```
-- file: DumpFrames.Mesa
-- Edited by:
--            Sandman on May 1, 1978  2:12 PM
--            Barbara on July 20, 1978  4:05 PM
--            Johnsson on August 29, 1978  2:00 PM

DIRECTORY
  ControlDefs: FROM "controldefs" USING [
    BytePC, ControlLink, Frame, FrameHandle, GlobalFrame, GlobalFrameHandle,
    localbase, NullFrame, NullGlobalFrame, WordPC],
  DebugBreakptDefs: FROM "debugbreakptdefs" USING [
    PrintLocation, SourceFileMissing],
  DebugContextDefs: FROM "debugcontextdefs" USING [
    FrameToModuleName, IncorrectVersion, ModuleNameToFrame],
  DebugData: FROM "debugdata" USING [gContext, lContext, sigGF, StatePtr],
  DebuggerDefs: FROM "debuggerdefs" USING [
    AmIaRecord, BodySei, DumpCtxFromState, DumpCtxList, FieldContext,
    FormatRecord, FrameRelBPC, FRPointer, fullbitaddress, MainBTI, PcToBTI,
    PcToCBTI, SA, SymFrameHandle, WriteBlanks, WriteBodyName, WriteFrameLocus,
    WriteSource],
  DebugMiscDefs: FROM "debugmiscdefs" USING [
    ControlDEL, CopyRead, DFreeString, DGetString, WriteEOL],
  DebugSymbolDefs: FROM "debugsymboldefs" USING [
    DAcquireSymbolTable, DReleaseSymbolTable, SymbolsForFrame,
    SymbolsForGFrame],
  DebugUtilityDefs: FROM "debugutilitydefs" USING [
    CheckFrame, LoadStateInvalid, MREAD],
  IODefs: FROM "iodefs" USING [
    CR, DEL, NUL, ReadChar, ReadDecimal, WriteChar, WriteDecimal, WriteLine,
    WriteOctal, WriteString],
  LoadStateDefs: FROM "loadstatedefs" USING [
    InputLoadState, ReleaseLoadState],
  StreamDefs: FROM "streamdefs" USING [ControlDELtyped],
  SymbolTableDefs: FROM "symboltabledefs" USING [
    NoSymbolTable, SymbolTableBase],
  SymDefs: FROM "symdefs" USING [
    BTIndex, BTNull, CBTIndex, CTXIndex, ISEIndex, ISENull, lG, SEIndex,
    SENull];

DEFINITIONS FROM DebugSymbolDefs, DebugUtilityDefs, DebuggerDefs;

DumpFrames: PROGRAM
IMPORTS DDptr: DebugData, DebugBreakptDefs, DebugContextDefs, DebuggerDefs,
  DebugMiscDefs, DebugSymbolDefs, DebugUtilityDefs, IODefs, LoadStateDefs,
  StreamDefs, SymbolTableDefs
EXPORTS DebuggerDefs =

BEGIN

CTXIndex: TYPE = SymDefs.CTXIndex;
FrameHandle: TYPE = ControlDefs.FrameHandle;
GlobalFrameHandle: TYPE = ControlDefs.GlobalFrameHandle;
ISEIndex: TYPE = SymDefs.ISEIndex;
ISENull: SymDefs.ISEIndex = SymDefs.ISENull;
BTIndex: TYPE = SymDefs.BTIndex;
SEIndex: TYPE = SymDefs.SEIndex;
SENull: SymDefs.SEIndex = SymDefs.SENull;
SymbolTableBase: TYPE = SymbolTableDefs.SymbolTableBase;

DisplayStack: PUBLIC PROCEDURE =
  BEGIN
  IF DDptr.lContext # NIL THEN LocalDump[DDptr.lContext]
  ELSE IF DDptr.gContext # NIL THEN GlobalDump[DDptr.gContext]
  ELSE IODefs.WriteString[" ! Current context invalid."L];
  RETURN
  END;

DisplayFrame: PUBLIC PROCEDURE [f: UNSPECIFIED] =
  BEGIN
  IF DebugUtilityDefs.CheckFrame[f] THEN LocalDump[f]
  ELSE GlobalDump[f];
  RETURN
  END;

LocalDump: PUBLIC PROCEDURE [frame: FrameHandle] =
  BEGIN  --dump local frames
```

```
  npf: STRING = "No previous frame!"L;
  DO
    ENABLE
      BEGIN
      QuitDump => EXIT;
      NoPreviousFrame =>
        BEGIN
        DebugMiscDefs.WriteEOL[];
        IODefs.WriteString[npf];
        EXIT;
        END
      END;
    FrameDump[frame ! NewFrame => BEGIN frame ← newframe; RETRY END];
    frame ← PreviousFrame[frame];
    ENDLOOP;
  DebugMiscDefs.WriteEOL[];
  RETURN
  END;

GlobalDump: PUBLIC PROCEDURE [gframe: GlobalFrameHandle] =
  BEGIN  --dump a global frame
  c: CHARACTER;
  f: SymFrameHandle ← FrameForMainBody[gframe];
  IF f.faddr # ControlDefs.NullFrame THEN
    BEGIN
    DReleaseSymbolTable[f.stbase];
    FrameDump[f.faddr ! QuitDump => CONTINUE]
    END
  ELSE
    BEGIN
    DebugMiscDefs.WriteEOL[];
    IF f.stbase = NIL THEN IODefs.WriteString["No symbols for "L];
    WriteModuleGFrame[gframe];
    IF f.stbase = NIL THEN RETURN;
    DO
      IODefs.WriteString["  >"L];
      c ← IODefs.ReadChar[];
      IODefs.WriteChar[c];
      SELECT c FROM
        'p, 'P => DumpGParams[gframe, f.stbase];
        'v, 'V => DumpGLocals[gframe, f.stbase];
        'r, 'R => DumpGRetVals[gframe, f.stbase];
        's, 'S => DumpGSource[gframe];
        'q, 'Q, IODefs.DEL => EXIT;
        ENDCASE => IODefs.WriteString["--Options are: p,q,r,s,v."L];
      ENDLOOP;
    DReleaseSymbolTable[f.stbase];
    END;
  RETURN
  END;

ModuleDump: PUBLIC PROCEDURE [m: STRING] =
  BEGIN --dump the global frame named by m
  f: GlobalFrameHandle ← DebugContextDefs.ModuleNameToFrame[m];
  IF f # ControlDefs.NullGlobalFrame THEN GlobalDump[f];
  RETURN
  END;

QuitDump: SIGNAL = CODE;

NextFrame:  SIGNAL = CODE;

DumpNext: PROCEDURE [ISEIndex, FRPointer, SymFrameHandle] =
  BEGIN
  SIGNAL NextFrame;
  RETURN
  END;

NewFrame:  SIGNAL [newframe: FrameHandle] = CODE;

DumpJump: PROCEDURE [bsei: ISEIndex, frp: FRPointer, f: SymFrameHandle] =
  BEGIN
  i, n: INTEGER;
  IODefs.WriteString[", n(10): "L];
  IF (n ← IODefs.ReadDecimal[]) <= 0 THEN RETURN;
  FOR i IN [0..n) DO
```

```
    f.faddr ← PreviousFrame[f.faddr ! NoPreviousFrame =>
      BEGIN OPEN IODefs;
      IF i < n-1 THEN
        BEGIN WriteChar['(]; WriteDecimal[i]; WriteChar[')]; END;
      EXIT;
      END];
    ENDLOOP;
  DReleaseSymbolTable[f.stbase];
  DebugMiscDefs.WriteEOL[];
  SIGNAL NewFrame[f.faddr];
  RETURN
  END;

FrameAtEntry: FrameTest =
  BEGIN OPEN f.stbase;
  pc: ControlDefs.BytePC ← FrameRelBPC[f.faddr];
  cbti: SymDefs.CBTIndex ← PcToCBTI[f.stbase, pc];
  WITH (bb+cbti).info SELECT FROM
    External => RETURN [pc = origin];
    ENDCASE => ERROR
  END;

FrameAtExit: FrameTest =
  BEGIN OPEN f.stbase;
  pc: ControlDefs.BytePC ← FrameRelBPC[f.faddr];
  cbti: SymDefs.CBTIndex ← PcToCBTI[f.stbase, pc];
  WITH (bb+cbti).info SELECT FROM
    External => RETURN [pc = (origin+bytes-1)];
    ENDCASE => ERROR
  END;

FrameTest: TYPE = PROCEDURE [f: SymFrameHandle] RETURNS [BOOLEAN];

DumpLocalContext: PROCEDURE [
  context: SEIndex, frp: FRPointer, f: SymFrameHandle, test: FrameTest] =
  BEGIN OPEN f.stbase;  --Display values of all parameters of body sei
  a: fullbitaddress ← fullbitaddress[bd: 0,
    wd: short[shortAddr:LOOPHOLE[f.faddr]]];
  IF context = SymDefs.SENull THEN RETURN;
  IF MainBodyFrame[f] THEN a.wd ← short[shortAddr:MREAD[@f.faddr.accesslink]];
  IF test[f] THEN
    DumpCtxFromState[context, frp, f.stbase, DDptr.StatePtr, FALSE]
  ELSE DumpCtxList[FieldContext[f.stbase, context], f.stbase, TRUE, a, frp
    ! AmIaRecord => RESUME[FALSE]];
  RETURN
  END;

DumpParameters: PUBLIC PROCEDURE [
  bsei: ISEIndex, frp: FRPointer, f: SymFrameHandle] =
  BEGIN OPEN f.stbase;
  DebugMiscDefs.WriteEOL[];
  DumpLocalContext[
    TransferTypes[(seb+bsei).idtype].typeIn, frp, f, FrameAtEntry];
  RETURN
  END;

DumpRetVals: PUBLIC PROCEDURE [
  bsei: ISEIndex, frp: FRPointer, f: SymFrameHandle] =
  BEGIN OPEN f.stbase; --Display values of all RETURN variables of body sei
  DebugMiscDefs.WriteEOL[];
  DumpLocalContext[
    TransferTypes[(seb+bsei).idtype].typeOut, frp, f, FrameAtExit];
  RETURN
  END;

DumpLocals: PUBLIC PROCEDURE [
  bsei: ISEIndex, frp: FRPointer, f: SymFrameHandle] =
  --Display values of all locals of procedure sei
  BEGIN OPEN f.stbase;
  a: short fullbitaddress ← fullbitaddress[bd: 0,
    wd: short[shortAddr:LOOPHOLE[f.faddr, DebuggerDefs.SA]]];
  typein, typeout: SymDefs.SEIndex;
  bti: BTIndex;
  DebugMiscDefs.WriteEOL[];
  [typein, typeout] ← TransferTypes[(seb+bsei).idtype];
  IF typein # SENull THEN DumpLocalContext[typein, frp, f, FrameAtEntry];
```

```
  IF typeout # SENull THEN DumpLocalContext[typeout, frp, f, FrameAtExit];
  IF (bti ← PcToBTI[f.stbase, FrameRelBPC[f.faddr]]) = SymDefs.BTNull
    THEN RETURN;
  DO
    WITH b:(bb+bti) SELECT FROM
      Callable => EXIT;
      ENDCASE;
    a.wd ← short[shortAddr:LOOPHOLE[
      IF MainBodyFrame[f] THEN MREAD[@f.faddr.accesslink]
      ELSE f.faddr, DebuggerDefs.SA]];
    DumpCtxList[(bb+bti).localCtx, f.stbase, TRUE, a, frp
      ! AmIaRecord => RESUME[FALSE]];
    UNTIL (bb+bti).link.which = parent DO
      bti ← (bb+bti).link.index;
      ENDLOOP;
    IF (bti ← (bb+bti).link.index) = SymDefs.BTNull THEN RETURN;
    ENDLOOP;
  a.wd ← short[shortAddr:LOOPHOLE[
    IF MainBodyFrame[f] THEN MREAD[@f.faddr.accesslink]
    ELSE f.faddr, DebuggerDefs.SA]];
  DumpCtxList[(bb+bti).localCtx, f.stbase, TRUE, a, frp
      ! AmIaRecord => RESUME[FALSE]];
  RETURN
  END;

DumpGlobalContext: PROCEDURE [context: SEIndex, frp: FRPointer,
  frame: GlobalFrameHandle, stbase: SymbolTableBase] =
  BEGIN OPEN stbase;  --Display values of all parameters of body sei
  a: fullbitaddress ← fullbitaddress[bd: 0,
    wd: short[shortAddr:LOOPHOLE[frame]]];
  IF context = SymDefs.SENull THEN RETURN;
  DebugMiscDefs.WriteEOL[];
  DumpCtxList[FieldContext[stbase, context], stbase, TRUE, a, frp
    ! AmIaRecord => RESUME[FALSE]];
  RETURN
  END;

DumpGLocals: PUBLIC PROCEDURE [
  gframe: GlobalFrameHandle, stbase: SymbolTableBase] =
  --Display values of all locals of program sei
  BEGIN OPEN IODefs, stbase;
  FR: FormatRecord ← FormatRecord[indentation: 2, symid: TRUE, firstsym: TRUE,
    symdelim: '=, startchar: NUL, termchar: NUL, intersym: CR];
  a: fullbitaddress ← fullbitaddress[bd: 0,
    wd: short[shortAddr:LOOPHOLE[gframe]]];
  typein, typeout: SymDefs.SEIndex;
  id: ISEIndex;
  IF (id ← (bb+MainBTI).id) = ISENull THEN RETURN;
  DebugMiscDefs.WriteEOL[];
  [typein, typeout] ← TransferTypes[(seb+id).idtype];
  DumpGlobalContext[typein, @FR, gframe, stbase];
  DumpGlobalContext[typeout, @FR, gframe, stbase];
  DumpCtxList[(bb+MainBTI).localCtx, stbase, TRUE, a, @FR
    ! AmIaRecord => RESUME[FALSE]];
  RETURN
  END;

DumpGParams: PROCEDURE [
  gframe: GlobalFrameHandle, stbase: SymbolTableBase] =
  --Display values of all locals of program sei
  BEGIN OPEN IODefs, stbase;
  FR: FormatRecord ← FormatRecord[indentation: 2, symid: TRUE, firstsym: TRUE,
    symdelim: '=, startchar: NUL, termchar: NUL, intersym: CR];
  id: ISEIndex;
  IF (id ← (bb+MainBTI).id) = ISENull THEN RETURN;
  DumpGlobalContext[
    TransferTypes[(seb+id).idtype].typeIn, @FR, gframe, stbase];
  RETURN
  END;

DumpGRetVals: PROCEDURE [
  gframe: GlobalFrameHandle, stbase: SymbolTableBase] =
  --Display values of all locals of program sei
  BEGIN OPEN IODefs, stbase;
  FR: FormatRecord ← FormatRecord[indentation: 2, symid: TRUE, firstsym: TRUE,
    symdelim: '=, startchar: NUL, termchar: NUL, intersym: CR];
```

```
  id: ISEIndex;
  IF (id ← (bb+MainBTI).id) = ISENull THEN RETURN;
  DumpGlobalContext[
    TransferTypes[(seb+id).idtype].typeOut, @FR, gframe, stbase];
  RETURN
  END;

DumpGSource: PUBLIC PROCEDURE [gframe: GlobalFrameHandle]=
  BEGIN
  na: STRING = " not available"L;
  DebugMiscDefs.WriteEOL[];
  IODefs.WriteString[" Source: "L];
  DebugBreakptDefs.PrintLocation[gframe, 0, TRUE
    ! DebugBreakptDefs.SourceFileMissing =>
      BEGIN
      IODefs.WriteString[sourcename];
      IODefs.WriteString[na];
      CONTINUE;
      END];
  RETURN
  END;

DumpSource: PUBLIC PROCEDURE [sei: ISEIndex, frp: FRPointer, f: SymFrameHandle]=
  BEGIN
  WriteSource[MREAD[@f.faddr.accesslink], FrameRelBPC[f.faddr], TRUE];
  RETURN
  END;

DumpChoice: PROCEDURE [sei: ISEIndex, frp: FRPointer, f: SymFrameHandle] =
  --Interrogate user and call chosen dump procedure to use for this frame only
  BEGIN
  which: PROCEDURE [ISEIndex, FRPointer, SymFrameHandle] ← choose[];
  which[sei, frp, f];
  RETURN
  END;

DumpCatchFrame: PUBLIC PROCEDURE [f: FrameHandle] =
  BEGIN
  sfh: SymFrameHandle;
  BEGIN
  sfh.stbase ← DAcquireSymbolTable[SymbolsForFrame[f
      ! SymbolTableDefs.NoSymbolTable => GOTO nosym]
    ! SymbolTableDefs.NoSymbolTable => GOTO nosym];
  sfh.faddr ← f;
  WriteBodyName[sfh, FALSE];
  IODefs.WriteString[", "L];
  DReleaseSymbolTable[sfh.stbase];
  EXITS
    nosym => NULL;
  END;
  IODefs.WriteString["CatchFrame:  "L]; IODefs.WriteOctal[f];
  DebugMiscDefs.WriteEOL[];
  RETURN
  END;

CatchFrame: PUBLIC PROCEDURE [f: FrameHandle] RETURNS [BOOLEAN] =
  BEGIN OPEN ControlDefs;
  next, L0: FrameHandle;
  SignalHandle: TYPE = POINTER TO signal Frame;
  CatchHandle: TYPE = POINTER TO catch Frame;
  nextbase: signal Frame;
  nextp: POINTER TO ARRAY[0..1) OF UNSPECIFIED ← LOOPHOLE[@nextbase];
  i: CARDINAL;
  BEGIN
  next ← PreviousFrame[f ! ANY => GOTO notcatch];
  IF MREAD[@next.accesslink] # DDptr.sigGF THEN GOTO notcatch;
  L0 ← MREAD[@LOOPHOLE[f, CatchHandle].staticlink];
  IF ~CheckFrame[next]
    OR MREAD[@LOOPHOLE[f, FrameHandle].accesslink]
      # MREAD[@LOOPHOLE[L0-localbase, FrameHandle].accesslink]
    THEN GOTO notcatch;
  FOR i IN [0..SIZE[signal Frame]) DO
    nextp[i] ← MREAD[next+i];
    ENDLOOP;
  IF ~nextbase.mark THEN GOTO notcatch;
  EXITS
```

```
    notcatch => RETURN[FALSE];
  END;
  RETURN[TRUE];
  END;

FrameDump: PROCEDURE [frame: FrameHandle] =
  BEGIN
  sei: ISEIndex;
  f: SymFrameHandle;
  FR: FormatRecord;

  BEGIN
  DebugMiscDefs.WriteEOL[];
  IF CatchFrame[frame]
    THEN BEGIN DumpCatchFrame[frame]; RETURN END;
  f.stbase ← DAcquireSymbolTable[
    SymbolsForFrame[frame !
      SymbolTableDefs.NoSymbolTable--[seg]-- => GOTO nosym;
      DebugContextDefs.IncorrectVersion => RESUME] !
    SymbolTableDefs.NoSymbolTable--[seg]-- => GOTO nosym];
  f.faddr ← frame;
  WriteFrameLocus[f, FALSE];
  sei ← BodySei[f];
  IF sei # ISENull THEN DO
    FR ← FormatRecord[indentation:2, symid:TRUE, firstsym:TRUE, symdelim: '=,
      startchar: IODefs.NUL, termchar: IODefs.NUL, intersym: IODefs.CR];
    WriteBlanks[3];
    DumpChoice[sei, @FR, f
      !QuitDump => DReleaseSymbolTable[f.stbase];
      AmIaRecord => RESUME[FALSE];
      NextFrame => EXIT];
    DebugMiscDefs.WriteEOL[];
    ENDLOOP;
  DReleaseSymbolTable[f.stbase];
  EXITS
    nosym => DumpLimitedChoice[frame];
  END;
  RETURN
  END;

DumpLimitedChoice: PROCEDURE [frame: FrameHandle] =
  BEGIN OPEN IODefs;
  c: CHARACTER;
  i, n: CARDINAL;
  DumpNoSymbols[frame];
  DO
    WriteString[" >"L];
    c ← ReadChar[];
    WriteChar[c];
    SELECT c FROM
      'j, 'J =>
        BEGIN
        IODefs.WriteString[", n(10): "L];
        IF (n ← ReadDecimal[]) <= 0 THEN EXIT;
        FOR i IN [0..n) DO
          frame ← PreviousFrame[frame ! NoPreviousFrame =>
            BEGIN OPEN IODefs;
            IF i < n-1 THEN
              BEGIN WriteChar['(]; WriteDecimal[i]; WriteChar[')]; END;
            EXIT;
            END];
          ENDLOOP;
        DebugMiscDefs.WriteEOL[];
        SIGNAL NewFrame[frame];
        END;
      'n, 'N => BEGIN DebugMiscDefs.WriteEOL[]; RETURN; END;
      'q, 'Q, DEL => SIGNAL QuitDump;
      ENDCASE => IODefs.WriteString["--Options are: j,n,q"L];
    ENDLOOP;
  RETURN
  END;

DumpNoSymbols: PUBLIC PROCEDURE [frame: FrameHandle] =
  BEGIN OPEN IODefs;
  gframe: ControlDefs.GlobalFrameHandle ← MREAD[@frame.accesslink];
  WriteString["No symbols for "L];
```

```
  WriteModuleGFrame[gframe];
  WriteString[", L: "L];
  WriteOctal[frame];
  WriteString[", PC: "L];
  WriteOctal[ABS[MREAD[@frame.pc]]];
  WriteChar[',];
  WriteChar[IF LOOPHOLE[MREAD[@frame.pc], ControlDefs.WordPC] <0 THEN 'O ELSE 'E];
  RETURN
  END;

WriteModuleGFrame: PROCEDURE [gframe: GlobalFrameHandle] =
  BEGIN OPEN IODefs;
  module: STRING ← DebugMiscDefs.DGetString[40];
  BEGIN ENABLE DebugUtilityDefs.LoadStateInvalid => GOTO end;
  [] ← LoadStateDefs.InputLoadState[];
  DebugContextDefs.FrameToModuleName[gframe, module];
  WriteString[module];
  DebugMiscDefs.DFreeString[module];
  LoadStateDefs.ReleaseLoadState[];
  WriteString[", "L];
  EXITS
   end => NULL;
  END;
  WriteString["G: "L];
  WriteOctal[gframe];
  RETURN
  END;

choose: PROCEDURE RETURNS [p: PROCEDURE [ISEIndex, FRPointer, SymFrameHandle]] =
  BEGIN OPEN IODefs;  -- read a character to select a dump procedure
  c: CHARACTER;
  DO
    WriteString["  >"L];
    c ← ReadChar[];
    WriteChar[c];
    SELECT c FROM
      'j, 'J => BEGIN p ← DumpJump; EXIT END;
      'n, 'N => BEGIN p ← DumpNext; EXIT END;
      'p, 'P => BEGIN p ← DumpParameters; EXIT END;
      'v, 'V => BEGIN p ← DumpLocals; EXIT END;
      'r, 'R => BEGIN p ← DumpRetVals; EXIT END;
      's, 'S => BEGIN p ← DumpSource; EXIT END;
      'q, 'Q, DEL => SIGNAL QuitDump;
      ENDCASE => ListOptions[FALSE];
    ENDLOOP;
  RETURN
  END;

ListOptions: PUBLIC PROCEDURE [trace: BOOLEAN]=
  BEGIN
  IODefs.WriteString["--Options are:  p,v,r,s,q"L];
  IF trace THEN IODefs.WriteLine[",b--"L]
  ELSE IODefs.WriteLine[",j,n--"L];
  END;

ClobberedFrame: PUBLIC ERROR [f: FrameHandle] = CODE;
NoPreviousFrame: PUBLIC ERROR [f: FrameHandle] = CODE;

PreviousFrame: PUBLIC PROCEDURE [f: FrameHandle] RETURNS [FrameHandle] =
  BEGIN OPEN ControlDefs;
  link: ControlLink ← MREAD[@f.returnlink];
  IF StreamDefs.ControlDELtyped[] THEN SIGNAL DebugMiscDefs.ControlDEL;
  THROUGH [0..100] DO
    SELECT link.tag FROM
      frame =>
        IF link.frame = NullFrame THEN GOTO noPrev
        ELSE IF ~CheckFrame[link.frame] THEN GOTO clobbered
        ELSE RETURN[link.frame];
      procedure => GOTO noPrev;
      indirect => link ← MREAD[link];
      ENDCASE => GOTO clobbered;
    REPEAT
      noPrev => ERROR NoPreviousFrame[f];
      clobbered => ERROR ClobberedFrame[f];
      FINISHED => ERROR ClobberedFrame[f];
    ENDLOOP
```

```
  END;

FrameForMainBody: PROCEDURE [gframe: GlobalFrameHandle]
  RETURNS [f:SymFrameHandle] =
  BEGIN OPEN DebugUtilityDefs, ControlDefs;
  f ← SymFrameHandle[faddr:NullFrame, stbase:NIL];
  BEGIN OPEN f.stbase;
  f.stbase ← DAcquireSymbolTable[SymbolsForGFrame[gframe
      !SymbolTableDefs.NoSymbolTable--[seg]-- => GOTO nullframe]
    !SymbolTableDefs.NoSymbolTable--[seg]-- => GOTO nullframe];
  IF ~ReadGlobalStarted[gframe] THEN RETURN;
  IF (bb+MainBTI).stopping THEN
    BEGIN f.faddr ← MREAD[@gframe.global[0]]; RETURN END;
  EXITS
    nullframe => RETURN;
  END;
  RETURN
  END;

ReadGlobalStarted: PROCEDURE [gframe: GlobalFrameHandle]
  RETURNS [BOOLEAN] =
  BEGIN
  localFrame: ControlDefs.GlobalFrame;
  DebugMiscDefs.CopyRead[from: gframe, to: @localFrame,
    nwords: SIZE[ControlDefs.GlobalFrame]];
  RETURN[localFrame.started]
  END;

MainBodyFrame: PUBLIC PROCEDURE [f: SymFrameHandle] RETURNS [BOOLEAN] =
  BEGIN OPEN f.stbase;
  cbti: SymDefs.CBTIndex ← PcToCBTI[f.stbase, FrameRelBPC[f.faddr]];
  RETURN[((bb+cbti).localCtx+ctxb).ctxlevel = SymDefs.1G]
  END;


END ...
```