPtr;
  DebuggerDefs.InitSOP[sop];
  BEGIN
  sop.stbase ← DAcquireSymbolTable[SymbolsForGFrame[DDptr.gContext !
      SymbolTableDefs.NoSymbolTable => GOTO nosym] !
    SymbolTableDefs.NoSymbolTable => GOTO nosym]
  EXITS
    nosym => RETURN[FALSE];
  END;
  sop.baddr.wd ← short[shortAddr:LOOPHOLE[DDptr.gContext, DebuggerDefs.SA]];
  desc ← StringDefs.SubStringDescriptor[base: s, offset: 0, length: s.length];
  hlp ← DFindString[sop.stbase, ss];
  IF hlp # NIL THEN
    IF SearchCtxForSym[FIRST[CTXIndex]+SIZE[nil CTXRecord], hlp, sop, FALSE, FALSE]
      THEN BEGIN FreeHashList[hlp]; RETURN[TRUE]; END
    ELSE FreeHashList[hlp];
  DReleaseSymbolTable[sop.stbase];
  RETURN[FALSE]
  END;

SymbolTableNotFound: PUBLIC SIGNAL = CODE;

QualifyRecord: PUBLIC PROCEDURE [sop: SOPointer, ss: StringDefs.SubString]
  RETURNS [found: BOOLEAN]=
  BEGIN OPEN sop.stbase;
  vso: SymbolObject;
  vsop: SOPointer ← @vso;
  sei, tag: SymDefs.ISEIndex ← SymDefs.ISENull;
  lb: INTEGER ← 0;
  c: SymDefs.CTXIndex ← FindTopContext[sop.stbase, sop.tsei];
  WITH t: (seb+UnderType[sop.tsei]) SELECT FROM
    record => WITH l: t SELECT FROM
      linked => IF l.linktype # SymDefs.SENull THEN
        WITH s:(seb+sop.tsei) SELECT FROM
          id =>
            BEGIN sei ← s.idinfo; tag ← (seb+sei).idvalue; END;
          ENDCASE;
      ENDCASE;
    ENDCASE;
  sei ← SymDefs.ISENull;
  DO
    [found, c] ← SearchCompleteContext[sop, ss, c, FALSE];
    IF found THEN RETURN[TRUE];
    FOR sei ← FirstCtxSe[c], NextSe[sei] UNTIL sei = SymDefs.SENull DO
      WITH (seb+UnderType[(seb+sei).idtype]) SELECT FROM
        union =>
          BEGIN
          --IF ~controlled THEN RETURN[FALSE]; check for COMPUTED
          IF tag = SymDefs.ISENull THEN
            BEGIN
            vso ← SymbolObject[sei: tagsei, tsei: (seb+tagsei).idtype,
              baddr: sop.baddr, stbase: sop.stbase, space: 0,
              there: sop.there];
            WITH (seb+UnderType[vsop.tsei]) SELECT FROM
              subrange => lb ← origin;
              ENDCASE;
            tag ← LOOPHOLE[DebuggerDefs.GetValue[vsop] + lb];
```

```
            END;
          c ← FindFieldCtx[sop.stbase, casectx, tag];
          EXIT;
          END;
        ENDCASE => NULL;
      REPEAT
        FINISHED => BEGIN DebugSymbolDefs.DReleaseSymbolTable[sop.stbase];
                    RETURN[FALSE]; END;
      ENDLOOP;
    IF c = SymDefs.CTXNull THEN EXIT;
    ENDLOOP;
  DebugSymbolDefs.DReleaseSymbolTable[sop.stbase];
  RETURN[FALSE]
  END;

FindFieldCtx: PROCEDURE [stbase: SymbolTableBase, c: SymDefs.CTXIndex,
  tag: SymDefs.ISEIndex]
  RETURNS [SymDefs.CTXIndex] =
  BEGIN OPEN stbase;
  sei: SymDefs.ISEIndex;
  FOR sei ← FirstCtxSe[c], NextSe[sei] UNTIL sei = SymDefs.SENull DO
    WITH (seb+UnderType[sei]) SELECT FROM
      record => IF (seb+sei).constant AND (seb+sei).idvalue = tag
        THEN RETURN[fieldctx];
      ENDCASE => RETURN[SymDefs.CTXNull];
    ENDLOOP;
  RETURN[SymDefs.CTXNull];
  END;

SearchCompleteContext: PROCEDURE [sop: SOPointer, ss: StringDefs.SubString,
  c: SymDefs.CTXIndex, type: BOOLEAN]
  RETURNS[BOOLEAN, SymDefs.CTXIndex] =
  BEGIN
  hlp: HashListPtr;
  DO
    hlp ← DFindString[sop.stbase, ss];
    IF hlp # NIL AND SearchCtxForSym[c, hlp, sop, FALSE, type]
      THEN RETURN[TRUE, c];
    WITH (sop.stbase.ctxb+c) SELECT FROM
      included => IF ~ctxcomplete THEN c ← FindNextCtx[c,sop] ELSE EXIT;
      ENDCASE => EXIT;
    IF c = SymDefs.CTXNull THEN SIGNAL SymbolTableNotFound;
    ENDLOOP;
  RETURN[FALSE, c]
  END;

FindTopContext: PROCEDURE [stbase: SymbolTableBase, sei: SymDefs.SEIndex]
  RETURNS [SymDefs.CTXIndex] =
  BEGIN
  WITH r: (stbase.seb+stbase.UnderType[sei]) SELECT FROM
    record =>
      WITH r SELECT FROM
        linked => RETURN[FindTopContext[stbase,linktype]];
        ENDCASE => RETURN[fieldctx];
    ENDCASE => ERROR;
  RETURN[SymDefs.CTXNull]
  END;

VariantRecord: PUBLIC PROCEDURE [sop: SOPointer, ss: StringDefs.SubString]
  RETURNS [found: BOOLEAN]=
  BEGIN OPEN sop.stbase;
  sei: SymDefs.ISEIndex;
  c: SymDefs.CTXIndex;

  WITH (seb+UnderType[sop.tsei]) SELECT FROM
    record => c ← fieldctx;
    ENDCASE => RETURN[FALSE];
  DO
    FOR sei ← FirstCtxSe[c], NextSe[sei] UNTIL sei = SymDefs.ISENull DO
      WITH (seb+UnderType[(seb+sei).idtype]) SELECT FROM
        union => BEGIN c ← casectx; GOTO found END;
        ENDCASE;
      REPEAT
        FINISHED => WITH (sop.stbase.ctxb+c) SELECT FROM
          included => c ← IF ~ctxcomplete THEN FindNextCtx[c,sop]
            ELSE SymDefs.CTXNull;
```

```
          ENDCASE => GOTO notfound;
      ENDLOOP;
    IF c = SymDefs.CTXNull THEN GOTO notfound;
    REPEAT
      found => NULL;
      notfound => SIGNAL SymbolTableNotFound;
    ENDLOOP;
  [found, ] ← SearchCompleteContext[sop, ss, c, TRUE];
  RETURN[found]
  END;

FindNextCtx: PROCEDURE [c: SymDefs.CTXIndex, sop: SOPointer]
  RETURNS [newctx: SymDefs.CTXIndex]=
  BEGIN OPEN DebuggerDefs, DebugSymbolDefs, StringDefs, sop.stbase;
  filename: STRING ← [40];
  desc: SubStringDescriptor;
  filess: SubString ← @desc;
  includedVersion: DebuggerDefs.VersionStamp;

  WITH (ctxb+c) SELECT FROM
    included =>
      BEGIN
      SubStringForHash[filess, (mdb+ctxmodule).mdhti];
      AppendSubString[filename, filess];
      newctx ← ctxmap;
      includedVersion ← (mdb+ctxmodule).mdStamp;
      DReleaseSymbolTable[sop.stbase];
      sop.stbase ← DAcquireSymbolTable[TableForString[filename !
          SymbolTableDefs.NoSymbolTable => GOTO continue] !
        SymbolTableDefs.NoSymbolTable => GOTO continue];
      IF ~EquivalentVersions[sop.stbase.stHandle.version,includedVersion]
        THEN BEGIN DReleaseSymbolTable[sop.stbase]; GOTO continue; END;
      RETURN
      EXITS
        continue => RETURN[SymDefs.CTXNull];
      END;
    ENDCASE => ERROR;
  END;

SearchCtxForSym: PROCEDURE [c: SymDefs.CTXIndex, ihlp: HashListPtr,
  sop: SOPointer, publiconly: BOOLEAN, type: BOOLEAN] RETURNS [BOOLEAN] =
  BEGIN OPEN sop.stbase;
  sei: ISEIndex;
  hlp: HashListPtr;
  testtype: PROCEDURE [sei: ISEIndex] RETURNS [BOOLEAN] =
    BEGIN OPEN sop.stbase;
    RETURN[TypeForm[(seb+sei).idtype] = mode]
    END;

  FOR sei ← FirstCtxSe[c], NextSe[sei] UNTIL sei = SymDefs.ISENull DO
    FOR hlp ← ihlp, hlp.link UNTIL hlp = NIL DO
      IF hlp.hti = (seb+sei).htptr AND (~publiconly OR (seb+sei).public)
      AND (~type OR (type AND testtype[sei])) THEN
            BEGIN
        sop.sei ← sei; sop.tsei ← (seb+sei).idtype;
        RETURN[TRUE]
            END;
      ENDLOOP;
    ENDLOOP;
  RETURN[FALSE]
  END;

HashListItem: TYPE = RECORD[
  link: HashListPtr,
  hti: SymDefs.HTIndex];

HashListPtr: TYPE = POINTER TO HashListItem;

FreeHashList: PROCEDURE[hlp: HashListPtr]=
  BEGIN
  nexthlp:  HashListPtr;
  UNTIL hlp = NIL DO
    nexthlp ← hlp.link;
    SystemDefs.FreeHeapNode[hlp];
    hlp ← nexthlp;
    ENDLOOP;
```

```
  RETURN
  END;

DFindString: PROCEDURE [stb: SymbolTableBase, s: StringDefs.SubString]
  RETURNS [hlp: HashListPtr]=
  BEGIN OPEN stb, SymDefs, StringDefs;
  desc: SubStringDescriptor;
  ss: SubString = @desc;
  found: BOOLEAN;
  thlp: HashListPtr;
  hti: HTIndex ← hashVec[HashValue[s]];

  hlp ← NIL;
  WHILE hti # HTNull DO
    SubStringForHash[ss, hti];
    found ← IF DDptr.caseignoring THEN EquivalentSubStrings[s,ss]
      ELSE EqualSubStrings[s,ss];
    IF found THEN
      BEGIN
      thlp ← SystemDefs.AllocateHeapNode[SIZE[HashListItem]];
      thlp↑ ← [link:  hlp, hti:  hti];
      hlp ← thlp;
      END;
    hti ← ht[hti].link;
    ENDLOOP;
   RETURN
   END;

FieldContext: PUBLIC PROCEDURE [stbase: SymbolTableBase, type: SymDefs.SEIndex]
  RETURNS [SymDefs.CTXIndex] =
  BEGIN OPEN SymDefs, stbase;
  sei: CSEIndex;
  IF type = SENull THEN RETURN [CTXNull];
  sei ← UnderType[type];
  RETURN [WITH (seb+sei) SELECT FROM record => fieldctx, ENDCASE => CTXNull]
  END;


END ...
```