```
-- DebugContext.mesa
-- Edited by:
--              Johnsson on August 30, 1978  10:55 AM
--              Sandman on May 24, 1978  10:11 AM
--              Barbara on June 23, 1978  2:11 PM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [PageSize],
  BcdDefs: FROM "bcddefs" USING [
    CTIndex, CTNull, MTHandle, MTIndex, NameString],
  CommandDefs: FROM "commanddefs" USING [WriteSignalString],
  ControlDefs: FROM "controldefs" USING [
    FrameCodeBase, FrameHandle, GFT, GFTIndex, GlobalFrameHandle,
    GlobalFrame, NullGlobalFrame],
  DebugContextDefs: FROM "debugContextdefs" USING [
    CleanupControlDEL, DAcquireBcd, DeletedFrame, DReleaseBcd, InitBCD,
    PrintName, SameConfig],
  DebugData: FROM "debugdata" USING [
    bcdseg, caseignoring, config, cti, gContext, initBCD, lContext, pContext,
    ssb],
  DebugMiscDefs: FROM "debugmiscdefs" USING [
    ControlDEL, CopyRead, DebugAbort, DFreeString, DGetString, LookupFail,
    WriteEOL],
  DebugSymbolDefs: FROM "debugsymboldefs" USING [DCleanSymbolItems],
  DebugUsefulDefs: FROM "debugusefuldefs",
  DebugUtilityDefs: FROM "debugutilitydefs" USING [
    CacheNewFile, CheckFrame, InvalidateFileCache, LoadStateInvalid, MREAD,
    ReadGlobalGFI, ReverseEnumerateGlobalFrames, ValidGlobalFrame],
  ImageDefs: FROM "imagedefs" USING [ImageHeader, VersionID],
  IODefs: FROM "iodefs" USING [
    CR, WriteChar, WriteLine, WriteOctal, WriteString],
  LoaderBcdUtilDefs: FROM "loaderbcdutildefs" USING [
    BcdBase, EnumerateModuleTable, FindName, ReleaseBcdSeg, SetUpBcd],
  LoadStateDefs: FROM "loadstatedefs" USING [
    BcdSegFromLoadState, ConfigIndex, ConfigNull, GetLoadState, GFTIndex,
    InputLoadState, MapConfigToReal, MapRealToConfig, ReleaseLoadState,
    SetLoadState],
  ProcessDefs: FROM "processdefs" USING [ProcessHandle],
  SegmentDefs: FROM "segmentdefs" USING [
    DefaultAccess, DeleteFileSegment, FileHandle, FileNameError,
    FileSegmentAddress, FileSegmentHandle, FileSegmentObject,
    InvalidFP, MoveFileSegment, NewFileSegment, Read, SwapIn, Unlock],
  StreamDefs: FROM "streamdefs" USING [ControlDELtyped],
  StringDefs: FROM "stringdefs" USING [
    AppendString, AppendSubString, EqualSubStrings, EquivalentSubStrings,
    SubString, SubStringDescriptor],
  SystemDefs: FROM "systemdefs" USING [AllocateHeapNode, FreeHeapNode];

DEFINITIONS FROM BcdDefs, DebugContextDefs, LoaderBcdUtilDefs, LoadStateDefs;

DebugContext: PROGRAM
IMPORTS DDptr: DebugData, CommandDefs, DebugContextDefs, DebugMiscDefs,
  DebugSymbolDefs, DebugUtilityDefs, IODefs, LoaderBcdUtilDefs, LoadStateDefs,
  SegmentDefs, StreamDefs, StringDefs, SystemDefs
EXPORTS DebugContextDefs, DebugUsefulDefs, DebugUtilityDefs
SHARES ImageDefs, ProcessDefs =

BEGIN

FileSegmentHandle: TYPE = SegmentDefs.FileSegmentHandle;
GlobalFrameHandle: TYPE = ControlDefs.GlobalFrameHandle;

SetOctalContext: PUBLIC PROCEDURE [f: UNSPECIFIED] =
  BEGIN
  IF DebugUtilityDefs.CheckFrame[f] THEN WriteLocalContext[f]
  ELSE WriteGlobalContext[f];
  RETURN
  END;

SetModuleContext: PUBLIC PROCEDURE [s: STRING] =
  BEGIN
  f: GlobalFrameHandle;
  f ← ModuleNameToFrame[s ! MultipleInstances => RESUME];
  IF f = ControlDefs.NullGlobalFrame THEN RETURN;
  DDptr.gContext ← f;
  DDptr.lContext ← NIL;
  DDptr.pContext ← NIL;
  RETURN
  END;

ModuleNameToFrame: PUBLIC PROCEDURE [s: STRING] RETURNS [f: GlobalFrameHandle] =
  BEGIN OPEN ControlDefs;
  FrameItem: TYPE = RECORD [next: POINTER TO FrameItem,
    faddr: GlobalFrameHandle];
  Fcount: INTEGER ← 0;
  Flist: POINTER TO FrameItem ← NIL;
  moddesc: StringDefs.SubStringDescriptor;
  modss: StringDefs.SubString ← @moddesc;
  bcd: BcdBase;

  FreeFrameItems: PROCEDURE =
    BEGIN
    nfl, fl: POINTER TO FrameItem;
    nfl ← Flist;
    UNTIL (fl ← nfl) = NIL DO
      nfl ← fl.next;
      SystemDefs.FreeHeapNode[fl];
      ENDLOOP;
    Flist ← NIL;
    RETURN
    END;

  WriteFrameItems: PROCEDURE =
    BEGIN OPEN IODefs;
    fl: POINTER TO FrameItem ← Flist;
    DebugMiscDefs.WriteEOL[];
    WriteChar['!]; WriteString[s]; WriteString[" has frames at"L];
    UNTIL fl = NIL DO
      WriteChar[' ]; WriteOctal[fl.faddr];
      fl ← fl.next;
      ENDLOOP;
    DebugMiscDefs.WriteEOL[];
    WriteString["Use SEt Octal context or Display Frame command."L];
    RETURN
    END;

  FindModuleString: PROCEDURE [mth: MTHandle, mti: MTIndex] RETURNS[BOOLEAN] =
    BEGIN OPEN DebugUtilityDefs, StringDefs;
    gfi: GFTIndex;
    ssd: SubStringDescriptor ←
      [base: @DDptr.ssb.string, offset: mth.name, length: DDptr.ssb.size[mth.name]];
    ss: SubString ← @ssd;
    fl: POINTER TO FrameItem ← Flist;
    IF StreamDefs.ControlDELtyped[] THEN CleanupControlDEL[DDptr.bcdseg];
    IF ~SameConfig[bcd, mth.config, DDptr.cti] THEN RETURN[FALSE];
    IF (EquivalentSubStrings[ss, modss] AND DDptr.caseignoring)
      OR EqualSubStrings[ss, modss] THEN
      BEGIN
      IF DeletedFrame[gfi ← MapConfigToReal[mth.gfi, DDptr.config]]
        THEN RETURN [FALSE];
      Flist ← SystemDefs.AllocateHeapNode[SIZE[FrameItem]];
      Flist↑ ← FrameItem[next: fl, faddr: MREAD[@GFT[gfi].frame]];
      Fcount ← Fcount + 1;
      END;
    RETURN[FALSE]
    END;

  moddesc ← StringDefs.SubStringDescriptor[base: s, offset: 0, length: s.length];
  [] ← InputLoadState[];
  bcd ← DAcquireBcd[];
  [,] ← EnumerateModuleTable[bcd, FindModuleString];
  DReleaseBcd[];
  ReleaseLoadState[];
  IF Fcount = 0 THEN SIGNAL DebugMiscDefs.LookupFail[s];
  IF Fcount = 1 THEN f ← Flist.faddr
  ELSE BEGIN
    SIGNAL MultipleInstances[s ! UNWIND => FreeFrameItems[]];
    f ← NullGlobalFrame;
    WriteFrameItems[];
    END;
  FreeFrameItems[];
  RETURN[f]
  END;

MultipleInstances: PUBLIC SIGNAL [s: STRING] = CODE;

InvalidGlobalFrame: PUBLIC ERROR [f: ControlDefs.GlobalFrameHandle] = CODE;

FrameToModuleName: PUBLIC PROCEDURE [f: GlobalFrameHandle, s: STRING] =
  BEGIN OPEN DebugUtilityDefs, ControlDefs;
  cgfi: GFTIndex;
  newconfig: ConfigIndex;
  tempssb: NameString;
  newbcdseg: FileSegmentHandle;
  newbcd: BcdBase;

  FindModuleString: PROCEDURE [mth: MTHandle, mti: MTIndex]
    RETURNS [BOOLEAN] =
    BEGIN
    ssd: StringDefs.SubStringDescriptor;
    IF cgfi IN[mth.gfi..mth.gfi+mth.ngfi) THEN
      BEGIN
      ssd ← [base: @tempssb.string, offset: mth.name,
        length: tempssb.size[mth.name]];
      StringDefs.AppendSubString[s,@ssd];
      RETURN[TRUE];
      END;
    RETURN[FALSE];
    END;

  [cgfi,newconfig] ← MapRC[
    IF VirtualGlobalFrame[f].copied THEN FindOriginal[f] ELSE f];
  IF newconfig = ConfigNull THEN ERROR InvalidGlobalFrame[f];
  IF newconfig # DDptr.config OR DDptr.initBCD THEN
    BEGIN
    newbcd ← SetUpBcd[newbcdseg ← BcdSegFromLoadState[newconfig]];
    tempssb ← LOOPHOLE[newbcd+newbcd.ssOffset];
    [] ← EnumerateModuleTable[newbcd, FindModuleString];
    ReleaseBcdSeg[newbcdseg];
    END
  ELSE
    BEGIN newbcd ← DAcquireBcd[];
    tempssb ← DDptr.ssb;
    [] ← EnumerateModuleTable[newbcd, FindModuleString];
    DReleaseBcd[];
    END;
  RETURN
  END;

ResetContext: PUBLIC PROCEDURE [f: ControlDefs.FrameHandle,
  psb: ProcessDefs.ProcessHandle] =
  BEGIN
  WriteLocalContext[f];
  DDptr.pContext ← psb;
  RETURN
  END;

WriteLocalContext: PUBLIC PROCEDURE [f: ControlDefs.FrameHandle] =
  BEGIN
  g: GlobalFrameHandle ← DebugUtilityDefs.MREAD[@f.accesslink];
  IF ~DebugUtilityDefs.CheckFrame[f] THEN
    BEGIN CleanupInvalidContext[f]; RETURN END;
  BEGIN
  WriteContext[g !SegmentDefs.InvalidFP => GOTO exit];
  DDptr.lContext ← f;
  DDptr.gContext ← g;
  EXITS
    exit => NULL;
  END;
  RETURN
  END;

WriteGlobalContext: PUBLIC PROCEDURE [f: ControlDefs.GlobalFrameHandle] =
  BEGIN
  IF ~DebugUtilityDefs.ValidGlobalFrame[f] THEN
    BEGIN CleanupInvalidContext[f]; RETURN END;
  BEGIN
  WriteContext[f !SegmentDefs.InvalidFP => GOTO exit];
  DDptr.lContext ← NIL;
  DDptr.gContext ← f;
  EXITS
    exit => NULL;
  END;
  RETURN
  END;

CleanupInvalidContext: PROCEDURE [f: UNSPECIFIED] =
  BEGIN
  DebugMiscDefs.WriteEOL[];
  IODefs.WriteOctal[f];
  IODefs.WriteString[" is not a valid frame!"L];
  IF ~DDptr.initBCD THEN SIGNAL DebugMiscDefs.DebugAbort
  ELSE
    BEGIN InitConfig[]; DDptr.initBCD ← FALSE; END;
  RETURN
  END;

WriteContext: PROCEDURE [f: ControlDefs.GlobalFrameHandle] =
  BEGIN OPEN DebugUtilityDefs, ControlDefs;
  cgfi: GFTIndex;
  newconfig: ConfigIndex;
  bcd: BcdBase;

  FindWhichModule: PROCEDURE[mth: MTHandle, mti: MTIndex] RETURNS [BOOLEAN] =
    BEGIN
    IF cgfi IN [mth.gfi..mth.gfi+mth.ngfi) THEN
      BEGIN DDptr.cti ← mth.config; RETURN[TRUE]; END;
    RETURN[FALSE];
    END;

  BEGIN
  DDptr.pContext ← NIL;
  [] ← InputLoadState[ ! LoadStateInvalid => GOTO noContext];
  [cgfi,newconfig] ← MapRC[
    IF VirtualGlobalFrame[f].copied THEN FindOriginal[f] ELSE f];
  IF newconfig = ConfigNull THEN ERROR InvalidGlobalFrame[f];
  IF newconfig # DDptr.config OR DDptr.initBCD THEN
    BEGIN
    DDptr.initBCD ← FALSE;
    DDptr.config ← newconfig;
    IF DDptr.bcdseg # NIL THEN SegmentDefs.DeleteFileSegment[DDptr.bcdseg];
    bcd ← SetUpBcd[DDptr.bcdseg ← BcdSegFromLoadState[DDptr.config]];
    DDptr.ssb ← LOOPHOLE[bcd+bcd.ssOffset];
    END
  ELSE bcd ← DAcquireBcd[];
  [] ← EnumerateModuleTable[bcd, FindWhichModule];
  DReleaseBcd[];
  ReleaseLoadState[];
  EXITS
    noContext =>
      BEGIN DDptr.lContext ← NIL; DDptr.gContext ← NIL; END;
  END;
  RETURN
  END;

MapRC: PUBLIC PROCEDURE [f: GlobalFrameHandle]
  RETURNS [cgfi: GFTIndex, config: ConfigIndex] =
  BEGIN
  [cgfi, config] ← MapRealToConfig[DebugUtilityDefs.ReadGlobalGFI[f]];
  RETURN
  END;

GlobalFrame: TYPE = ControlDefs.GlobalFrame;
globalFrame: GlobalFrame;

VirtualGlobalFrame: PUBLIC PROCEDURE [frame: GlobalFrameHandle]
  RETURNS [GlobalFrameHandle] =
  BEGIN OPEN DebugMiscDefs;
  CopyRead[to: @globalFrame, from: frame, nwords: SIZE[GlobalFrame]];
  RETURN[@globalFrame]
  END;

FindOriginal: PUBLIC PROCEDURE [copy: GlobalFrameHandle]
  RETURNS [GlobalFrameHandle] =
  BEGIN
  Original: PROCEDURE [f: GlobalFrameHandle] RETURNS [BOOLEAN] =
    BEGIN
    RETURN[f # copy AND ~VirtualGlobalFrame[f].copied AND
      SameModule[copy, f]];
    END;
  RETURN[DebugUtilityDefs.ReverseEnumerateGlobalFrames[Original]]
  END;

SameModule: PROCEDURE [f1, f2: GlobalFrameHandle] RETURNS [BOOLEAN] =
  BEGIN OPEN DebugUtilityDefs;
  o1, o2: CARDINAL;
  s1, s2: FileSegmentHandle;
  fcb: ControlDefs.FrameCodeBase;
  s1 ← MREAD[@f1.codesegment];
  s2 ← MREAD[@f2.codesegment];
  IF s1 # s2 THEN RETURN[FALSE];
  fcb ← MREAD[@f1.code];
  IF ~fcb.swappedout THEN
    o1 ← fcb.codebase - UserFileSegmentAddress[s1]
  ELSE
    BEGIN
    fcb.swappedout ← FALSE;
    o1 ← fcb.offset;
    END;
  fcb ← MREAD[@f2.code];
  IF ~fcb.swappedout THEN
    o2 ← fcb.codebase - UserFileSegmentAddress[s2]
  ELSE
    BEGIN
    fcb.swappedout ← FALSE;
    o2 ← fcb.offset;
    END;
  RETURN[o1 = o2];
  END;

UserFileSegmentAddress: PROCEDURE [seg: FileSegmentHandle] RETURNS [POINTER] =
  BEGIN OPEN DebugMiscDefs, SegmentDefs;
  lseg: FileSegmentObject;
  IF LOOPHOLE[seg, CARDINAL] <= 255 THEN RETURN[LOOPHOLE[0]];
  CopyRead[from: seg, to: @lseg, nwords: SIZE[FileSegmentObject]];
  RETURN[LOOPHOLE[lseg.VMpage*AltoDefs.PageSize]]
  END;

WhereAmI: PUBLIC PROCEDURE =
  BEGIN OPEN IODefs;
  module: STRING ← DebugMiscDefs.DGetString[40];
  bcd: BcdBase;
  ctb: CARDINAL;
  BEGIN
  [] ← InputLoadState[ ! DebugUtilityDefs.LoadStateInvalid => GOTO noContext];
  WriteLine["context --"L];
  WriteString["  Module: "L];
  FrameToModuleName[DDptr.gContext, module];
  WriteString[module];
  WriteString[", G: "L];
  WriteOctal[DDptr.gContext];
  IF DDptr.lContext # NIL THEN
    BEGIN WriteString[", L: "L]; WriteOctal[DDptr.lContext]; END;
  IF DDptr.pContext # NIL THEN
    BEGIN WriteString[", PSB: "L]; WriteOctal[DDptr.pContext]; END;
  WriteChar[CR];
  bcd ← DAcquireBcd[];
  IF bcd.nConfigs # 0 THEN
    BEGIN
    WriteString["  Configuration: "L];
    ctb ← LOOPHOLE[bcd+bcd.ctOffset];
    IF (ctb+DDptr.cti).namedinstance THEN
      BEGIN
      PrintName[DDptr.ssb, FindName[bcd,[config[DDptr.cti]]]];
      IODefs.WriteString[": "L];
      END;
    PrintName[DDptr.ssb,(ctb+DDptr.cti).name];
    END;
  DReleaseBcd[];
  ReleaseLoadState[];
  DebugMiscDefs.DFreeString[module];
  EXITS
    noContext => IODefs.WriteString["No valid context!!"L];
  END;
  RETURN
  END;

WriteWorld: PUBLIC PROCEDURE =
  BEGIN OPEN DebugUtilityDefs;

  GFWrite: PROCEDURE[frame: ControlDefs.GlobalFrameHandle] RETURNS [BOOLEAN] =
    BEGIN OPEN ControlDefs, IODefs;
    module: STRING;
  --code: UNSPECIFIED;
    IF frame = NullGlobalFrame THEN RETURN[FALSE];
    module ← DebugMiscDefs.DGetString[40];
    FrameToModuleName[frame, module];
    WriteString[module];
    WriteString[", G:"L]; WriteOctal[frame];
  --IF (code ← MREAD[@frame.code]) MOD 2 = 0 THEN WriteString[", C:"L]
  --ELSE WriteString[", offset:"L];
  --WriteOctal[code];
    WriteString[", gfi:"L];
    WriteOctal[ReadGlobalGFI[frame]];
    DebugMiscDefs.WriteEOL[];
    IF StreamDefs.ControlDELtyped[] THEN
      BEGIN
      LoadStateDefs.ReleaseLoadState[];
      DebugMiscDefs.DFreeString[module];
      SIGNAL DebugMiscDefs.ControlDEL;
      END;
    DebugMiscDefs.DFreeString[module];
    RETURN[FALSE]
    END;

  DebugMiscDefs.WriteEOL[];
  [] ← LoadStateDefs.InputLoadState[];
  [] ← ReverseEnumerateGlobalFrames[GFWrite];
  LoadStateDefs.ReleaseLoadState[];
  RETURN
  END;

InvalidImageFile: PUBLIC SIGNAL [image: STRING] = CODE;

AttachImageFile: PUBLIC PROCEDURE [name: STRING]=
  BEGIN OPEN SegmentDefs;
  file: FileHandle;
  oldseg, seg: FileSegmentHandle;
  image: POINTER TO ImageDefs.ImageHeader;
  base, pages: CARDINAL;
  CheckForExtension[name, ".image"L];
  file ← DebugUtilityDefs.CacheNewFile[name, DefaultAccess !
    FileNameError =>
      BEGIN
      CommandDefs.WriteSignalString[file];
      IODefs.WriteString[name];
      SIGNAL DebugMiscDefs.DebugAbort;
      END];
  seg ← NewFileSegment[file, 1, 1, Read];
  SwapIn[seg];
  image ← FileSegmentAddress[seg];
  IF image.prefix.versionident # ImageDefs.VersionID
    THEN SIGNAL InvalidImageFile[name ! UNWIND =>
      BEGIN Unlock[seg]; DeleteFileSegment[seg]; END];
  base ← image.prefix.initialLoadStateBase;
  pages ← image.prefix.loadStatePages;
  Unlock[seg];
  MoveFileSegment[seg, base, pages];
  IF (oldseg ← GetLoadState[]) # NIL THEN
    BEGIN
    UNTIL oldseg.lock = 0 DO Unlock[oldseg]; ENDLOOP;
    DeleteFileSegment[oldseg];
    END;
  SetLoadState[seg];
  DebugContextDefs.InitBCD[];
  DebugUtilityDefs.InvalidateFileCache[];
  DebugSymbolDefs.DCleanSymbolItems[];
  RETURN
  EXITS return => RETURN
  END;

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

InitConfig: PROCEDURE =
  BEGIN
  bcd: BcdBase;
  DDptr.lContext ← NIL;
  DDptr.gContext ← NIL;
  DDptr.pContext ← NIL;
  IF DDptr.bcdseg # NIL THEN
    BEGIN
    SegmentDefs.DeleteFileSegment[DDptr.bcdseg];
    DDptr.bcdseg ← NIL;
    END;
  BEGIN
  DDptr.config ← InputLoadState[
    ! DebugUtilityDefs.LoadStateInvalid => GOTO noContext] - 1;
  bcd ← SetUpBcd[DDptr.bcdseg ← BcdSegFromLoadState[DDptr.config]];
  DDptr.ssb ← LOOPHOLE[bcd+bcd.ssOffset];
  DDptr.cti ← IF bcd.nConfigs = 0 AND bcd.nModules = 1
    THEN CTNull ELSE FIRST[CTIndex];
  DReleaseBcd[];
  ReleaseLoadState[];
  EXITS
    noContext =>
      BEGIN DDptr.config ← ConfigNull; DDptr.cti ← CTNull; END;
  END;
  RETURN
  END;


END...
```