

```
-- Commander.mesa; edited by Sandman, May 5, 1978 11:40 AM
```

```
DIRECTORY
```

```
CommanderDefs: FROM "commanderdefs",
ControlDefs: FROM "controldefs",
FrameDefs: FROM "framedefs",
InlineDefs: FROM "inlinedefs",
ImageDefs: FROM "imagedefs",
IODefs: FROM "iodefs",
MiscDefs: FROM "miscdefs",
RectangleDefs: FROM "rectangledefs",
SegmentDefs: FROM "segmentdefs",
StreamDefs: FROM "streamdefs",
StringDefs: FROM "stringdefs",
SystemDefs: FROM "systemdefs";
```

```
DEFINITIONS FROM StringDefs, CommanderDefs, IODefs;
```

```
Commander: PROGRAM [herald: STRING]
  IMPORTS ImageDefs, IODefs, StreamDefs, StringDefs, SystemDefs
  EXPORTS CommanderDefs =
BEGIN
```

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

```
CommandItem: TYPE = RECORD [
  cb: CommandBlockHandle,
  link: POINTER TO CommandItem];
```

```
StringItem: TYPE = RECORD [
  link: POINTER TO StringItem,
  string: STRING];
```

```
commandHead: POINTER TO CommandItem ← NIL;
stringHead: POINTER TO StringItem ← NIL;
```

```
SyntaxError: ERROR = CODE;
Help: SIGNAL = CODE;
BadName: ERROR = CODE;
BadParam: ERROR [type: ParamType] = CODE;
```

```
GetDebugger: PROCEDURE =
  BEGIN
  MiscDefs.CallDebugger[NIL];
  END;
```

```
Load: PROCEDURE [name: STRING] =
  BEGIN OPEN StringDefs;
  module: STRING ← [40];
  frame: ControlDefs.GlobalFrameHandle;
  AppendString[module, name | StringBoundsFault => GOTO error];
  CheckForExtension[module, ".bcd"L];
  frame ← FrameDefs.New[module | ANY => GOTO error];
  START LOOPHOLE[frame, PROGRAM][ ! ANY => GOTO error];
  RETURN
  EXITS error => WriteString["Bad file!"L];
  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;
```

```
AddCommand: PUBLIC PROCEDURE [name: STRING, proc: PROCEDURE, numargs: CARDINAL] RETURNS [CommandBlockHa
**ndle] =
  BEGIN OPEN SystemDefs;
  c: POINTER TO CommandItem ← AllocateHeapNode[SIZE[CommandItem]];
  cb: CommandBlockHandle ←
    AllocateHeapNode[SIZE[CommandBlock]+numargs*SIZE[CommandParam]];
  c↑ ← CommandItem[cb, link: commandHead];
  commandHead ← c;
```

```

    cb.name ← name;
    cb.proc ← proc;
    cb.nparams ← numargs;
    RETURN[cb]
    END;

NewString: PROCEDURE [s: STRING] RETURNS [ns: STRING] =
    BEGIN
    si: POINTER TO StringItem ←
        SystemDefs.AllocateHeapNode[SIZE[StringItem]];
    ns ← SystemDefs.AllocateHeapString[s.length];
    si ← StringItem[link: stringHead, string: ns];
    stringHead ← si;
    AppendString[ns,s];
    RETURN
    END;

FreeStrings: PROCEDURE =
    BEGIN
    next: POINTER TO StringItem;
    WHILE stringHead # NIL DO
        next ← stringHead.link;
        SystemDefs.FreeHeapString[stringHead.string];
        SystemDefs.FreeHeapNode[stringHead];
        stringHead ← next;
    ENDLOOP;
    RETURN
    END;

WriteEOL: PROCEDURE =
    BEGIN
    IF ~NewLine[] THEN WriteChar[CR];
    RETURN
    END;

ComplementScreen: PROCEDURE =
    BEGIN OPEN RectangleDefs;
    dcbptr: DCBptr;
    CompBackGround: ARRAY backgtype OF backgtype = [black, white];

    FOR dcbptr ← MEMORY[420B], dcbptr.next UNTIL dcbptr = DCBnil DO
        dcbptr.background ← CompBackGround[dcbptr.background];
    ENDLOOP;
    RETURN
    END;

FlashScreen: PROCEDURE =
    BEGIN
    ComplementScreen[];
    THROUGH [1..10000] DO NULL ENDLOOP;          -- delay a while (38us per iteration)
    ComplementScreen[];
    RETURN
    END;

interactive: BOOLEAN;
commandStream: StreamDefs.StreamHandle;
comcmRequest: short ImageDefs.FileRequest ← [
    body: short[fill:, name: "Com.Cm."],
    file: NIL,
    access: SegmentDefs.Read,
    link: ];

CommanderCleanupItem: ImageDefs.CleanupItem ←
    [proc: CommanderCleanup, mask: ImageDefs.CleanupMask[Save], link: ];

CommanderCleanup: ImageDefs.CleanupProcedure =
    BEGIN
    SELECT why FROM
        Save => ImageDefs.AddFileRequest[@comcmRequest];
    ENDCASE;
    RETURN
    END;

SetCommandInput: PROCEDURE =
    BEGIN
    c: CHARACTER;

```

```
IF comcmRequest.file = NIL THEN commandStream ← NIL
ELSE
  BEGIN OPEN StreamDefs;
  commandStream ← CreateByteStream[comcmRequest.file, StreamDefs.Read];
  [] ← SkipStreamBlanks[];
  UNTIL commandStream.eof[commandStream] OR
    (c←commandStream.get[commandStream]) = SP OR c = CR DO NULL ENDLOOP;
  [] ← SkipStreamBlanks[];
  IF commandStream.eof[commandStream] THEN
    BEGIN commandStream.destroy[commandStream]; commandStream ← NIL; END
  ELSE SetIndex[commandStream,ModifyIndex[GetIndex[commandStream],-1]];
  END;
interactive ← commandStream = NIL;

RETURN
END;
```

```
-- code to get a line from the user, handling ESC and ?; stuffs it in line
```

```
line: STRING ← NIL;
lineTerminator: CHARACTER;
Lindex: CARDINAL;
```

```
AppendStringToLine: PROCEDURE[s: STRING] =
  BEGIN
    UNTIL (s.length+line.length) <= line.maxlength DO AddToLine[]; ENDLOOP;
    AppendString[line,s];
    RETURN
  END;
```

```
AppendCharToLine: PROCEDURE[c: CHARACTER] =
  BEGIN
    IF line.length = line.maxlength THEN AddToLine[];
    AppendChar[line,c];
    RETURN
  END;
```

```
ReadUserLine: PROCEDURE[newstring: BOOLEAN] =
  BEGIN -- read line from user; also handles <ESC> and '?' for input from user
    IF line = NIL THEN line ← SystemDefs.AllocateHeapString[80];
    [] ← ReadEditedString[line, LineMonitor, newstring
      | resume => BEGIN newstring ← FALSE; RETRY END];
    Lindex ← 0;
    RETURN
  END;
```

```
resume: SIGNAL = CODE;
```

```
LineMonitor: PROCEDURE [c: CHARACTER] RETURNS [BOOLEAN] =
  BEGIN
    SELECT c FROM
      CR => RETURN[TRUE];
      '?' =>
        BEGIN
          WriteChar['?'];
          IF line.length = 0 THEN SIGNAL Help;
          PromptCompletions[]; SIGNAL resume
        END;
      ESC => BEGIN ExtendLine[]; SIGNAL resume END;
    ENDCASE => RETURN[FALSE]
  END;
```

```
PromptCompletions: PROCEDURE =
  BEGIN
    id: STRING = [40];
    atLeastOne: BOOLEAN ← FALSE;
    p: POINTER TO CommandItem;
    IF GetLastID[id] THEN
      FOR p ← commandHead, p.link UNTIL p = NIL DO
        IF PrefixString[prefix: id, of: p.cb.name] THEN
          BEGIN
            IF ~atLeastOne THEN WriteEOL[];
            WriteChar[SP]; WriteChar[SP]; WriteString[p.cb.name];
            atLeastOne ← TRUE;
          END;
        ENDLOOP;
      IF atLeastOne THEN ReTypeLine[] ELSE FlashScreen[];
    RETURN
  END;
```

```
ExtendLine: PROCEDURE =
  BEGIN
    i: CARDINAL;
    id: STRING = [40];
    match: STRING = [40];
    moreThanOne, atLeastOne: BOOLEAN ← FALSE;
    p: POINTER TO CommandItem;

    IF GetLastID[id] THEN
      BEGIN
```

```

FOR p ← commandHead, p.link UNTIL p = NIL DO
  IF PrefixString[prefix: id, of: p.cb.name] THEN
    IF ~atLeastOne THEN
      BEGIN AppendString[match, p.cb.name]; atLeastOne ← TRUE; END
    ELSE BEGIN AndString[match, p.cb.name]; moreThanOne ← TRUE; END;
  ENDOLOOP;
END;
IF atLeastOne AND id.length # match.length THEN
  BEGIN
  FOR i IN [id.length..match.length) DO
    AppendCharToLine[match[i]];
    WriteChar[match[i]];
  ENDOLOOP;
  IF moreThanOne THEN FlashScreen[];
  END
ELSE FlashScreen[];
RETURN
END;

```

```

PrefixString: PROCEDURE [prefix, of: STRING] RETURNS [BOOLEAN] =
  BEGIN
  i: CARDINAL;

  IF prefix.length > of.length THEN RETURN[FALSE];
  FOR i IN [0..prefix.length) DO
    IF ~EquivalentChar[prefix[i], of[i]] THEN RETURN[FALSE];
  ENDOLOOP;
  RETURN[TRUE]
  END;

```

```

AndString: PROCEDURE [accum, s: STRING] =
  BEGIN
  i: CARDINAL;

  FOR i IN [0..s.length) DO
    IF ~EquivalentChar[accum[i], s[i]] THEN
      BEGIN accum.length ← i; RETURN END;
    ENDOLOOP;
  accum.length ← s.length;
  RETURN;
  END;

```

```

GetLastID: PROCEDURE [id: STRING] RETURNS [BOOLEAN] =
  BEGIN
  i, start: CARDINAL;
  c: CHARACTER;

  IF line.length = 0 THEN RETURN[FALSE];
  start ← line.length;
  FOR i DECREASING IN [0..line.length) DO
    IF AlphaNumeric[c ← line[i]] THEN start ← i ELSE
      IF c = ']' OR c = SP THEN EXIT
    ELSE RETURN[FALSE];
  ENDOLOOP;
  FOR i IN [start..line.length) DO id[i-start] ← line[i] ENDOLOOP;
  id.length ← line.length-start;
  RETURN[id.length # 0]
  END;

```

```

AlphaNumeric: PROCEDURE [c: CHARACTER] RETURNS [BOOLEAN] =
  BEGIN OPEN InlineDefs;
  RETURN[Alphabetic[c] OR Digit[c]]
  END;

```

```

Alphabetic: PROCEDURE [c: CHARACTER] RETURNS [BOOLEAN] =
  BEGIN
  RETURN[InlineDefs.BITAND[LOOPHOLE[c], 337B] IN [100B..132B]]
  END;

```

```

Digit: PROCEDURE [c: CHARACTER] RETURNS [BOOLEAN] =
  BEGIN
  RETURN[c IN ['0..'9]]
  END;

```

```
EquivalentChar: PROCEDURE [c,d: CHARACTER] RETURNS [BOOLEAN] =
  BEGIN OPEN InlineDefs;
  RETURN[BITOR[LOOPHOLE[c], 40B] = BITOR[LOOPHOLE[d], 40B]]
  END;
```

```
AddToLine: PROCEDURE =
  BEGIN
  newline: STRING ← SystemDefs.AllocateHeapString[line.maxlength+80];

  AppendString[to: newline, from: line];
  SystemDefs.FreeHeapString[line];
  line ← newline;
  RETURN
  END;
```

```
ReTypeLine: PROCEDURE =
  BEGIN
  WriteEOL[];
  WriteString[line];
  RETURN
  END;
```

```

-- code to handle characters

command: STRING = [100];
executing: BOOLEAN ← FALSE;
Cindex: CARDINAL;
currentChar: CHARACTER;

EndOfString: SIGNAL = CODE;
GetChar: PROCEDURE RETURNS [CHARACTER] ← GetCommandChar;
PutBackChar: PROCEDURE ← PutBackCommandChar;

GetCommandChar: PROCEDURE RETURNS [CHARACTER] =
  BEGIN
    IF Cindex >= command.length THEN currentChar ← NUL
    ELSE BEGIN currentChar ← command[Cindex]; Cindex ← Cindex+1; END;
    RETURN[currentChar]
  END;

PutBackCommandChar: PROCEDURE =
  BEGIN
    IF currentChar = NUL THEN RETURN;
    IF Cindex = 0 THEN ERROR;
    Cindex ← Cindex-1;
    RETURN
  END;

CommandOverFlow: SIGNAL = CODE;

SetUpCommand: PROCEDURE RETURNS [BOOLEAN] =
  BEGIN
    BEGIN ENABLE StringBoundsFault => SIGNAL CommandOverFlow;
    RETURN[IF interactive THEN CopyFromLine[] ELSE CopyFromStream[]];
  END
END;

CopyFromLine: PROCEDURE RETURNS[BOOLEAN] =
  BEGIN
    c: CHARACTER ← NUL;

    DO
      IF Lindex >= line.length THEN RETURN[FALSE];
      c ← line[Lindex];
      Lindex ← Lindex+1;
      IF c # SP AND c # CR THEN EXIT;
    ENDLOOP;
    command.length ← 0;
    DO
      AppendChar[command, c];
      IF c = ']' OR Lindex >= line.length THEN EXIT;
      c ← line[Lindex];
      Lindex ← Lindex+1;
    ENDLOOP;
    Cindex ← 0;
    RETURN [TRUE]
  END;

SkipStreamBlanks: PROCEDURE RETURNS [c: CHARACTER] =
  BEGIN
    UNTIL commandStream.eofof[commandStream] DO
      c ← commandStream.get[commandStream];
      IF c # SP AND c # CR THEN EXIT;
    ENDLOOP;
  END;

CopyFromStream: PROCEDURE RETURNS[BOOLEAN] =
  BEGIN
    c: CHARACTER;

    c ← SkipStreamBlanks[];
    IF commandStream.eofof[commandStream] THEN ImageDefs.StopMesa[];
    command.length ← 0;
    WriteEOL[];
    WriteChar['<']; WriteChar['>'];
    DO
      AppendChar[command, c];

```

```
    WriteChar[c];
    IF c = ']' OR commandStream.eof[commandStream] THEN EXIT;
    c ← commandStream.get[commandStream];
    ENDLOOP;
    WriteEOL[];
    Cindex ← 0;
    RETURN [TRUE]
END;

GetName: PROCEDURE [n: STRING] =
BEGIN
    n.length ← 0;
    DO
        IF AlphaNumeric[GetChar[]] THEN AppendChar[n, currentChar]
        ELSE EXIT;
        ENDLOOP;
    PutBackChar[]; SkipBlanks[]; IF GetChar[] # '[' THEN SE[];
    RETURN
END;

SkipBlanks: PROCEDURE =
BEGIN
    DO
        IF GetChar[] # SP THEN BEGIN PutBackChar[]; RETURN END;
        ENDLOOP
    END;
END;
```



```
-- code to parse user command
```

```
ParseCommand: PROCEDURE[state: POINTER TO ControlDefs.StateVector] =
  BEGIN
    proc: STRING = [40];
    cb: CommandBlockHandle;
    i: CARDINAL;

    GetName[proc];
    cb ← FindProc[proc].cb;
    FOR i IN [0..cb.nparams) DO
      state.stk[i] ← GetArg[cb,cb.params[i].type];
      IF GetChar[] # (IF i = cb.nparams-1 THEN ' ' ELSE ',') THEN SE[];
    ENDLOOP;
    state.dest ← cb.proc;
    state.stkptr ← cb.nparams;
    RETURN
  END;
```

```
FindProc: PROCEDURE [name: STRING] RETURNS [p: POINTER TO CommandItem] =
  BEGIN
    FOR p ← commandHead, p.link UNTIL p = NIL DO
      IF EquivalentString[name,p.cb.name] THEN RETURN;
    ENDLOOP;
    ERROR BadName;
  END;
```

```
GetArg: PROCEDURE[cb: CommandBlockHandle, t: ParamType] RETURNS [a: UNSPECIFIED] =
  BEGIN
    s: STRING = [100];

    SkipBlanks[];
    SELECT GetChar[] FROM
      '"' =>
        BEGIN
          IF t # string THEN ERROR BadParam[t];
          DO
            IF GetChar[] = '"' AND GetChar[] # ' ' THEN
              BEGIN PutBackChar[]; EXIT END;
            IF executing THEN AppendChar[s, currentChar];
          ENDLOOP;
          IF executing THEN a ← NewString[s];
        END;
      ''' =>
        BEGIN
          IF t # character THEN ERROR BadParam[t];
          a ← GetChar[];
        END;
      IN['0..'9], '(' , '-' =>
        BEGIN
          IF t # numeric THEN ERROR BadParam[t];
          PutBackChar[];
          a ← ExpressionToNumber[];
        END;
      'T' =>
        BEGIN
          IF t # boolean THEN ERROR BadParam[t];
          a ← GetTRUE[t];
        END;
      'F' =>
        BEGIN
          IF t # boolean THEN ERROR BadParam[t];
          a ← GetFALSE[t];
        END;
    ENDCASE => ERROR BadParam[t];
    SkipBlanks[];
    RETURN
  END;
```

```
GetTRUE: PROCEDURE [t: ParamType] RETURNS [BOOLEAN] =
  BEGIN
    IF GetChar[] # 'R THEN ERROR BadParam[t];
    IF GetChar[] # 'U THEN ERROR BadParam[t];
    IF GetChar[] # 'E THEN ERROR BadParam[t];
    RETURN[TRUE];
```

```
END;
```

```
GetFALSE: PROCEDURE [t: ParamType] RETURNS [BOOLEAN] =  
  BEGIN  
    IF GetChar[] # 'A THEN ERROR BadParam[t];  
    IF GetChar[] # 'L THEN ERROR BadParam[t];  
    IF GetChar[] # 'S THEN ERROR BadParam[t];  
    IF GetChar[] # 'E THEN ERROR BadParam[t];  
    RETURN[FALSE];  
  END;
```

```

-- code to parse user commands in interactive mode

ParsePromptedCommand: PROCEDURE =
  BEGIN
  proc: STRING = [40];
  cb: CommandBlockHandle;

  IF GetLastID[proc] THEN
    BEGIN
    cb ← FindProc[proc].cb;
    GetPromptedArgs[cb];
    Confirm[];
    RETURN
    END;
  lineTerminator ← CR;
  RETURN
  END;

CRFound: PROCEDURE [c: CHARACTER] RETURNS[BOOLEAN] =
  BEGIN RETURN[c = CR] END;

GetPromptedArgs: PROCEDURE[cb: CommandBlockHandle] =
  BEGIN
  i: CARDINAL;
  cindex: CARDINAL;
  cstring: STRING = [100];
  GetArgChar: PROCEDURE RETURNS [c: CHARACTER] =
    BEGIN
    IF cindex >= cstring.length THEN currentChar ← NUL
    ELSE BEGIN currentChar ← cstring[cindex]; cindex ← cindex+1; END;
    RETURN[currentChar]
    END;
  PutBackArgChar: PROCEDURE =
    BEGIN
    IF currentChar = NUL THEN RETURN;
    IF cindex = 0 THEN ERROR;
    cindex ← cindex-1;
    RETURN
    END;
  GetChar ← GetArgChar;
  PutBackChar ← PutBackArgChar;
  AppendCharToLine['[]];
  FOR i IN [0..cb.nparams) DO
    WriteString["
    "];
    WriteString[cb.params[i].prompt];
    WriteChar[':']; WriteChar[' '];
    [] ← ReadEditedString[cstring,CRFound,TRUE];
    cindex ← 0;
    [] ← GetArg[cb, cb.params[i].type];
    AppendStringToLine[cstring]; AppendCharToLine['.'];
    ENDLOOP;
  IF cb.nparams # 0 THEN line[line.length-1] ← ' ' ELSE AppendCharToLine[''];
  GetChar ← GetCommandChar;
  PutBackChar ← PutBackCommandChar;
  RETURN
  END;

Confirm: PROCEDURE =
  BEGIN
  char: CHARACTER;

  WriteString[" [confirm]"];
  DO
  char ← ReadChar[];
  SELECT char FROM
    DEL => SIGNAL Rubout;
    CR => BEGIN WriteEOL[]; EXIT END;
    SP => BEGIN WriteString["
    <>"]; EXIT END;
    ENDCASE => WriteChar['?'];
  ENDLOOP;
  lineTerminator ← char;
  RETURN

```

END;

```
-- parsing arithmetic expressions
```

```
symbol: Symbol;
Symbol: TYPE = RECORD[
  body: SELECT tag: * FROM
    delim => [char: CHARACTER],
    num => [val: INTEGER],
  ENDCASE];
Num: TYPE = num Symbol;

SE: PROCEDURE = BEGIN ERROR SyntaxError END;

Scan: PROCEDURE =
  BEGIN
    v8, v10, radix, number: CARDINAL;
    digits: ARRAY CHARACTER['0..'9] OF CARDINAL = [0,1,2,3,4,5,6,7,8,9];
    firstchar: BOOLEAN ← TRUE;

    v8 ← v10 ← 0;
    SkipBlanks[];
    DO
      SELECT GetChar[] FROM
        IN ['0..'9] =>
          BEGIN
            v8 ← v8*8 + digits[currentChar];
            v10 ← v10*10 + digits[currentChar];
            END;
          'M =>
            BEGIN
              IF ~firstchar THEN SE[];
              IF ~(GetChar[] = 'O AND GetChar[] = 'D) THEN SE[];
              IF ~Alphabetic[GetChar[]] THEN PutBackChar[] ELSE SE[];
              symbol ← [delim['!]];
              RETURN
            END;
          'b,'B => BEGIN number ← v8; radix ← 8; GOTO exponent END;
          'd,'D => BEGIN number ← v10; radix ← 10; GOTO exponent END;
          SP => GOTO done;
          NUL => IF ~firstchar THEN GOTO done
            ELSE BEGIN symbol ← nul; RETURN END;
          '(, '/', '*', '+, '-', ')', ')', ' ', =>
            IF firstchar THEN BEGIN symbol ← [delim[currentChar]]; RETURN END
            ELSE BEGIN PutBackChar[]; GOTO done END;
          ENDCASE => SIGNAL InvalidNumber;
      firstchar ← FALSE;
    REPEAT
      done => BEGIN symbol ← [num[v10]]; RETURN END;
      exponent =>
        BEGIN
          IF firstchar THEN SE[];
          v10 ← 0;
          WHILE Digit[GetChar[]] DO
            v10 ← v10*10 + digits[currentChar];
            REPEAT
              FINISHED => PutBackChar[]; -- took one too many
            ENDLOOP;
            THROUGH [1 .. v10] DO number ← number*radix ENDLOOP;
            symbol ← [num[number]];
            RETURN
          END;
        ENDLOOP;
    END;

    nul: Symbol = [delim[NUL]];

Primary: PROCEDURE RETURNS [n: Num] =
  BEGIN
    WITH s: symbol SELECT FROM
      delim =>
        BEGIN
          IF s.char # '(' THEN SE[];
          Scan[]; in ← Exp[];
          WITH symbol SELECT FROM
            delim =>
              IF char = ')' THEN BEGIN Scan[]; RETURN END;
```

```

        ENDCASE;
        SE[];
    END;
    num => BEGIN n ← s; Scan[]; RETURN END;
    ENDCASE
END;

Factor: PROCEDURE RETURNS [n: Num] =
BEGIN
    WITH symbol SELECT FROM
        delim =>
            IF char = '-' THEN
                BEGIN Scan[]; n ← Primary[]; n.val ← -n.val; RETURN END;
            ENDCASE;
    RETURN [Primary[]]
END;

Product: PROCEDURE RETURNS [n: Num] =
BEGIN
    x: Num;
    n ← Factor[];
    DO
        WITH symbol SELECT FROM
            delim =>
                SELECT char FROM
                    '*' => BEGIN Scan[]; n.val ← Factor[].val * n.val; END;
                    '/' => BEGIN Scan[]; x ← Factor[]; n.val ← n.val / x.val; END;
                    '!' => BEGIN Scan[]; x ← Factor[]; n.val ← n.val MOD x.val; END;
                ENDCASE => EXIT;
            ENDCASE => EXIT;
        ENDLOOP;
    RETURN
END;

Exp: PROCEDURE RETURNS [n: Num] =
BEGIN
    n ← Product[];
    DO
        WITH symbol SELECT FROM
            delim =>
                SELECT char FROM
                    '+' => BEGIN Scan[]; n.val ← Product[].val + n.val; END;
                    '-' => BEGIN Scan[]; n.val ← n.val - Product[].val; END;
                    '[', ',', => BEGIN PutBackChar[]; EXIT END;
                    NUL, ')' => EXIT;
                ENDCASE => SE[];
            ENDCASE => EXIT;
        ENDLOOP;
    RETURN
END;

ExpressionToNumber: PROCEDURE RETURNS [INTEGER] =
BEGIN
    Scan[];
    RETURN [Exp[].val]
END;

```

```

ShowSE: PROCEDURE =
  BEGIN
  IF ~executing THEN BEGIN WriteChar['?']; RETURN END;
  WriteEOL[];
  IF interactive THEN WriteString[command];
  WriteEOL[];
  THROUGH [1..(Cindex+(IF interactive THEN 0 ELSE 2))]
    DO WriteChar['.']; ENDOLOOP;
  WriteChar['↑'];
  RETURN
  END;

Driver: PROCEDURE =
  BEGIN
  state: ControlDefs.StateVector;
  newline: BOOLEAN;
  ci: POINTER TO CommandItem;
  i: CARDINAL;

  BEGIN
  ENABLE
  BEGIN
  SyntaxError, LineOverflow, InvalidNumber, StringBoundsFault =>
    BEGIN ShowSE[]; GO TO abort END;
  CommandOverflow =>
    BEGIN WriteEOL[]; WriteString["Command too long!"]; GO TO abort END;
  BadName =>
    BEGIN ShowSE[]; WriteString[" not found!"]; GO TO abort END;
  BadParam =>
    BEGIN
    ShowSE[];
    WriteString[" expected "];
    SELECT type FROM
      string => WriteString["string"];
      character => WriteString["character"];
      numeric => WriteString["numerical"];
    ENDCASE;
    WriteString[" parameter"]; GO TO abort
    END;
  Rubout => BEGIN WriteString[" XXX"]; GO TO abort END;
  Help =>
    BEGIN
    WriteEOL[];
    FOR ci ← commandHead, ci.link UNTIL ci = NIL DO
      WriteString[ci.cb.name];
      WriteChar[' '];
      FOR i IN[0..ci.cb.nparams) DO
        IF i # 0 THEN WriteChar['.'];
        SELECT ci.cb.params[i].type FROM
          string => WriteChar['"'];
          character => WriteChar[''];
        ENDCASE;
        WriteString[ci.cb.params[i].prompt];
        SELECT ci.cb.params[i].type FROM
          string => WriteChar['"'];
        ENDCASE;
      ENDOLOOP;
      WriteChar[' '];
      IF ci.link # NIL THEN
        BEGIN WriteChar['.']; WriteChar[' ']; END;
      ENDOLOOP;
    GO TO abort
    END;
  UNWIND => FreeStrings[]
  END;

  newline ← TRUE;
  executing ← FALSE;
  IF interactive THEN
    BEGIN
    WriteEOL[];
    WriteChar['<']; WriteChar['>'];
    DO
      ENABLE LineOverflow => BEGIN AddToLine[]; RESUME[line] END;
      ReadUserLine[newline];
    
```

```

        newline ← FALSE;
        ParsePromptedCommand[];
        IF lineTerminator = CR THEN EXIT;
        ENDOLOOP;
    END;
    GetChar ← GetCommandChar;
    PutBackChar ← PutBackCommandChar;
    executing ← TRUE;
    WHILE SetUpCommand[] DO
        ParseCommand[@state];
        state.instbyte ← 0;
        state.source ← REGISTER[Control]Defs.Lreg];
        TRANSFER WITH state;
        state ← STATE;
    ENDOLOOP;
    executing ← FALSE;
    EXITS abort => NULL;
    END;

    FreeStrings[];
    RETURN
    END;

Quit: PROCEDURE =
    BEGIN
        ImageDefs.StopMesa[];
    END;

-- main body
[] ← AddCommand["Quit",Quit,0];
[] ← AddCommand["Debug",GetDebugger,0];
AddCommand["Load",LOOPHOLE[Load],1].params[0] ← [string, "File name"];
ImageDefs.AddCleanupProcedure[@CommanderCleanupItem];
STOP; -- restart here when commander is started inside subsystem

WriteEOL[]; WriteString[herald]; WriteEOL[];
SetCommandInput[];

DO Driver[]; ENDOLOOP;

END.
-- Here is the grammar for the command line

CommandLine ::= PromptedCommandList <CR> | NonPromptedCommandList ;
                NonPromptedCommandList <EOF>
PromptedCommandList ::= PromptedCommand | Command | CommandList <SP> Command
                        | CommandList <SP> PromptedCommand
NonPromptedCommandList ::= Command | CommandList <SP> Command
Command ::= ID [ ParamList ]
PromptedCommand ::= ID <CR> PromptedParamList
ParamList ::= Param | ParamList , Param
PromptedParamList ::= Param | PromptedParamList <CR> Param
Param ::= " STRING " | ' CHARACTER | Expression | <empty>
Expression ::= Product | Expression + Product | Expression - Product
Product ::= Factor | Product * Factor | Product / Factor | Product MOD Factor
Factor ::= - Primary | Primary
Primary ::= NUM | ( Expression )

```