```
-- Command.Mesa;
-- Edited by:
--          Sandman on May 2, 1978  9:26 PM
--          Barbara on July 31, 1978  4:31 PM
--          Johnsson on August 29, 1978  11:02 AM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [Address],
  BinaryDefs: FROM "binarydefs" USING [DIGrammar],
  CommandDefs: FROM "commanddefs" USING [
    CommandName, GetCommandString, IDCode, WriteCommandString,
    WriteErrorString, WriteIDString, WriteIString],
  ControlDefs: FROM "controldefs" USING [FrameHandle, GlobalFrameHandle],
  DebugBreakptDefs: FROM "debugbreakptdefs" USING [
    BreakPoint, ClearAllBT, ClearTextBreakPoint, ListAll, OctalBreakPoint,
    TextBreakPoint, TraceAll],
  DebugContextDefs: FROM "debugcontextdefs" USING [
    AttachImageFile, DisplayConfiguration, DisplayProcess, DisplayQueue,
    IncorrectVersion, ListConfigurations, ListProcesses, ResetContext,
    SetConfiguration, SetModuleContext, SetOctalContext, SetProcessContext,
    SetRootConfiguration, WhereAmI, WriteWorld],
  DebugData: FROM "debugdata" USING [caseignoring, worrybreaks, worryentry],
  DebugFTPDefs: FROM "debugftpdefs" USING [CallFTP],
  DebuggerDefs: FROM "debuggerdefs" USING [
    Display, DisplayFrame, DisplayStack, DumpCharacter, DumpVar, FormatRecord,
    FRPointer, LA, ModuleDump, SOPointer, SymbolObject],
  DebugInterpretDefs: FROM "debuginterpretdefs" USING [
    Iaddress, Iarray, Icall, Ideref, Iexpression, Ipointer, Istring],
  DebugMiscDefs: FROM "debugmiscdefs" USING [
    ControlDEL, coremap, DebugAbort, DebugProceed, DFreeString, DGetString,
    DisplayEvalStack, IgnoreComment, LookupFail, Quit, WriteCharZ, WriteEOL],
  DebugUsefulDefs: FROM "debugusefuldefs",
  DebugUtilityDefs: FROM "debugutilitydefs" USING [
    CheckFrame, CoreSwap, KillSession, LongREAD, LongWRITE, MREAD, MWRITE,
    UserProc, UserStart, ValidGlobalFrame],
  DebugSymbolDefs: FROM "debugsymboldefs" USING [AttachSymbols],
  DIActionDefs: FROM "diactiondefs" USING [
    CleanUp, espTosop, EvalStackEmpty, EvalStackOverflow, FreeStackItem,
    GetSetUp, IncorrectType, InvalidExpression, InvalidInterval, InvalidType,
    NILesp, NotImplemented, NotOnEvalStack, Transfer, TypesDontMatch,
    TypeStackEmpty, TypeStackOverflow],
  DIDefs: FROM "didefs" USING [
    ESPointer, hereESPointer, InvalidCharacter, InvalidNumber, Parse,
    ParseError, SyntaxError, TwoParse],
  DILitDefs: FROM "dilitdefs" USING [LitTabInit],
  DITypeDefs: FROM "ditypedefs" USING [
    SeiBoolean, SeiCardinal, SeiCharacter, SeiInteger, SeiLongInteger,
    SeiUnspecified, TypeString],
  InlineDefs: FROM "inlinedefs" USING [LDIVMOD],
  IODefs: FROM "iodefs" USING [
    ControlD, ControlF, ControlU, CR, DEL, ESC, NUL, NumberFormat, ReadChar,
    ReadEditedString, ReadID, ReadLine, Rubout, SP, WriteChar, WriteDecimal,
    WriteNumber, WriteOctal, WriteString],
  MiscDefs: FROM "miscdefs" USING [DestroyFakeModule],
  Mopcodes: FROM "mopcodes" USING [zKFCB],
  ProcessDefs: FROM "processdefs" USING [ProcessHandle],
  SDDefs: FROM "sddefs" USING [sAlternateBreak, sBreak, sCallDebugger, SD],
  StreamDefs: FROM "streamdefs" USING [ControlDELtyped, ResetControlDEL],
  StringDefs: FROM "stringdefs" USING [
    AppendChar, AppendLongNumber, AppendString, AppendSubString,
    EquivalentString, InvalidNumber, SubString, SubStringDescriptor],
  SystemDefs: FROM "systemdefs" USING [FreeHeapNode];

Command: PROGRAM
  IMPORTS BinaryDefs, CommandDefs, DebugBreakptDefs, DebugContextDefs,
    DDptr: DebugData, DebugFTPDefs, DebuggerDefs, DebugInterpretDefs,
    DebugMiscDefs, DebugSymbolDefs, DebugUtilityDefs, DIActionDefs, DIDefs,
    DILitDefs, DITypeDefs, IODefs, MiscDefs, StreamDefs, StringDefs,
    SystemDefs
  EXPORTS DebugMiscDefs, DebugUsefulDefs
  SHARES ProcessDefs =

BEGIN

FrameHandle: TYPE = ControlDefs.FrameHandle;
GlobalFrameHandle: TYPE = ControlDefs.GlobalFrameHandle;
```

```
Address: TYPE = AltoDefs.Address;
abort: SIGNAL = DebugMiscDefs.DebugAbort;
CR: CHARACTER = IODefs.CR;
NUL: CHARACTER = IODefs.NUL;
SP: CHARACTER = IODefs.SP;

commander: PUBLIC PROCEDURE [startingcontext: FrameHandle, startingpsb: ProcessDefs.ProcessHandle] =
  BEGIN OPEN DebugContextDefs, DebugUtilityDefs, DebugInterpretDefs, CommandDefs, DebuggerDefs, DebugBr
**eakptDefs, DebugMiscDefs, DebugSymbolDefs;
  GetChar ← FirstChar;
  SELECT ReadUChar[displayComs, kill] FROM
    'D => SELECT ReadUChar[module, config] FROM
            'M => ModuleDump[getID[CR, modulename]];
            'F => DisplayFrame[geteitherframe[':]];
            'V => DumpVar[getcolonid[CR, varname], mod];
            'E => DisplayEvalStack[];
            'S => DisplayStack[];
            'G => WriteWorld[];
            'P => DisplayProcess[getID[CR, processname]];
            'Q => DisplayQueue[getID[CR, queuename]];
            'C => DisplayConfiguration[];
            ENDCASE;
    'S => SELECT ReadUChar[start, setComs] FROM
            'T => BEGIN CheckWorry[]; UserStart[getgframe[CR]] END;
            'E => SELECT ReadUChar[config, rootCtx] FROM
                    'C => SetConfiguration[getcolonid[CR, configname]];
                    'M => SetModuleContext[getID[CR, modulename]];
                    'O => SetOctalContext[geteitherframe[NUL]];
                    'P => SetProcessContext[getID[CR, processname]];
                    'R => SetRootConfiguration[getcolonid[CR, rconfigname]];
                    ENDCASE;
            ENDCASE;
    'R => BEGIN confirm[]; ResetContext[startingcontext, startingpsb]; END;
    'C => SELECT ReadUChar[clearComs, current] FROM
            'U => WhereAmI[];
            'A => CaseSwitch[];
            'O => BEGIN confirm[]; coremap[] END;
            'L => SELECT ReadUChar[break, modBr] FROM
                    'B => ClearTextBreakPoint[getparam[proc, NUL],
                          getsource[], proc];
                    'T => ClearTextBreakPoint[getparam[proc, NUL],
                          getsource[], proc];
                    'A => SELECT ReadUChar[breaks, exits] FROM
                            'B => BEGIN confirm[]; ClearAllBT[break] END;
                            'T => BEGIN confirm[]; ClearAllBT[trace] END;
                            'E => TraceAll[getmodule[CR], clear, entry];
                            'X => TraceAll[getmodule[CR], clear, exit];
                            ENDCASE;
                    'E => SELECT ReadUChar[breakComs, traceComs] FROM
                            'B => BreakPoint[getparam[proc,CR], NIL,
                                  break, clear, entry];
                            'T => BreakPoint[getparam[proc,CR], NIL,
                                  trace, clear, entry];
                            ENDCASE;
                    'X => SELECT ReadUChar[breakComs, traceComs] FROM
                            'B => BreakPoint[getparam[proc,CR], NIL,
                                  break, clear, exit];
                            'T => BreakPoint[getparam[proc,CR], NIL,
                                  break, clear, exit];
                            ENDCASE;
                    'M => SELECT ReadUChar[breakComs, traceComs] FROM
                            'B => ClearTextBreakPoint[getmodule[NUL],
                                  getsource[], prog];
                            'T => ClearTextBreakPoint[getmodule[NUL],
                                  getsource[], prog];
                            ENDCASE;
                    ENDCASE;
            ENDCASE;
    'P => BEGIN confirm[]; SIGNAL DebugProceed END;
    'L => SELECT ReadUChar[configs, traces] FROM
            'C => BEGIN confirm[]; ListConfigurations[]; END;
            'P => BEGIN confirm[]; ListProcesses[]; END;
            'B => BEGIN confirm[]; ListAll[break]; END;
            'T => BEGIN confirm[]; ListAll[trace]; END;
            ENDCASE;
    'B => SELECT ReadUChar[entry, procBr] FROM
```

```
          'E => BreakPoint[getproccondition[CR],
           (IF conditionfound THEN condition ELSE NIL), break, set, entry];
          'X => BreakPoint[getproccondition[CR],
           (IF conditionfound THEN condition ELSE NIL), break, set, exit];
          'M => TextBreakPoint[getprogcondition[], getsource[],
                (IF conditionfound THEN condition ELSE NIL),
                break, prog];
          'P => TextBreakPoint[getproccondition[NUL], getsource[],
                (IF conditionfound THEN condition ELSE NIL),
                break, proc];
          ENDCASE;
    'T => SELECT ReadUChar[all, procBr] FROM
          'A => SELECT ReadUChar[entries, exits] FROM
                  'E => TraceAll[getmodule[CR], set, entry];
                  'X => TraceAll[getmodule[CR], set, exit];
                  ENDCASE;
          'E => BreakPoint[getproccondition[CR],
           (IF conditionfound THEN condition ELSE NIL), trace, set, entry];
          'X => BreakPoint[getproccondition[CR],
           (IF conditionfound THEN condition ELSE NIL), trace, set, exit];
          'M => TextBreakPoint[getprogcondition[], getsource[],
                (IF conditionfound THEN condition ELSE NIL),
                trace, prog];
          'P => TextBreakPoint[getproccondition[NUL], getsource[],
                (IF conditionfound THEN condition ELSE NIL),
                trace, proc];
          ENDCASE;
    'O => SELECT ReadUChar[read, clearBreak] FROM
          'R => ReadOctal[];
          'W => WriteOctal[];
          'C => OctalBreakPoint[getgframe[NUL], getbytepc[], clear];
          'S => OctalBreakPoint[getgframe[NUL], getbytepc[], set];
        ENDCASE;
    'I => SELECT ReadUChar[call, string] FROM
          'C => BEGIN CheckWorry[]; Icall[getparam[proc,NUL]] END;
          '@ => Iaddress[getcolonid[NUL, varname]];
          'P => Ipointer[getcolonoctal[], getparam[type,NUL]];
          'A => Iarray[getidplus[array], arrayindex, arraycount];
          'D => Ideref[getcolonid[NUL, varname]];
          'E => Iexpression[];
          'S => Istring[getidplus[string], stringindex, stringcount];
          ENDCASE;
    'Q => BEGIN CheckWorry[]; confirm[]; SIGNAL Quit; END;
    'U => BEGIN confirm[]; CoreSwap[showscreen]; END;
    'W => WorrySwitch[];
    'A => SELECT ReadUChar[ascii, attach] FROM
          'S => AsciiRead[];
          'T => SELECT ReadUChar[attachI, attachS] FROM
                  'I => AttachImageFile[getcolonid[CR, imagename]];
                  'S => AttachSymbols[getgframe[NUL],
                        getparam[file, CR]];
                  ENDCASE;
          ENDCASE;
    'F => DumpVar[getcolonid[CR, varname], config];
    IODefs.ControlU => BEGIN confirm[]; UserProc[]; END;
    IODefs.ControlF => BEGIN confirm[]; DebugFTPDefs.CallFTP[]; END;
    IODefs.ControlD => BEGIN confirm[]; CallDebugger[]; END;
    '- => IgnoreComment[];
    SP => BEGIN IODefs.ReadLine[expression]; Interpreter[expression]; END;
    'K => BEGIN confirm[]; SIGNAL DebugUtilityDefs.KillSession END;
    ENDCASE;
  RETURN
  END;

CallDebugger: PROCEDURE = MACHINE CODE
  BEGIN Mopcodes.zKFCB, SDDefs.sCallDebugger END;

CheckWorry: PROCEDURE =
  BEGIN
  IF ~DDptr.worryentry THEN RETURN;
  CommandDefs.WriteErrorString[naworry];
  SIGNAL abort;
  RETURN
  END;

confirm: PUBLIC PROCEDURE =
  BEGIN
  CommandDefs.WriteIDString[confirm];
  IF inchar[] # IODefs.CR THEN DO
    [] ← inchar[]; IODefs.WriteChar['?] ENDLOOP;
  DebugMiscDefs.WriteEOL[];
  RETURN
  END;

inchar: PROCEDURE RETURNS [c: CHARACTER] =
  BEGIN OPEN IODefs;
  IF (c ← ReadChar[]) = DEL THEN SIGNAL Rubout;
  RETURN
  END;

GetChar: PROCEDURE RETURNS [CHARACTER];

FirstChar: PROCEDURE RETURNS [c: CHARACTER] =
  BEGIN
  index ← 0;
  IF (c ← inchar[]) = IODefs.ESC THEN
    BEGIN GetChar ← RepeatChars; RETURN[command[0]]; END;
  GetChar ← NewChars;
  RETURN[command[0] ← c]
  END;

RepeatChars: PROCEDURE RETURNS [CHARACTER] =
  BEGIN
  RETURN[command[index ← index+1]]
  END;

NewChars: PROCEDURE RETURNS [c: CHARACTER] =
  BEGIN
  c ← inchar[];
  RETURN[command[index ← index+1] ← c]
  END;

command: STRING ← [6];
index: CARDINAL;

ReadUChar: PROCEDURE [beginning, last: CommandDefs.CommandName]
  RETURNS [c: CHARACTER] =
  BEGIN OPEN CommandDefs, IODefs;
  i: CommandName;
  ssd: StringDefs.SubStringDescriptor;
  ss: StringDefs.SubString ← @ssd;
  c ← GetChar[];
  IF c IN ['a..'z] THEN c ← c - 40B;
  IF c = '? THEN
    BEGIN
    WriteChar[c];
    typeoptions[beginning, last];
    SIGNAL abort;
    END;
  FOR i IN CommandName[beginning..last] DO
    GetCommandString[i,ss];
    IF ss.base[ss.offset] = c THEN
      BEGIN WriteCommandString[i]; StreamDefs.ResetControlDEL[]; RETURN END;
    ENDLOOP;
  WriteChar[c];  WriteChar['?];
  SIGNAL abort;
  RETURN
  END;

typeoptions: PROCEDURE [beginning, last: CommandDefs.CommandName] =
  BEGIN OPEN CommandDefs;
  i: CommandName;
  DebugMiscDefs.WriteEOL[];
  WriteErrorString[options];
  FOR i IN CommandName[beginning..last) DO
    WriteCommandString[i];
    IODefs.WriteString[", "L];
    ENDLOOP;
  WriteCommandString[last];
  DebugMiscDefs.WriteEOL[]; WriteErrorString[retry];
  RETURN
  END;
```

```
expression: STRING ← [100];
condition: STRING ← [100];
conditionfound: BOOLEAN ← FALSE;
attachname: STRING ← [40];
imagename: STRING ← [40];
varname: STRING ← [40];
procname: STRING ← [40];
modulename: STRING ← [40];
arrayname: STRING ← [40];
typename: STRING ← [40];
sourceline: STRING ← [60];
strname:  STRING ← [40];
rconfigname:  STRING ← [40];
configname: STRING ← [40];
lastoctal, lastlframe, lastframe, lastgframe, lastpc: UNSPECIFIED;

SysProc1: PROCEDURE [v: UNSPECIFIED] =
  BEGIN
  CommandDefs.WriteIDString[dashes];
  IODefs.WriteOctal[v];
  RETURN
  END;

DReadNumber: PUBLIC PROCEDURE [default: UNSPECIFIED, radix: CARDINAL]
  RETURNS [UNSPECIFIED] =
  BEGIN OPEN InlineDefs;
  s: STRING ← [60];
  c: ARRAY [0..6) OF [0..9];
  cp, i: CARDINAL ← 0;
  IF radix = 10 AND LOOPHOLE[default, INTEGER] < 0 THEN
    BEGIN default ← -default; s[0] ← '-; cp ← 1 END;
  DO
    [default,c[i]] ← LDIVMOD[default,0,radix];
    IF default = 0 THEN EXIT;
    i ← i + 1;
    ENDLOOP;
  FOR i DECREASING IN [0..i] DO
    s[cp] ← LOOPHOLE[c[i] + LOOPHOLE['0, INTEGER], CHARACTER];
    cp ← cp + 1;
   ENDLOOP;
  IF radix = 8 THEN
    BEGIN s[cp] ← 'B; cp ← cp + 1 END;
  s.length ← cp;
  IODefs.ReadID[s];
  RETURN[StringExpressionToNumber[s,radix]];
  END;

StringExpressionToNumber: PUBLIC PROCEDURE [s: STRING, defradix: CARDINAL]
  RETURNS [v:UNSPECIFIED] =
  BEGIN
  lv: DebuggerDefs.LA ← [LI[StringExpressionToLongNumber[s,defradix]]];
  RETURN[lv.low]
  END;

StringExpressionToLongNumber: PROCEDURE [s: STRING, defradix: CARDINAL]
  RETURNS [v: LONG INTEGER] =
  BEGIN OPEN InlineDefs;
  char, lastop: CHARACTER;
  cp: CARDINAL ← 0;
  radix: CARDINAL;
  v8, v10, number: LONG INTEGER;
  endofstring: BOOLEAN ← FALSE;
  getchar: PROCEDURE RETURNS [CHARACTER] =
    BEGIN
    char ← s[cp];
    IF (cp ← cp+1) > s.length THEN char ← NUL;
    RETURN[char];
    END;
  digits: ARRAY CHARACTER['0..'9] OF CARDINAL = [0,1,2,3,4,5,6,7,8,9];

  v ← number ← 0; lastop ← '+;
  UNTIL endofstring DO
    v8 ← v10 ← 0;
    radix ← defradix;
    DO
```

```
      SELECT getchar[] FROM
        IN ['0..'9] =>
          BEGIN
          v8 ← v8*8 + digits[char];
          v10 ← v10*10 + digits[char];
          number ← IF radix = 8 THEN v8 ELSE v10;
          END;
        'b,'B => BEGIN number ← v8; radix ← 8; GOTO exponent END;
        'd,'D => BEGIN number ← v10; radix ← 10; GOTO exponent END;
        '+,'-,'*,'/ => GOTO operation;
        NUL => BEGIN endofstring ← TRUE; GOTO operation END;
        <= '  => NULL;
        ENDCASE => SIGNAL StringDefs.InvalidNumber;
      REPEAT
        operation =>
          BEGIN
          SELECT lastop FROM
            '+ => v ← v + number;
            '- => v ← v - number;
            '* => v ← v * number;
            '/ => v ← v / number;
            ENDCASE;
          lastop ← char;
          END;
        exponent =>
          BEGIN
          li: LONG INTEGER;
          v10 ← 0;
          WHILE getchar[] IN ['0..'9] DO
            v10 ← v10*10 + digits[char];
            ENDLOOP;
          cp ← cp-1; -- took one too many
          FOR li ← 1, li+1 UNTIL li > v10 DO
            number ← number*radix;
            ENDLOOP;
          END;
      ENDLOOP;
    ENDLOOP;
  END;

getsource: PROCEDURE RETURNS [STRING] =
  BEGIN
  s: STRING ← [60];
  CopyString[s, sourceline];
  CommandDefs.WriteIDString[source];
  IODefs.ReadLine[s];
  CopyString[sourceline, s];
  RETURN[sourceline]
  END;

CommandAbort: PUBLIC SIGNAL = CODE;

getcolonid: PROCEDURE [c: CHARACTER, s: STRING] RETURNS [STRING] =
  BEGIN
  IODefs.WriteChar[':]; IODefs.WriteChar[' ];
  RETURN[getIDcheck[c,s]]
  END;

getcolonoctal: PROCEDURE RETURNS [n: UNSPECIFIED] =
  BEGIN
  IODefs.WriteChar[':]; IODefs.WriteChar[' ];
  n ← lastoctal ← DReadNumber[lastoctal,8];
  RETURN
  END;

arraycount, arrayindex: CARDINAL ← 0;
stringcount, stringindex: CARDINAL ← 0;
AS: TYPE = {array, string};

getidplus: PROCEDURE [type: AS] RETURNS [STRING] =
  BEGIN
  s: STRING ← IF type = array THEN arrayname ELSE strname;
  s ← getcolonid[NUL,s];
  IF type = array
    THEN [arrayindex, arraycount] ← getindexcount[arrayindex, arraycount]
    ELSE [stringindex, stringcount] ← getindexcount[stringindex, stringcount];
  IODefs.WriteChar[CR];
  RETURN[s];
  END;

getindexcount: PROCEDURE [index, count: CARDINAL] RETURNS [CARDINAL, CARDINAL] =
  BEGIN
  CommandDefs.WriteIDString[start];
  index ← DReadNumber[count + index,10];
  CommandDefs.WriteIDString[num];
  count ← DReadNumber[count, 10];
  RETURN[index, count];
  END;

getframe: PROCEDURE [c: CHARACTER] RETURNS [n: FrameHandle] =
  BEGIN
  IF c = ': THEN IODefs.WriteString[": "L]
  ELSE CommandDefs.WriteIDString[frame];
  n ← DReadNumber[lastlframe,8];
  IF ~DebugUtilityDefs.CheckFrame[n] THEN
    BEGIN CommandDefs.WriteErrorString[notframe]; SIGNAL abort END;
  lastlframe ← n;
  IODefs.WriteChar[CR];
  RETURN
  END;

getgframe: PROCEDURE [c: CHARACTER] RETURNS [g: GlobalFrameHandle] =
  BEGIN
  CommandDefs.WriteIDString[gframe];
  g ← DReadNumber[lastgframe,8];
  IF ~DebugUtilityDefs.ValidGlobalFrame[g] THEN
    BEGIN CommandDefs.WriteErrorString[notgframe]; SIGNAL abort END;
  lastgframe ← g;
  DebugMiscDefs.WriteCharZ[c];
  RETURN
  END;

geteitherframe: PROCEDURE [c: CHARACTER] RETURNS [f: UNSPECIFIED] =
  BEGIN
  IF c = ': THEN IODefs.WriteString[": "L]
  ELSE CommandDefs.WriteIDString[frame];
  f ← DReadNumber[lastframe,8];
  IF DebugUtilityDefs.CheckFrame[f] OR DebugUtilityDefs.ValidGlobalFrame[f]
    THEN lastframe ← f
  ELSE BEGIN CommandDefs.WriteErrorString[notframe]; SIGNAL abort END;
  IODefs.WriteChar[CR];
  RETURN
  END;

getbytepc: PROCEDURE RETURNS [n: UNSPECIFIED] =
  BEGIN
  CommandDefs.WriteIDString[bytepc];
  lastpc ← n ← DReadNumber[lastpc,8];
  IODefs.WriteChar[CR];
  RETURN
  END;

getmodule: PROCEDURE [c: CHARACTER] RETURNS [STRING] =
  BEGIN OPEN CommandDefs;
  temp: STRING ← [40];
  WriteIDString[mod];
  CopyString[temp,modulename];
  IODefs.ReadID[temp];
  DebugMiscDefs.WriteCharZ[c];
  CopyString[modulename,temp];
  RETURN[modulename]
  END;

getparam: PROCEDURE [name: CommandDefs.IDCode, c: CHARACTER]
  RETURNS [STRING] =
  BEGIN OPEN CommandDefs;
  WriteIDString[name];
  SELECT name FROM
    proc => RETURN[getIDcheck[c, procname]];
    type => RETURN[getIDcheck[c, typename]];
    file => RETURN[getIDcheck[c, attachname]];
    ENDCASE => ERROR;
  END;

getIDcheck: PROCEDURE [c: CHARACTER, s: STRING] RETURNS [STRING] =
  BEGIN
  temp: STRING ← [40];
  CopyString[temp,s];
  IODefs.ReadID[temp];
  IF temp[0] ~IN ['A..'Z] AND temp[0] ~IN['a..'z] THEN
    BEGIN CommandDefs.WriteErrorString[invalidID]; SIGNAL abort END;
  DebugMiscDefs.WriteCharZ[c];
  CopyString[s,temp];
  RETURN[s]
  END;

idfound: PROCEDURE [c: CHARACTER] RETURNS [BOOLEAN] =
  BEGIN RETURN[c = IODefs.CR OR c = IODefs.SP] END;

crfound: PROCEDURE [c: CHARACTER] RETURNS [BOOLEAN] =
  BEGIN RETURN[c = IODefs.CR] END;

getproccondition: PROCEDURE [c: CHARACTER] RETURNS [STRING] =
  BEGIN OPEN CommandDefs;
  IF c # NUL THEN WriteIDString[proc]
  ELSE IODefs.WriteString[": "L];
  IF IODefs.ReadEditedString[procname, idfound, TRUE] = SP THEN
    BEGIN --terminate by SP means go on, CR means done
    WriteIDString[condition];
    --condition terminated by CR
    [] ← IODefs.ReadEditedString[condition, crfound, TRUE];
    conditionfound ← TRUE;
    END
  ELSE conditionfound ← FALSE;
  DebugMiscDefs.WriteCharZ[c];
  RETURN[procname]
  END;

getprogcondition: PROCEDURE RETURNS [STRING] =
  BEGIN OPEN CommandDefs;
  IODefs.WriteChar[':]; IODefs.WriteChar[' ];
  IF IODefs.ReadEditedString[modulename, idfound, TRUE] = SP THEN
    BEGIN --terminate by SP means go on, CR means done
    WriteIDString[condition];
    IODefs.ReadLine[condition]; --terminated by CR
    conditionfound ← TRUE;
    END
  ELSE conditionfound ← FALSE;
  RETURN[modulename]
  END;

queuename: STRING ← [40];
processname: STRING ← [40];

getID: PROCEDURE [c: CHARACTER, s: STRING] RETURNS [STRING] =
  BEGIN
  temp: STRING ← [40];
  CopyString[temp, s];
  IODefs.WriteString[": "L];
  IODefs.ReadID[s];
  DebugMiscDefs.WriteCharZ[c];
  CopyString[temp, s];
  RETURN[s]
  END;

CopyString: PROCEDURE [to: STRING, from: STRING]=
  BEGIN
  to.length ← 0;
  StringDefs.AppendString[to, from];
  RETURN
  END;

WorrySwitch: PROCEDURE =
  BEGIN OPEN DebugUtilityDefs, SDDefs, CommandDefs;
  oldsBRK: UNSPECIFIED;
  IF DDptr.worrybreaks THEN WriteIDString[off] ELSE WriteIDString[on];
  confirm[];
  DDptr.worrybreaks ← ~DDptr.worrybreaks;
  oldsBRK ← MREAD[SD+sBreak];
  MWRITE[SD+sBreak, MREAD[SD+sAlternateBreak]];
  MWRITE[SD+sAlternateBreak, oldsBRK];
  RETURN
  END;

CaseSwitch: PROCEDURE =
  BEGIN
  IF DDptr.caseignoring THEN CommandDefs.WriteIDString[on]
    ELSE CommandDefs.WriteIDString[off];
  confirm[];
  DDptr.caseignoring ← ~DDptr.caseignoring;
  RETURN
  END;

LA: TYPE = DebuggerDefs.LA;

raddress, waddress: LA ← LA[LA[low:0, high:0]];
rcount: CARDINAL ← 0;

ReadOctal: PROCEDURE =
  BEGIN OPEN IODefs;
  j:  CARDINAL;
  n: INTEGER;
  i: INTEGER ← -1;

  CommandDefs.WriteIDString[addr];
  waddress ← raddress ← DReadLongAddress[LA[LI[li:raddress.li+rcount]],8];
  CommandDefs.WriteIDString[num];
  rcount ← DReadNumber[rcount,10];
  FOR j IN [0..rcount) DO
    IF StreamDefs.ControlDELtyped[] THEN SIGNAL DebugMiscDefs.ControlDEL;
    IF (i ← i+1) MOD 8 = 0 THEN
      BEGIN
      DebugMiscDefs.WriteEOL[];
      DWriteLongAddress[LA[LI[li:raddress.li+j]], 8];
      WriteChar['/]
      END;
    WriteChar[' ];
    WriteNumber[n ← DebugUtilityDefs.LongREAD[raddress.lp+j], NumberFormat[8,FALSE,TRUE,6]];
    WriteChar[IF n ~IN[0..7] THEN 'B ELSE ' ];
    ENDLOOP;
  RETURN
  END;

WriteOctal: PROCEDURE =
  BEGIN OPEN DebugUtilityDefs;
  CommandDefs.WriteIDString[addr];
  waddress ← DReadLongAddress[waddress,8];
  CommandDefs.WriteIDString[gets];
  LongWRITE[waddress.lp, DReadNumber[LongREAD[waddress.lp],8]];
  waddress.li ← waddress.li+1;
  RETURN
  END;

DReadLongNumber: PROCEDURE [default: LONG INTEGER, radix: CARDINAL]
  RETURNS [LONG INTEGER] =
  BEGIN
  longAddress: STRING ← [30];
  DAppendLongNumber[longAddress, default, radix];
  IODefs.ReadID[longAddress];
  RETURN[DStringToLongNumber[longAddress, radix]]
  END;

DWriteLongInteger: PUBLIC PROCEDURE [number: LONG INTEGER, radix: CARDINAL] =
  BEGIN
  longAddress: STRING ← [30];
  DAppendLongNumber[longAddress, number, radix];
  IODefs.WriteString[longAddress];
  RETURN
  END;

DReadLongAddress: PROCEDURE [default: LA, radix: CARDINAL]
  RETURNS [LA] = LOOPHOLE[DReadLongNumber];

DWriteLongAddress: PROCEDURE [number: LA, radix: CARDINAL] =
  LOOPHOLE[DWriteLongInteger];

DWriteLongPointer: PUBLIC PROCEDURE [number: LONG POINTER, radix: CARDINAL] =
  LOOPHOLE[DWriteLongInteger];

DAppendLongNumber: PROCEDURE [s: STRING, number: LONG INTEGER, radix: CARDINAL] =
  BEGIN OPEN StringDefs; --to check for overflow
  IF number = FIRST[LONG INTEGER] THEN
    IF radix = 8 THEN
      BEGIN AppendString[s, "20000000000B"L]; RETURN END
    ELSE AppendString[s, "-2147483648"L]
  ELSE AppendLongNumber[s, number, radix];
  IF radix = 8 THEN AppendChar[s, 'B];
  RETURN
  END;

DStringToLongNumber: PUBLIC PROCEDURE [s: STRING, radix: CARDINAL]
  RETURNS [LONG INTEGER] =
  BEGIN OPEN StringDefs;  --to check for overflow
  IF (EquivalentString[s,"20000000000B"L] AND radix = 8) OR
   (EquivalentString[s,"-2147483648"L] AND radix = 10)
   THEN RETURN[FIRST[LONG INTEGER]];
  RETURN[StringExpressionToLongNumber[s, radix]]
  END;

asciiaddress: LA ← LA[LA[low: 0, high: 0]];
acount: CARDINAL ← 0;

AsciiRead: PROCEDURE =
  BEGIN
  i: CARDINAL;
  s: PACKED ARRAY [0..1] OF CHARACTER;
  p: POINTER = @s;
  CommandDefs.WriteIDString[addr];
  asciiaddress ← DReadLongAddress[LA[LI[li:asciiaddress.li+acount/2]],8];
  CommandDefs.WriteIDString[num];
  acount ← DReadNumber[acount,10];
  DebugMiscDefs.WriteEOL[];
  FOR i IN [0..acount) DO
    IF i MOD 2 = 0 THEN
      p↑ ← DebugUtilityDefs.LongREAD[asciiaddress.lp+i/2];
    IODefs.WriteChar[s[i MOD 2]];
    ENDLOOP;
  RETURN
  END;

table: POINTER;

PrintExp: PUBLIC PROCEDURE [esp: DIDefs.ESPointer] =
  BEGIN OPEN IODefs, DebuggerDefs;
  --check for fakes, then interface to old debugger Display
  s: STRING;
  so: SymbolObject;
  sop: SOPointer ← @so;
  fr: FormatRecord;
  frp: FRPointer ← @fr;
  SELECT TRUE FROM
    esp.intN                 => --ignore interval, already printed
      NULL;
    esp.desc                 => --print as descriptor
      WITH esp SELECT FROM
        here =>
          BEGIN
          WriteString["DESCRIPTOR[base: "L];
          WriteOctal[(ptr)↑];
          WriteString["↑, length: "L];
          WriteOctal[(ptr+1)↑];
          WriteString["]  "L];
          SystemDefs.FreeHeapNode[ptr];
          END;
        ENDCASE => ERROR;
    (esp.indirection # 0)   => --print as pointer
      WITH e:esp SELECT FROM
        here => BEGIN WriteOctal[e.value]; WriteChar['↑]; END;
        there =>
          BEGIN
          WITH e SELECT FROM
            short => WriteOctal[DebugUtilityDefs.MREAD[shortAddr]];
            long => WriteOctal[DebugUtilityDefs.LongREAD[longAddr.lp]];
            ENDCASE;
          WriteChar['↑];
          END;
        ENDCASE => ERROR;
    (DITypeDefs.TypeString[esp] AND esp.tag = here)     => --print here strings
      WITH esp SELECT FROM
        here =>
          BEGIN OPEN ss: LOOPHOLE[value, StringDefs.SubString];
          s ← DebugMiscDefs.DGetString[ss.length];
          StringDefs.AppendSubString[s,@ss];
          WriteString[s];
          DebugMiscDefs.DFreeString[s];
          END;
        ENDCASE => ERROR;
    (esp.stbase = NIL)      => --print predefined types
      BEGIN
      p: DIDefs.hereESPointer ← DIActionDefs.Transfer[esp];
      SELECT p.tsei FROM
        DITypeDefs.SeiInteger => WriteDecimal[p.value];
        DITypeDefs.SeiCardinal => WriteOctal[p.value];
        DITypeDefs.SeiCharacter => DebuggerDefs.DumpCharacter[p.value];
        DITypeDefs.SeiBoolean => WriteString[
          IF LOOPHOLE[p.value, BOOLEAN] THEN "TRUE"L ELSE "FALSE"L];
        DITypeDefs.SeiUnspecified => WriteOctal[p.value];
        DITypeDefs.SeiLongInteger =>
          BEGIN num: LONG INTEGER;
          num ← LOOPHOLE[p.ptr, POINTER TO LONG INTEGER]↑;
          DWriteLongInteger[num, 10];
          END;
        ENDCASE => ERROR;
      END;
    ENDCASE                 =>
      BEGIN
      fr ← [indentation: 2, symdelim: '=, intersym: CR, startchar: NUL,
        termchar: NUL, symid: TRUE, firstsym: TRUE];
      DIActionDefs.espTosop[esp,sop];
      DebuggerDefs.Display[sop,frp,TRUE];
      END;
  DIActionDefs.FreeStackItem[esp];
  RETURN
  END;

Interpreter: PROCEDURE [s: STRING] =
  BEGIN OPEN DIActionDefs, DIDefs, DITypeDefs, IODefs, CommandDefs;
  IF s.length = 0 THEN RETURN;
  InterpretString[s, PrintExp, FALSE
    ! DebugMiscDefs.LookupFail =>
        BEGIN WriteChar['!]; WriteString[s]; CONTINUE END;
      DebugContextDefs.IncorrectVersion => RESUME;
      TypesDontMatch, IncorrectType, InvalidType =>
        BEGIN WriteIString[type]; CONTINUE END;
      EvalStackOverflow, EvalStackEmpty, NILesp, NotOnEvalStack,
      TypeStackOverflow, TypeStackEmpty, InvalidInterval,
      InvalidExpression, SyntaxError, ParseError =>
        BEGIN WriteIString[exp]; CONTINUE END;
      NotImplemented =>
        BEGIN WriteIString[ni]; CONTINUE END;
      InvalidCharacter =>
        BEGIN WriteIString[char]; WriteOctal[index];
        WriteIString[bracket]; CONTINUE END;
      InvalidNumber =>
        BEGIN WriteIString[num]; WriteOctal[index];
        WriteIString[bracket]; CONTINUE END];
  DIActionDefs.CleanUp[];
  RETURN
  END;

InterpretString: PUBLIC PROCEDURE [s: STRING,
  proc: PROCEDURE[esp: DIDefs.ESPointer], locals: BOOLEAN] =
  BEGIN
  IF s.length = 0 THEN RETURN;
  DILitDefs.LitTabInit[];
  DIActionDefs.GetSetUp[];
  [] ← DIDefs.Parse[s, table, DIDefs.TwoParse[], proc, locals];
  DIActionDefs.CleanUp[];
  RETURN
  END;

Init: PROCEDURE =
  BEGIN
  f: GlobalFrameHandle ← LOOPHOLE[BinaryDefs.DIGrammar];
  table ← f.code.codebase;
  [] ← MiscDefs.DestroyFakeModule[f];
  RETURN
  END;

Init[];

END...
```