```
-- DebugConfig.mesa
-- Edited by:
--            Johnsson on August 30, 1978  12:01 PM
--            Sandman on May 25, 1978  9:43 AM
--            Barbara on July 13, 1978  2:23 PM

DIRECTORY
  BcdDefs: FROM "bcddefs" USING [
    BCD, CTHandle, CTIndex, CTNull, FTHandle, FTNull, FTSelf, MTHandle,
    MTIndex, MTNull, NameRecord, NameString, SGIndex, VersionStamp],
  ControlDefs: FROM "controldefs" USING [
    GFT, GFTIndex, GFTItem, GlobalFrameHandle, NullEpBase, NullGlobalFrame],
  DebugContextDefs: FROM "debugContextdefs" USING [InvalidGlobalFrame, MapRC],
  DebugData: FROM "debugdata" USING [
    bcdseg, caseignoring, config, cti, gContext, initBCD, lContext, pContext,
    ssb],
  DebuggerDefs: FROM "debuggerdefs" USING [WriteBlanks, WriteSubString],
  DebugMiscDefs: FROM "debugmiscdefs" USING [
    ControlDEL, LookupFail, WriteEOL],
  DebugSymbolDefs: FROM "debugsymboldefs" USING [SymbolsForGFrame],
  DebugUsefulDefs: FROM "debugusefuldefs",
  DebugUtilityDefs: FROM "debugutilitydefs" USING [
    CacheNewFile, CodeFile, FileName, FindOriginal,
    LoadStateInvalid, MREAD, ValidGlobalFrame, VirtualGlobalFrame],
  IODefs: FROM "iodefs" USING [
    WriteChar, WriteDecimal, WriteOctal, WriteString],
  LoaderBcdUtilDefs: FROM "loaderbcdutildefs" USING [
    BcdBase, EnumerateConfigTable, EnumerateModuleTable, FindName,
    ReleaseBcdSeg, SetUpBcd],
  LoadStateDefs: FROM "loadstatedefs" USING [
    BcdAddress, BcdSegFromLoadState, ConfigIndex, ConfigNull,
    EnumerateLoadStateBcds, GFTIndex, InitializeRelocation, InputLoadState,
    MapConfigToReal, ReleaseLoadState, ReleaseRelocation, Relocation],
  SegmentDefs: FROM "segmentdefs" USING [
    DefaultAccess, DeleteFileSegment, FileError, FileHandle, FileNameError,
    FileSegmentAddress, FileSegmentHandle, MoveFileSegment,
    NewFileSegment, Read, SwapIn, SwapOut, Unlock],
  StreamDefs: FROM "streamdefs" USING [ControlDELtyped],
  StringDefs: FROM "stringdefs" USING [
    AppendString, AppendSubString, EqualSubStrings, EquivalentSubStrings,
    SubStringDescriptor],
  SymbolTableDefs: FROM "symboltabledefs" USING [SegmentForTable];

DEFINITIONS FROM BcdDefs, LoaderBcdUtilDefs, LoadStateDefs;

DebugConfig: PROGRAM
IMPORTS DDptr: DebugData, DebugContextDefs, DebuggerDefs, DebugMiscDefs,
  DebugSymbolDefs, DebugUtilityDefs, IODefs, LoaderBcdUtilDefs, LoadStateDefs,
  SegmentDefs, StreamDefs, StringDefs, SymbolTableDefs
EXPORTS DebugUsefulDefs, DebugContextDefs =

BEGIN

FileSegmentHandle: TYPE = SegmentDefs.FileSegmentHandle;

ListConfigurations: PUBLIC PROCEDURE =
  BEGIN
  GetBcdSetUp: PROCEDURE [config: ConfigIndex, bcdAddr: BcdAddress]
  RETURNS [BOOLEAN] =
    BEGIN
    bcdseg: FileSegmentHandle ← NIL;
    bcd: BcdBase;
    tempssb: NameString;

    ListSons: PROCEDURE [level: CARDINAL, parent: CTIndex] =
      BEGIN
      WriteConfigNames: PROCEDURE [cth: CTHandle, cti: CTIndex]
      RETURNS [BOOLEAN] =
        BEGIN
        IF cth.config = parent THEN
          BEGIN
          IF StreamDefs.ControlDELtyped[] THEN CleanupControlDEL[bcdseg];
          DebugMiscDefs.WriteEOL[];
          DebuggerDefs.WriteBlanks[level*2];
          IF cth.namedinstance THEN
            BEGIN
```

```
            PrintName[tempssb, FindName[bcd,[config[cti]]]];
            IODefs.WriteString[": "L];
            END;
          PrintName[tempssb, cth.name];
          ListSons[level+1, cti];
          END;
        RETURN[FALSE]
        END;

      [] ← EnumerateConfigNames[bcd, WriteConfigNames];
      RETURN
      END;

    IF config = DDptr.config AND ~DDptr.initBCD THEN bcd ← DAcquireBcd[]
    ELSE bcd ← SetUpBcd[bcdseg ← BcdSegFromLoadState[config]];
    tempssb ← LOOPHOLE[bcd+bcd.ssOffset];
    ListSons[0, CTNull];
    IF bcdseg # NIL THEN ReleaseBcdSeg[bcdseg] ELSE DReleaseBcd[];
    RETURN[FALSE]
    END;

  [] ← InputLoadState[];
  [] ← EnumerateLoadStateBcds[recentfirst, GetBcdSetUp];
  ReleaseLoadState[];
  RETURN
  END;

EnumerateConfigNames: PROCEDURE[bcd: BcdBase,
proc: PROCEDURE [CTHandle, CTIndex] RETURNS [BOOLEAN]]
RETURNS[CTIndex]=
  BEGIN
  mtb: CARDINAL = LOOPHOLE[bcd+bcd.mtOffset];
  mti: MTIndex ← FIRST[MTIndex];
  tempssb: NameString = LOOPHOLE[bcd+bcd.ssOffset];
  IF bcd.nConfigs = 0 AND bcd.nModules = 1 THEN
    BEGIN OPEN m: mtb+mti;
    DebugMiscDefs.WriteEOL[];
    IF m.namedinstance THEN
      BEGIN
      PrintName[tempssb, FindName[bcd,[module[mti]]]];
      IODefs.WriteString[": "L];
      END;
    PrintName[tempssb, m.name];
    RETURN[CTNull];
    END;
  RETURN[EnumerateConfigTable[bcd, proc].cti]
  END;

DisplayConfiguration: PUBLIC PROCEDURE =
  BEGIN OPEN DebugUtilityDefs, ControlDefs;
  ctb: CARDINAL;
  bcd: BcdBase;
  rel: Relocation;
  GFT: POINTER TO ARRAY [0..0) OF GFTItem = ControlDefs.GFT;

  PrintModules: PROCEDURE [mth: MTHandle, mti: MTIndex] RETURNS [BOOLEAN] =
    BEGIN
    gft: GFTIndex;
    IF StreamDefs.ControlDELtyped[] THEN CleanupControlDEL[DDptr.bcdseg];
    DebugMiscDefs.WriteEOL[];
    IF ~SameConfig[bcd, mth.config, DDptr.cti] THEN RETURN[FALSE];
    IF mth.namedinstance THEN
      BEGIN
      PrintName[DDptr.ssb, FindName[bcd,[module[mti]]]];
      IODefs.WriteString[": "L];
      END;
    PrintName[DDptr.ssb, mth.name]; IODefs.WriteString[", G: "L];
    IF DeletedFrame[gft ← rel[mth.gfi]] THEN IODefs.WriteString[" deleted"L]
      ELSE IODefs.WriteOctal[MREAD[@GFT[gft].frame]];
    RETURN[FALSE];
    END;

  [] ← InputLoadState[];
  bcd ← DAcquireBcd[];
  rel ← InitializeRelocation[DDptr.config];
  DebuggerDefs.WriteBlanks[2];
```

```
  IF bcd.nConfigs # 0 THEN
    BEGIN
    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;
  [] ← EnumerateModuleTable[bcd,PrintModules];
  ReleaseRelocation[rel];
  DReleaseBcd[];
  ReleaseLoadState[];
  RETURN;
  END;

SetRootConfiguration: PUBLIC PROCEDURE [configname: STRING] =
  BEGIN
  Rcount: CARDINAL ← 0;
  configdesc: StringDefs.SubStringDescriptor;
  savecti: CTIndex ← CTNull;
  saveconfig: ConfigIndex;

  GetSetUp: PROCEDURE [config: ConfigIndex, bcdAddr: BcdAddress]
  RETURNS [BOOLEAN] =
    BEGIN
    bcdseg: FileSegmentHandle ← NIL;
    found: BOOLEAN ← FALSE;
    bcd: BcdBase;
    mtb: CARDINAL;
    mti: MTIndex;

    CheckForRoot: PROCEDURE [cth: CTHandle, cti: CTIndex] RETURNS [BOOLEAN] =
      BEGIN
      IF StreamDefs.ControlDELtyped[] THEN CleanupControlDEL[
        IF config = DDptr.config AND ~DDptr.initBCD THEN DDptr.bcdseg ELSE bcdseg];
      IF cth.config # CTNull THEN RETURN[FALSE];
      IF ~(found ← TestName[cth.name]) THEN
        IF cth.namedinstance
          THEN found ← TestName[FindName[bcd,[config[cti]]]];
      IF found THEN
        BEGIN savecti ← cti; Rcount ← Rcount +1; saveconfig ← config; END;
      RETURN[FALSE]
      END;

    TestName: PROCEDURE [name: NameRecord] RETURNS [BOOLEAN] =
      BEGIN OPEN StringDefs;
      tempssb: NameString = LOOPHOLE[bcd+bcd.ssOffset];
      ssd: SubStringDescriptor ←
        [base: @tempssb.string, offset: name, length: tempssb.size[name]];
      IF DDptr.caseignoring
        THEN RETURN[EquivalentSubStrings[@configdesc, @ssd]]
      ELSE RETURN[EqualSubStrings[@configdesc, @ssd]]
      END;

    IF config = DDptr.config AND ~DDptr.initBCD THEN bcd ← DAcquireBcd[]
    ELSE bcd ← SetUpBcd[bcdseg ← BcdSegFromLoadState[config]];
    mtb ← LOOPHOLE[bcd+bcd.mtOffset];
    IF bcd.nConfigs = 0 AND bcd.nModules = 1 THEN
      BEGIN
      mti ← FIRST[MTIndex];
      IF ~(found ← TestName[(mtb+mti).name]) THEN
        IF (mtb+mti).namedinstance
          THEN found ← TestName[FindName[bcd,[module[mti]]]];
      IF found THEN
        BEGIN Rcount ← Rcount+1; saveconfig ← config; savecti ← CTNull; END;
      END
    ELSE [] ← EnumerateConfigTable[bcd, CheckForRoot];
    IF bcdseg # NIL THEN ReleaseBcdSeg[bcdseg] ELSE DReleaseBcd[];
    RETURN[FALSE]
    END;

  configdesc ← StringDefs.SubStringDescriptor[
    base: configname, offset: 0, length: configname.length];
  [] ← InputLoadState[];
```

```
  [] ← EnumerateLoadStateBcds[recentfirst,GetSetUp];
  IF Rcount # 1 THEN ReleaseLoadState[];
  IF Rcount = 0 THEN SIGNAL DebugMiscDefs.LookupFail[configname];
  IF Rcount = 1 THEN SetupRootConfig[saveconfig, savecti]
    ELSE WriteAmbiguousContext[configname, Rcount];
  RETURN
  END;

WriteAmbiguousContext: PROCEDURE [s: STRING, c: CARDINAL] =
  BEGIN OPEN IODefs;
  WriteChar['!]; WriteString[s]; WriteString[" has "L];
  WriteDecimal[c];
  WriteString[" instances -- this is an ambiguous reference."L];
  DebugMiscDefs.WriteEOL[];
  RETURN
  END;

SetupRootConfig: PROCEDURE [config: ConfigIndex, cti: CTIndex] =
  BEGIN
  bcd: BcdBase;
  bcdseg: FileSegmentHandle;
  IF config # DDptr.config THEN
    BEGIN
    bcd ← SetUpBcd[bcdseg ← BcdSegFromLoadState[config]];
    IF SetupConfig[bcd, cti, config] THEN
      BEGIN
      SegmentDefs.DeleteFileSegment[DDptr.bcdseg];
      DDptr.bcdseg ← bcdseg; DDptr.config ← config;
      DDptr.ssb ← LOOPHOLE[bcd+bcd.ssOffset];
      DDptr.cti ← cti;
      END
    ELSE ReleaseBcdSeg[bcdseg];
    END
  ELSE IF SetupConfig[DAcquireBcd[], cti, config] THEN DDptr.cti ← cti;
  DReleaseBcd[];
  RETURN
  END;

SetConfiguration: PUBLIC PROCEDURE [s: STRING] =
  BEGIN
  bcd: BcdBase;
  count: CARDINAL ← 0;
  configdesc: StringDefs.SubStringDescriptor;
  savecti: CTIndex;

  CheckConfigName: PROCEDURE [cth: CTHandle, cti: CTIndex] RETURNS [BOOLEAN] =
    BEGIN
    found: BOOLEAN ← FALSE;

    TestName: PROCEDURE [name: NameRecord] RETURNS [BOOLEAN] =
      BEGIN OPEN StringDefs;
      ssd: SubStringDescriptor ←
        [base: @DDptr.ssb.string, offset: name, length: DDptr.ssb.size[name]];
      IF DDptr.caseignoring
        THEN RETURN[EquivalentSubStrings[@configdesc, @ssd]]
      ELSE RETURN[EqualSubStrings[@configdesc, @ssd]]
      END;

    IF StreamDefs.ControlDELtyped[] THEN SIGNAL DebugMiscDefs.ControlDEL;
    IF ~SameConfig[bcd, cth.config, DDptr.cti] THEN RETURN[FALSE];
    IF ~(found ← TestName[cth.name]) THEN
      IF cth.namedinstance THEN
        found ← TestName[FindName[bcd,[config[cti]]]];
    IF found THEN BEGIN count ← count + 1; savecti ← cti; END;
    RETURN[FALSE]
    END;

  configdesc ← StringDefs.SubStringDescriptor[
    base: s, offset: 0, length: s.length];
  IF DDptr.cti = CTNull
    THEN BEGIN IODefs.WriteString[" -- Not allowed !"L]; RETURN END;
  bcd ← DAcquireBcd[];
  [] ← EnumerateConfigTable[bcd, CheckConfigName];
  IF count = 0 THEN
    BEGIN DReleaseBcd[]; SIGNAL DebugMiscDefs.LookupFail[s]; END;
  IF count = 1 THEN
```

```
    BEGIN
    [] ← InputLoadState[];
    IF SetupConfig[bcd, savecti, DDptr.config] THEN DDptr.cti ← savecti;
    END
  ELSE WriteAmbiguousContext[s, count];
  DReleaseBcd[];
  RETURN
  END;

SetupConfig: PROCEDURE [bcd: BcdBase, cti: CTIndex, config: ConfigIndex] RETURNS [BOOLEAN] =
  BEGIN OPEN ControlDefs;
  ctb: CARDINAL ← LOOPHOLE[bcd+bcd.ctOffset];
  mtb: CARDINAL ← LOOPHOLE[bcd+bcd.mtOffset];
  mti1: MTIndex ← FIRST[MTIndex];

  FindFirstModule: PROCEDURE[mth: MTHandle, mti: MTIndex] RETURNS [BOOLEAN] =
    BEGIN OPEN DebugUtilityDefs;
    gft: GFTIndex;
    IF ~SameConfig[bcd, mth.config, cti] OR
      DeletedFrame[gft ← MapConfigToReal[mth.gfi, config]]
      THEN RETURN[FALSE];
    DDptr.gContext ← MREAD[@GFT[gft].frame];
    DDptr.lContext ← NIL;
    DDptr.pContext ← NIL;
    RETURN[TRUE];
    END;

  BEGIN
  IF cti = CTNull THEN
    BEGIN
    IF ~FindFirstModule[(mtb+mti1), mti1] THEN GOTO notallowed
    ELSE GOTO done;
    END;
  IF (ctb+cti).control # MTNull THEN
    IF FindFirstModule[(ctb+cti).control+mtb,(ctb+cti).control]
      THEN GOTO done;
  IF EnumerateModuleTable[bcd, FindFirstModule].mti = MTNull
    THEN GOTO notallowed;
  EXITS
    notallowed =>
      BEGIN
      IODefs.WriteString[" -- Not Allowed !"L];
      ReleaseLoadState[];
      RETURN[FALSE];
      END;
    done => NULL;
  END;
  ReleaseLoadState[];
  RETURN[TRUE];
  END;

EnumerateConfiguration: PUBLIC PROCEDURE [
  p: PROCEDURE [GlobalFrameHandle] RETURNS [BOOLEAN]] =
  --sequences through frames of modules in current config
  BEGIN OPEN ControlDefs;
  bcd: BcdBase;
  rel: Relocation;

  SearchModules: PROCEDURE [mth: MTHandle, mti: MTIndex] RETURNS [BOOLEAN] =
    BEGIN
    frame: GlobalFrameHandle;
    IF StreamDefs.ControlDELtyped[] THEN SIGNAL DebugMiscDefs.ControlDEL;
    IF ~SameConfig[bcd, mth.config, DDptr.cti] THEN RETURN[FALSE];
    frame ← DebugUtilityDefs.MREAD[@GFT[rel[mth.gfi]].frame];
    IF frame = NullGlobalFrame THEN RETURN[FALSE];
    IF ~DebugUtilityDefs.ValidGlobalFrame[frame]
      THEN ERROR DebugContextDefs.InvalidGlobalFrame[frame];
    IF p[frame] THEN RETURN[TRUE];
    RETURN[FALSE];
    END;

  BEGIN
  [] ← InputLoadState[ !DebugUtilityDefs.LoadStateInvalid => GOTO nil];
  rel ← InitializeRelocation[DDptr.config];
  [] ← EnumerateModuleTable[bcd ← DAcquireBcd[], SearchModules !UNWIND =>
     BEGIN ReleaseRelocation[rel]; DReleaseBcd[]; ReleaseLoadState[]; END];
```

```
  ReleaseRelocation[rel];
  DReleaseBcd[];
  ReleaseLoadState[];
  EXITS
    nil => RETURN;
  END;
  RETURN
  END;

GlobalFrameHandle: TYPE = ControlDefs.GlobalFrameHandle;

SymbolSegForFrame: PUBLIC PROCEDURE [f: GlobalFrameHandle]
  RETURNS [seg: FileSegmentHandle] =
  BEGIN OPEN DebugContextDefs, DebugUtilityDefs, SegmentDefs;
  cgfi: GFTIndex;
  config: ConfigIndex;
  bcdseg: FileSegmentHandle ← NIL;
  bcd: POINTER TO BcdDefs.BCD;
  cdesc, segdesc: BcdDefs.SGIndex;
  Cleanup: PROCEDURE =
    BEGIN
    IF bcdseg # NIL THEN ReleaseBcdSeg[bcdseg] ELSE DReleaseBcd[];
    ReleaseLoadState[];
    RETURN;
    END;
  FindModule: PROCEDURE [mth: MTHandle, mti: MTIndex] RETURNS [BOOLEAN] =
    BEGIN
    IF cgfi IN [mth.gfi..mth.gfi+mth.ngfi) THEN
      BEGIN
      cdesc ← mth.code.sgi;
      segdesc ← mth.sseg;
      RETURN[TRUE];
      END;
    RETURN[FALSE];
    END;
  BEGIN OPEN DebugSymbolDefs;
  IF VirtualGlobalFrame[f].copied THEN
    RETURN[
      SymbolTableDefs.SegmentForTable[SymbolsForGFrame[FindOriginal[f]]]];
  [] ← InputLoadState[ ! LoadStateInvalid => GOTO nil];
  [cgfi,config] ← MapRC[f];
  IF config = ConfigNull THEN ERROR InvalidGlobalFrame[f];
  IF config = DDptr.config AND ~DDptr.initBCD THEN bcd ← DAcquireBcd[]
  ELSE bcd ← SetUpBcd[bcdseg ← BcdSegFromLoadState[config]];
  [] ← EnumerateModuleTable[bcd, FindModule];
  seg ← FindSegment[
    f, IF bcdseg # NIL THEN bcdseg ELSE DDptr.bcdseg, segdesc, cdesc !
      FileNameError, FileError => BEGIN seg ← NIL; CONTINUE END;
      UNWIND => Cleanup[]];
  Cleanup[];
  EXITS
    nil => RETURN[NIL];
  END;
  RETURN
  END;

IncorrectVersion: PUBLIC SIGNAL [file: STRING] = CODE;

FindSegment: PROCEDURE [frame: GlobalFrameHandle, seg: FileSegmentHandle, segdesc, codesegdesc: BcdDefs
**.SGIndex]
RETURNS [bcdseg: FileSegmentHandle] =
  BEGIN OPEN DebugUtilityDefs, SegmentDefs;
  ss: StringDefs.SubStringDescriptor;
  tempssb: NameString;
  name: STRING ← [40];
  file: FileHandle;
  symsbcd: POINTER TO BcdDefs.BCD;
  bcd: POINTER TO BcdDefs.BCD ← FileSegmentAddress[seg];
  sgb: CARDINAL ← LOOPHOLE[bcd+bcd.sgOffset];
  f: BcdDefs.FTHandle = LOOPHOLE[bcd+bcd.ftOffset, CARDINAL]
    +(sgb+segdesc).file;
  BEGIN
  SELECT (sgb+segdesc).file FROM
    BcdDefs.FTNull => RETURN[NIL];
    BcdDefs.FTSelf => RETURN[NewFileSegment[seg.file, (sgb+segdesc).base,
      (sgb+segdesc).pages+(sgb+segdesc).extraPages, Read]];
```

```
    (sgb+codesegdesc).file =>
      IF (file ← CodeFile[frame]) # NIL THEN GOTO found;
    ENDCASE;
  tempssb ← LOOPHOLE[bcd+bcd.ssOffset];
  ss ← [@tempssb.string, f.name, tempssb.size[f.name]];
  StringDefs.AppendSubString[name, @ss];
  CheckForExtension[name, ".bcd"L];
  file ← CacheNewFile[name, DefaultAccess];
  EXITS
    found => NULL;
  END;
  bcdseg ← NewFileSegment[file, 1, 1, Read];
  SwapIn[bcdseg];
  symsbcd ← FileSegmentAddress[bcdseg];
  IF ~EqVer[@symsbcd.version, @f.version] THEN
    BEGIN
    IF name.length = 0 THEN FileName[name, file];
    Unlock[bcdseg];
    DeleteFileSegment[bcdseg];
    SIGNAL IncorrectVersion[name];
    RETURN[NIL];
    END;
  Unlock[bcdseg];
  MoveFileSegment[bcdseg, (sgb+segdesc).base,
    (sgb+segdesc).pages+(sgb+segdesc).extraPages];
  RETURN;
  END;

EqVer: PROCEDURE [v1, v2: POINTER TO BcdDefs.VersionStamp] RETURNS [BOOLEAN] =
  BEGIN
  RETURN[v1.zapped OR v2.zapped OR v1↑ = v2↑];
  END;

--utilities

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;

CleanupControlDEL: PUBLIC PROCEDURE [bcdseg: SegmentDefs.FileSegmentHandle] =
  BEGIN
  IF bcdseg # DDptr.bcdseg AND bcdseg # NIL THEN ReleaseBcdSeg[bcdseg];
  DReleaseBcd[];
  ReleaseLoadState[];
  SIGNAL DebugMiscDefs.ControlDEL;
  RETURN
  END;

DAcquireBcd: PUBLIC PROCEDURE RETURNS [bcd: BcdBase] =
  BEGIN OPEN SegmentDefs;
  SwapIn[DDptr.bcdseg];
  bcd ← FileSegmentAddress[DDptr.bcdseg];
  DDptr.ssb ← LOOPHOLE[bcd+bcd.ssOffset];
  RETURN
  END;

DReleaseBcd: PUBLIC PROCEDURE =
  BEGIN OPEN SegmentDefs;
  Unlock[DDptr.bcdseg];
  IF DDptr.bcdseg.lock = 0 THEN SwapOut[DDptr.bcdseg];
  RETURN
  END;

PrintName: PUBLIC PROCEDURE [ssb: NameString, name: NameRecord] =
  BEGIN
  ssd: StringDefs.SubStringDescriptor ←
    [base: @ssb.string, offset: name, length: ssb.size[name]];
  DebuggerDefs.WriteSubString[@ssd];
  RETURN
  END;
```

```
SameConfig: PUBLIC PROCEDURE [bcd: BcdBase, child, parent: CTIndex]
RETURNS [BOOLEAN]=
  BEGIN
  cti: CTIndex;
  ctb: CARDINAL = LOOPHOLE[bcd+bcd.ctOffset];
  --checks to see if child is related to parent
  FOR cti ← child, (cti+ctb).config UNTIL cti = CTNull DO
    IF cti = parent THEN RETURN[TRUE];
    ENDLOOP;
  RETURN[parent = CTNull]
  END;

DeletedFrame:  PUBLIC PROCEDURE [gfi: ControlDefs.GFTIndex] RETURNS [BOOLEAN]=
  BEGIN OPEN DebugUtilityDefs, ControlDefs;
  RETURN[MREAD[@GFT[gfi].frame] = NullGlobalFrame AND
    MREAD[@GFT[gfi].epbase] = NullEpBase]
  END;

InitBCD: PUBLIC PROCEDURE =
  BEGIN
  DDptr.initBCD ← TRUE;
  RETURN
  END;

END...
```