XEROX XEROX

on Last Jam 4 Sources . dm

(edited stream version using old Window Package)

Alto II/Orbit/Dover (Menlo) Press file printer

Spruce version 9.200 -- spooler version 9.200

File: jamtypechk.mesa

Creation date: 12-Jun-79 9:45:39

Name: Newell

22 total sheets = 21 pages, 1 copy.

XEROX XEROX

jamcontrol.mesa jamfns.mesa jamdictionary.mesa jamexec.mesa jamio.mesa jamliteral.mesa jamscanner.mesa
jamstack.mesa jamstart.mesa jamstring.mesa jamtypechk.mesa jamattributes.mesa jamvm.mesa jammath.mesa
jaminterrupt.mesa jamarray.mesa jam.config

```
--file JaMControl.mesa
--Written by John Warnock, Feb., 1979.
--Updated: March 19, 1979  9:07 AM by MN

DIRECTORY

SegmentDefs: FROM "SegmentDefs",
StringDefs: FROM "StringDefs",
IODefs: FROM "IODefs",
SystemDefs: FROM "SystemDefs",
FrameDefs: FROM "FrameDefs",
ControlDefs: FROM "ControlDefs",
ImageDefs: FROM "ImageDefs",
JaMMasterDefs: FROM "JaMMasterDefs",
JaMExecDefs: FROM "JaMExecDefs",
JaMFnsDefs: FROM "JaMFnsDefs",
JaMVMDefs: FROM "JaMVMDefs",
JaMControlDefs: FROM "JaMControlDefs",
JaMLiteralDefs: FROM "JaMLiteralDefs",
JaMDictionaryDefs: FROM "JaMDictionaryDefs",
JaMStackDefs: FROM "JaMStackDefs";

JaMControl: PROGRAM
IMPORTS SegmentDefs,StringDefs,IODefs,FrameDefs,JaMExecDefs,JaMFnsDefs,
JaMVMDefs,JaMLiteralDefs,JaMDictionaryDefs,JaMStackDefs
EXPORTS JaMControlDefs =
BEGIN
OPEN JaMMasterDefs,JaMVMDefs,JaMLiteralDefs,
JaMDictionaryDefs,JaMStackDefs;


-- This is the main control program for JaM. This program registers several
-- critical commands, creates an edited stream, and then transfers execution
-- control.

FrameStack: TYPE = ARRAY FrameStkPtr OF Frame;
FrameStkPtr: TYPE = [0..16);

frame: Frame;

GetCurrentFrame: PUBLIC PROCEDURE RETURNS [frm:Frame] =
BEGIN
RETURN[frame];
END;

RegisterCommand: PUBLIC PROCEDURE [stringname:STRING,procedure:PROCEDURE ]=
BEGIN
frame:Frame;
command:CommandType Object← [nolit,CommandType[procedure]];
frame ←GetCurrentFrame[];
StringLit[stringname,frame.opstk];
Push[command,frame.opstk];
DictDefine[];
END;

RegisterString: PUBLIC PROCEDURE [stringname:STRING,stringval:STRING]=
BEGIN
frame:Frame;
frame ←GetCurrentFrame[];
StringLit[stringname,frame.opstk];
Push[MakeStringObject[stringval],frame.opstk];
DictDefine[];
END;


RegisterBoolean: PUBLIC PROCEDURE [obname:STRING,b:BOOLEAN]=
BEGIN
frame:Frame;
frame ←GetCurrentFrame[];
StringLit[obname,frame.opstk];
BooleanLit[b,frame.opstk];
DictDefine[];
END;

RegisterObject: PUBLIC PROCEDURE [obname:STRING,ob:Object]=
BEGIN
```

```
frame:Frame;
frame ←GetCurrentFrame[];
StringLit[obname,frame.opstk];
Push[ob,frame.opstk];
DictDefine[];
END;

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

LoadBCD: PROCEDURE =
-- Expects opstk: (bcdFileName)
-- Loads and STARTS the configuration in bcdFileName
BEGIN
   s: STRING ← [256];
   frame: FrameDefs.GlobalFrameHandle;
   JaMFnsDefs.PopString[s ! IODefs.LineOverflow =>
                              JaMExecDefs.JaMError[LongFileName,TRUE]];
   frame ← FrameDefs.New[s ! SegmentDefs.FileNameError =>
                              BEGIN
                              --maybe extension omitted
                              StringDefs.AppendString[s,".bcd"];
                              frame ← FrameDefs.New[s !
                                 SegmentDefs.FileNameError =>
                                 JaMExecDefs.JaMError[BadFileName,TRUE]];
                              CONTINUE;
                              END];
   IF frame#ControlDefs.NullGlobalFrame THEN FrameDefs.Start[frame];
END;

LongFileName: StringType Object;  --must be initialized after starting VM
BadFileName: StringType Object;

OldVM: BOOLEAN;
SysDict:Object;
frameplace:INTEGER ←0;

-- Build the initial frame stack. (note: 16 entries).

framestack:FrameStack;

stackLinkArray: PRIVATE ARRAY [0..SIZE[Frame]*LENGTH[framestack]) OF StackLink;

i: CARDINAL;

FOR i IN [0..LENGTH[stackLinkArray]) DO
  stackLinkArray[i] ← NIL
  ENDLOOP;

FOR i IN [0..LENGTH[framestack]) DO
  framestack[i] ←
   [@stackLinkArray[SIZE[Frame]*i],
    @stackLinkArray[SIZE[Frame]*i+1],
    @stackLinkArray[SIZE[Frame]*i+2]]
  ENDLOOP;

frame ← framestack[frameplace];
OldVM ← RestartVM["JaM.VM",1000,20];

IF OldVM THEN
BEGIN
GetWordsVM[LOOPHOLE[LONG[0]],@SysDict,SIZE[Object]];
Push[SysDict,frame.dictstk];
END
ELSE
BEGIN
-- Build the System Dictionary (note: only 100 entries).
vma:LONG POINTER ←AllocateWordsVM[SIZE[Object]]; --This must be zero.
IntegerLit[256,frame.opstk];
DictDict[];
DictBegin[];
SysDict←Top[frame.dictstk];
PutWordsVM[vma,@SysDict,SIZE[Object]];
END;

LongFileName ← MakeStringObject[".longname"];
```

```
BadFileName ← MakeStringObject[".badname"];

-- Register the needed commands that the stack, and dictionary modules
-- could not register.

RegisterCommand[".quit"L,Quit];
--Stack commands
RegisterCommand[".pop"L,PopOpStk];
RegisterCommand[".exch"L,ExchOpStk];
RegisterCommand[".dup"L,DupOpStk];
RegisterCommand[".clrstk"L,ClearOpStk];
RegisterCommand[".copy"L,CopyOpStk];
RegisterCommand[".roll"L,RollOpStk];
RegisterCommand[".cntstk"L,CountOpStk];
RegisterCommand[".cnttomrk"L,CountToMrk];
RegisterCommand[".clrtomrk"L,ClearToMrk];
RegisterCommand[".mark"L,Mark];
--Dictionary commands
RegisterObject[".sysdict"L,SysDict];
RegisterCommand[".dict"L,DictDict];
RegisterCommand[".maxlength"L,DictMaxLength];
RegisterCommand[".known"L,DictKnown];
RegisterCommand[".where"L,DictWhere];
RegisterCommand[".get"L,DictGet];
RegisterCommand[".put"L,DictPut];
RegisterCommand[".def"L,DictDefine];
RegisterCommand[".load"L,DictLoad];
RegisterCommand[".store"L,DictStore];
RegisterCommand[".del"L,DictDelete];
RegisterCommand[".clrdict"L,DictClear];
RegisterCommand[".begin"L,DictBegin];
RegisterCommand[".end"L,DictEnd];
RegisterCommand[".dictforall"L,DictForall];
RegisterString[".undefkey"L," (Undefined key - .undefkey: ).print
     (                    ) .cvis .print"];
--RegisterString[".prompt"L,"(*) .print"];
--Booleans
RegisterBoolean[".true"L,TRUE];
RegisterBoolean[".false"L,FALSE];
--Loading a configuration
RegisterCommand[".loadbcd"L,LoadBCD];

-- All other commands are expected to register their own intrinsics.
END.
```

```
--file JaMFns.mesa
--Written by John Warnock
--Last Edited: March 1, 1979  1:43 PM by MN

DIRECTORY

IODefs: FROM "IODefs",
StreamDefs: FROM "StreamDefs",
JaMStackDefs: FROM "JaMStackDefs",
JaMControlDefs: FROM "JaMControlDefs",
JaMFnsDefs: FROM "JaMFnsDefs",
JaMLiteralDefs: FROM "JaMLiteralDefs",
JaMExecDefs: FROM "JaMExecDefs",
JaMVMDefs: FROM "JaMVMDefs",
JaMMasterDefs: FROM "JaMMasterDefs";

JaMFns: PROGRAM
IMPORTS IODefs,JaMStackDefs,JaMControlDefs,JaMLiteralDefs,
   JaMExecDefs,JaMVMDefs
EXPORTS JaMFnsDefs =
BEGIN
OPEN JaMMasterDefs,JaMFnsDefs;

Register: PUBLIC PROCEDURE[string:STRING, proc: PROCEDURE] =
BEGIN
   JaMControlDefs.RegisterCommand[string,proc];
END;

JaMExec: PUBLIC PROCEDURE[s:STRING] =
BEGIN
   JaMLiteralDefs.StringLit[s,JaMControlDefs.GetCurrentFrame[].execstk];
END;

PushInteger: PUBLIC PROCEDURE[i:INTEGER]=
BEGIN
   JaMLiteralDefs.IntegerLit[i,JaMControlDefs.GetCurrentFrame[].opstk];
END;

PopInteger: PUBLIC PROCEDURE RETURNS[INTEGER]=
BEGIN
   stack: Stack ← JaMControlDefs.GetCurrentFrame[].opstk;
   ob: Object = JaMStackDefs.Pop[stack];
   WITH dob:ob SELECT FROM
   IntegerType    => RETURN[dob.IntegerVal];
   ENDCASE        => BEGIN
                     JaMStackDefs.Push[ob,stack];
                     JaMExecDefs.JaMError[TypeChk,TRUE];
                     END;
END;

PushLongInteger: PUBLIC PROCEDURE[l: LONG INTEGER] =
BEGIN
   JaMLiteralDefs.LongIntegerLit[l,JaMControlDefs.GetCurrentFrame[].opstk];
END;

PopLongInteger: PUBLIC PROCEDURE RETURNS[LONG INTEGER] =
BEGIN
   stack: Stack ← JaMControlDefs.GetCurrentFrame[].opstk;
   ob: Object = JaMStackDefs.Pop[stack];
   WITH dob:ob SELECT FROM
   LongIntegerType   => RETURN[dob.LongIntegerVal];
   ENDCASE           => BEGIN
                        JaMStackDefs.Push[ob,stack];
                        JaMExecDefs.JaMError[TypeChk,TRUE];
                        END;
END;

PushReal: PUBLIC PROCEDURE[r: REAL] =
BEGIN
   JaMLiteralDefs.RealLit[r,JaMControlDefs.GetCurrentFrame[].opstk];
END;

PopReal: PUBLIC PROCEDURE RETURNS[REAL] =
BEGIN
   stack: Stack ← JaMControlDefs.GetCurrentFrame[].opstk;
   ob: Object = JaMStackDefs.Pop[stack];
```

```
   WITH dob:ob SELECT FROM
   RealType          => RETURN[dob.RealVal];
   ENDCASE           => BEGIN
                        JaMStackDefs.Push[ob,stack];
                        JaMExecDefs.JaMError[TypeChk,TRUE];
                        END;
END;

PushBoolean: PUBLIC PROCEDURE[b: BOOLEAN] =
BEGIN
   JaMLiteralDefs.BooleanLit[b,JaMControlDefs.GetCurrentFrame[].opstk];
END;

PopBoolean: PUBLIC PROCEDURE RETURNS[BOOLEAN] =
BEGIN
   stack: Stack ← JaMControlDefs.GetCurrentFrame[].opstk;
   ob: Object = JaMStackDefs.Pop[stack];
   WITH dob:ob SELECT FROM
   BooleanType       => RETURN[dob.BooleanVal];
   ENDCASE           => BEGIN
                        JaMStackDefs.Push[ob,stack];
                        JaMExecDefs.JaMError[TypeChk,TRUE];
                        END;
END;

PushString: PUBLIC PROCEDURE[s: STRING] =
BEGIN
   JaMLiteralDefs.StringLit[s,JaMControlDefs.GetCurrentFrame[].opstk];
END;

PopString: PUBLIC PROCEDURE[s: STRING] =
--Note different calling sequence - user supplies string
--Generates IODefs.LineOverflow if s too small, and expects
-- another string into which it can continue putting chars.  This
-- other string can be the same one, emptied, or a new bigger one
-- into which the old string has been copied, or whatever.
BEGIN
   stack: Stack ← JaMControlDefs.GetCurrentFrame[].opstk;
   ob: Object = JaMStackDefs.Pop[stack];
   WITH dob:ob SELECT FROM
   StringType  => BEGIN
                  i: CARDINAL;
                  FOR i IN [0..dob.Length)
                  DO UNTIL s.length<s.maxlength
                     DO s ← SIGNAL IODefs.LineOverflow[s];
                     ENDLOOP;
                     s[s.length] ←
                        JaMVMDefs.GetCharVM[dob.Address, dob.Offset,i];
                     s.length ← s.length + 1;
                  ENDLOOP;
                  END;
   ENDCASE     => BEGIN
                  JaMStackDefs.Push[dob,stack];
                  JaMExecDefs.JaMError[TypeChk,TRUE];
                  END;
END;

PushStream: PUBLIC PROCEDURE[s: StreamDefs.StreamHandle] =
BEGIN
   JaMLiteralDefs.StreamLit[s,JaMControlDefs.GetCurrentFrame[].opstk];
END;

PopStream: PUBLIC PROCEDURE RETURNS[StreamDefs.StreamHandle] =
BEGIN
   stack: Stack ← JaMControlDefs.GetCurrentFrame[].opstk;
   ob: Object = JaMStackDefs.Pop[stack];
   WITH dob:ob SELECT FROM
   StreamType        => RETURN[dob.SHandle];
   ENDCASE           => BEGIN
                        JaMStackDefs.Push[ob,stack];
                        JaMExecDefs.JaMError[TypeChk,TRUE];
                        END;
END;

PushObject: PUBLIC PROCEDURE[obj: Object] =
BEGIN
```

```
   JaMStackDefs.Push[obj,JaMControlDefs.GetCurrentFrame[].opstk];
END;

PopObject: PUBLIC PROCEDURE RETURNS[Object] =
BEGIN
   RETURN[JaMStackDefs.Pop[JaMControlDefs.GetCurrentFrame[].opstk]];
END;

TypeChk: StringType Object ← JaMLiteralDefs.MakeStringObject[".typechk"];

END.
```

```
-- JaMDictionary.mesa --
-- Written by Martin Newell, January 1979
-- Updated March 20, 1979  10:10 AM by MN

-- Administers dictionaries in VM
-- A dictionary is a type of Object, however additional information is held
-- in a DictDesc, which also lives in VM
-- A Dictionary is a set of <key,value> tuples, in which both key and value
-- are Objects (key is not necessarily a string)
-- Present implementation:
-- Dictionary is allocated as a contiguous vector of <object,object> tuples
-- Hash coding is used to access tuples
-- Dictionary stack is cached with single level cache.

DIRECTORY
   Mopcodes: FROM "Mopcodes",
   InlineDefs: FROM "InlineDefs",
   JaMFnsDefs: FROM "JaMFnsDefs",
   JaMVMDefs: FROM "JaMVMDefs",
   JaMLiteralDefs: FROM "JaMLiteralDefs",
   JaMDictionaryDefs: FROM "JaMDictionaryDefs",
   JaMStackDefs: FROM "JaMStackDefs",
   JaMTypeChkDefs: FROM "JaMTypeChkDefs",
   JaMExecDefs: FROM "JaMExecDefs",
   JaMControlDefs: FROM "JaMControlDefs",
   JaMMasterDefs: FROM "JaMMasterDefs";

JaMDictionary: PROGRAM
   IMPORTS JaMFnsDefs,JaMVMDefs,JaMLiteralDefs,
      JaMDictionaryDefs,JaMStackDefs,JaMTypeChkDefs,JaMExecDefs,
      JaMControlDefs
   EXPORTS JaMDictionaryDefs =
   BEGIN OPEN JaMMasterDefs,InlineDefs;

-- TYPES

DictDesc: TYPE = RECORD
--Dictionary descriptor, pointed to by DictType Object.Address
   [  maxLen,curLen: INTEGER, --in units of tuples
      base: LONG POINTER,  --word address of zone in VM
      sizew: CARDINAL,  --in units of words
      sizet: CARDINAL,  --in units of tuples
      stepw: CARDINAL    --for hash search, in units of words
   ];

Tuple: TYPE = RECORD[key,value: Object];

-- GLOBALS

tupleSize: CARDINAL = 2*SIZE[Object];
nullObject: NullType Object ← [lit,NullType[]];

Cache: ARRAY [0..cachesize) OF Tuple;
cachesize: CARDINAL = 64;  --Must be power of 2
cachemaxLen: CARDINAL = cachesize*2/3; --max # entries allowed
cachecurLen: CARDINAL;
nullTuple: Tuple = [nullObject,nullObject];



-- CREATION

   Dictionary: PUBLIC PROCEDURE[maxLen: CARDINAL]
               RETURNS[dict: DictType Object] =
-- Return a new dictionary for up to maxlen tuples
   BEGIN
      t: CARDINAL ← maxLen/2*3;
      w: CARDINAL ← t*tupleSize;
      loc: LONG POINTER;
      DD: DictDesc ←[
            maxLen: maxLen,
            curLen: 0,
            base: JaMVMDefs.AllocateWordsVM[w],
            sizew: w,
            sizet: t,
            stepw: tupleSize];
```

```
      dict ← [lit,DictType[JaMVMDefs.AllocateWordsVM[SIZE[DictDesc]]]];
   -- Write DictDesc to VM
      JaMVMDefs.PutWordsVM[dict.Address,@DD,SIZE[DictDesc]];
   -- Clear the Dictionary
      FOR loc ← DD.base, loc+tupleSize UNTIL loc=DD.base+DD.sizew
      DO JaMVMDefs.PutWordsVM[loc,@nullObject,SIZE[Object]];
      ENDLOOP;
   END;

-- ATTRIBUTES

   Length: PUBLIC PROCEDURE[dict: DictType Object]
           RETURNS[length: CARDINAL] =
-- Return current length of dictionary dict
   BEGIN
      DD: DictDesc;
      JaMVMDefs.GetWordsVM[dict.Address,@DD,SIZE[DictDesc]];
      RETURN[DD.curLen];
   END;

   MaxLength: PUBLIC PROCEDURE[dict: DictType Object]
           RETURNS[length: CARDINAL] =
-- Return maximum allowable length of dictionary dict
   BEGIN
      DD: DictDesc;
      JaMVMDefs.GetWordsVM[dict.Address,@DD,SIZE[DictDesc]];
      RETURN[DD.maxLen];
   END;

-- ACCESS

   Known: PUBLIC PROCEDURE[dict: DictType Object, key: Object]
         RETURNS[known: BOOLEAN] =
   BEGIN
      DD: DictDesc;
      JaMVMDefs.GetWordsVM[dict.Address,@DD,SIZE[DictDesc]];
      [known,] ← LookUp[DD,key,Hash[key] MOD DD.sizet];
   END;

   Where: PUBLIC PROCEDURE[dictstk: JaMMasterDefs.Stack, key: Object]
         RETURNS[known: BOOLEAN, dict: DictType Object] =
   BEGIN
      hash: CARDINAL ← Hash[key];
      location: LONG POINTER;
      cacheloc: INTEGER;
      value: Object;
      Find: PROCEDURE[obj: Object] RETURNS[done: BOOLEAN] =
      BEGIN
         DD: DictDesc;
         dict ← JaMTypeChkDefs.DescDictType[obj];
         JaMVMDefs.GetWordsVM[dict.Address,@DD,SIZE[DictDesc]];
         [done, location] ← LookUp[DD,key,hash MOD DD.sizet];
      END;

      known ← JaMStackDefs.StackForall[dictstk,Find];
   --enter it in cache
      IF known
      THEN  BEGIN
            JaMVMDefs.GetWordsVM[location+SIZE[Object],@value,SIZE[Object]];
            [,cacheloc] ← LookUpCache[key, BITAND[hash,cachesize-1]];
            EnterInCache[[key,value], hash, cacheloc];
            END;
   END;

   Get: PUBLIC PROCEDURE[dict: DictType Object, key: Object]
         RETURNS[value: Object] =
-- Generates undefkey
   BEGIN
      DD: DictDesc;
      known: BOOLEAN;
      location: LONG POINTER;
      JaMVMDefs.GetWordsVM[dict.Address,@DD,SIZE[DictDesc]];
      [known,location] ← LookUp[DD,key,Hash[key] MOD DD.sizet];
      IF known
      THEN  JaMVMDefs.GetWordsVM[location+SIZE[Object],@value,SIZE[Object]]
      ELSE  ERROR JaMExecDefs.JaMError[undefkey,TRUE];
```

```
   END;

   Put: PUBLIC PROCEDURE[dict: DictType Object, key,value: Object] =
-- Generates dictfull:
   BEGIN
      DD: DictDesc;
      hash: CARDINAL = Hash[key];
      known: BOOLEAN;
      location: LONG POINTER;
      cacheloc: INTEGER;
      JaMVMDefs.GetWordsVM[dict.Address,@DD,SIZE[DictDesc]];
      [known,location] ← LookUp[DD, key, hash MOD DD.sizet];
      IF ~known THEN
         IF location=NIL
         THEN  ERROR JaMExecDefs.JaMError[dictfull,TRUE]
         ELSE  BEGIN
               JaMVMDefs.PutWordsVM[location,@key,SIZE[Object]];
               DD.curLen ← DD.curLen+1;
               JaMVMDefs.PutWordsVM[dict.Address,@DD,SIZE[DictDesc]];
               END;
      JaMVMDefs.PutWordsVM[location+SIZE[Object],@value,SIZE[Object]];
   --just in case dict is on stack:
      [known,cacheloc] ← LookUpCache[key, BITAND[hash,cachesize-1]];
      IF known THEN Cache[cacheloc] ← nullTuple;
   END;

   Load: PUBLIC PROCEDURE[dictstk: JaMMasterDefs.Stack, key: Object]
         RETURNS[value: Object] =
-- Return value looked up in highest dictionary having key in dictstk
-- Generates undefkey
   BEGIN
      hash: CARDINAL ← Hash[key];
      known: BOOLEAN;
      location: LONG POINTER;
      cacheloc: INTEGER;
      Find: PROCEDURE[obj: Object] RETURNS[done: BOOLEAN] =
      BEGIN
         DD: DictDesc;
         dict: DictType Object ← JaMTypeChkDefs.DescDictType[obj];
         JaMVMDefs.GetWordsVM[dict.Address,@DD,SIZE[DictDesc]];
         [done, location] ← LookUp[DD,key,hash MOD DD.sizet];
      END;

      [known,cacheloc] ← LookUpCache[key, BITAND[hash,cachesize-1]];
      IF known THEN  RETURN[Cache[cacheloc].value];
-- Must search dictionary stack
      IF JaMStackDefs.StackForall[dictstk,Find]
      THEN  BEGIN
            JaMVMDefs.GetWordsVM[location+SIZE[Object],@value,SIZE[Object]];
            EnterInCache[[key,value], hash, cacheloc];
            END
      ELSE  ERROR JaMExecDefs.JaMError[undefkey,TRUE];
   END;

   Store: PUBLIC PROCEDURE[dictstk: JaMMasterDefs.Stack, key,value: Object] =
-- Enters <key,value> into highest dictionary having key in dictstk,
--    or into dict on top of dictstk
-- Generates dictfull:
   BEGIN
      hash: CARDINAL ← Hash[key];
      location: LONG POINTER;
      cacheloc: INTEGER;
      Find: PROCEDURE[obj: Object] RETURNS[done: BOOLEAN] =
      BEGIN
         dict: DictType Object ← JaMTypeChkDefs.DescDictType[obj];
         DD: DictDesc;
         JaMVMDefs.GetWordsVM[dict.Address,@DD,SIZE[DictDesc]];
         [done, location] ← LookUp[DD,key,hash MOD DD.sizet];
      END;

      IF JaMStackDefs.StackForall[dictstk,Find]
      THEN  JaMVMDefs.PutWordsVM[location+SIZE[Object],@value,SIZE[Object]]
      ELSE  Put[JaMTypeChkDefs.DescDictType[JaMStackDefs.Top[dictstk]],
                key,value];
-- Update cache
      [,cacheloc] ← LookUpCache[key, BITAND[hash, cachesize-1]];
```

```
      EnterInCache[[key,value], hash, cacheloc];
   END;

   Delete: PUBLIC PROCEDURE[dict: DictType Object, key: Object] =
-- Deletes object key from dictionary
-- Generates undefkey if object not found:
   BEGIN
      DD: DictDesc;
      hash: CARDINAL = Hash[key];
      known: BOOLEAN;
      location: LONG POINTER;
      cacheloc: INTEGER;
      JaMVMDefs.GetWordsVM[dict.Address,@DD,SIZE[DictDesc]];
      [known,location] ← LookUp[DD, key, hash MOD DD.sizet];
      IF ~known THEN ERROR JaMExecDefs.JaMError[undefkey,TRUE];
      JaMVMDefs.PutWordsVM[location,@nullObject,SIZE[Object]];
      DD.curLen ← DD.curLen-1;
      JaMVMDefs.PutWordsVM[dict.Address,@DD,SIZE[DictDesc]];
   -- Remove entry from cache just in case dict on stack:
      [known,cacheloc] ← LookUpCache[key, BITAND[hash,cachesize-1]];
      IF known THEN Cache[cacheloc] ← nullTuple;
   END;

   Clear: PUBLIC PROCEDURE[dict: DictType Object] =
-- Deletes all objects from dictionary dict
   BEGIN
      DD: DictDesc;
      loc: LONG POINTER;
      JaMVMDefs.GetWordsVM[dict.Address,@DD,SIZE[DictDesc]];
      FOR loc ← DD.base, loc+tupleSize UNTIL loc=DD.base+DD.sizew
      DO JaMVMDefs.PutWordsVM[loc,@nullObject,SIZE[Object]];
      ENDLOOP;
      DD.curLen ← 0;
      JaMVMDefs.PutWordsVM[dict.Address,@DD,SIZE[DictDesc]];
   -- just in case dict is on stack:
      ClearCache[];
   END;

   NextTuple: PUBLIC PROCEDURE[dict: DictType Object,
                     oldtupleptr: LONG POINTER]
              RETURNS[key,value: Object, tupleptr: LONG POINTER] =
-- Increments oldtupleptr to next non-null tuple and returns it together
--    with the key and object of the tuple
-- Initial call should have oldtupleptr=NIL
-- Returns tupleptr=NIL if no more tuples
-- Normal enumeration is not used because of way JaM control works
   BEGIN
      DD: DictDesc;
      JaMVMDefs.GetWordsVM[dict.Address,@DD,SIZE[DictDesc]];
      FOR   tupleptr ←  IF oldtupleptr=NIL
                        THEN DD.base
                        ELSE oldtupleptr+tupleSize,
                        tupleptr+tupleSize
                  UNTIL tupleptr-DD.base>=DD.sizew
      DO JaMVMDefs.GetWordsVM[tupleptr,@key,SIZE[Object]];
         IF key.Type#NullType
         THEN  BEGIN
               JaMVMDefs.GetWordsVM[tupleptr+SIZE[Object],@value,SIZE[Object]];
               RETURN;
               END;
      ENDLOOP;
      tupleptr ← NIL;
   END;

   Begin: PUBLIC PROCEDURE[dictstk: JaMMasterDefs.Stack,
                           dict: DictType Object] =
-- Push dictionary dict onto stack dictstk, dealing with cache
   BEGIN
      JaMStackDefs.Push[dict,dictstk];
      ClearCache[];
   END;

   End: PUBLIC PROCEDURE[dictstk: JaMMasterDefs.Stack] =
-- Pop stack dictstk, dealing with cache
   BEGIN
      IF JaMStackDefs.CountStk[dictstk] <= 1 --disallow popping .sysdict
```

```
      THEN ERROR JaMExecDefs.JaMError[StkUndFlw,TRUE];
      [] ← JaMStackDefs.Pop[dictstk];
      ClearCache[];
   END;


-- PRIVATE Procedures

   Hash: PROCEDURE[key: Object] RETURNS[h: CARDINAL] =
-- hashes object key into a CARDINAL
-- Expects caller to do: (h MOD range) to allow for different ranges with same key
   BEGIN OPEN InlineDefs;
      WITH k:key SELECT FROM
         NullType       => h ← 0;
         IntegerType    => h ← k.IntegerVal;
         LongIntegerType   => h ← LowHalfLI[k.LongIntegerVal];
         RealType       => h ← HighHalfRE[k.RealVal];
         BooleanType    => h ← 0;
         StringType        =>
            BEGIN
               i: CARDINAL;
               h ← 0;
               FOR i IN [0..k.Length) DO
                  h ← BITXOR[BITSHIFT[h,1],
                        LOOPHOLE[JaMVMDefs.GetCharVM[k.Address,k.Offset,i]]];
               ENDLOOP;
            END;
         StreamType     => h ← LOOPHOLE[k.SHandle];
         CommandType => h ← LOOPHOLE[k.Command];
         DictType    => h ← LowHalfLP[k.Address];
         ArrayType      => h ← LowHalfLP[k.ArrayPtr];
         StackType      => h ← LOOPHOLE[k.StkPtr];
         FrameType      => h ← LOOPHOLE[k.FrmPtr];
         ENDCASE;
   END;

   LookUpCache: PROCEDURE[key: Object, hash: CARDINAL]
            RETURNS[known: BOOLEAN, loc: INTEGER] =
-- Looks in dictionary cache Cache for object with key, hashed to hash.
-- If found then returns [TRUE, where it is]
-- else returns [FALSE, index where it will go if
--                      you want to do a Put or -1 if no room]
   BEGIN
      k: Object;
      loc ← hash;
      THROUGH [0..cachesize)
      DO k ← Cache[loc].key;
         SELECT TRUE FROM
            EqualObject[key,k] => RETURN[TRUE,loc];
            k.Type=NullType => RETURN[FALSE, IF cachecurLen < cachemaxLen
                                             THEN loc ELSE -1];
         ENDCASE;
         loc ← loc + 1;
         IF loc=cachesize THEN loc ← 0;
      ENDLOOP;
      RETURN[FALSE,-1]; --shouldn't ever get here
   END;

   LookUp: PROCEDURE[DD: DictDesc, key: Object, hash: CARDINAL]
            RETURNS[known: BOOLEAN, location: LONG POINTER] =
-- Looks in dictionary DD for object with key, hashed to hash.
-- If found then returns [TRUE, where it is]
-- else returns [FALSE,tuple address where it will go if
--                      you want to do a Put or NIL if no room]
   BEGIN
      start: LONG POINTER ← DD.base + hash*tupleSize;
      notFirstTime: BOOLEAN ← FALSE;
      k: Object;

      FOR   location ← start,
         IF DD.stepw<DD.base+DD.sizew-location
         THEN location+DD.stepw
         ELSE location-(DD.sizew-DD.stepw)
      DO
         IF location=start AND notFirstTime THEN RETURN[FALSE,NIL];
         JaMVMDefs.GetWordsVM[location,@k,SIZE[Object]];
```

```
         SELECT TRUE FROM
            EqualObject[key,k] => RETURN[TRUE,location];
            k.Type=NullType => RETURN[FALSE, IF DD.curLen < DD.maxLen
                                             THEN location ELSE NIL];
         ENDCASE;
         notFirstTime ← TRUE;
      ENDLOOP;
   END;

   EqualObject: PROCEDURE[ob1,ob2: Object]
               RETURNS[equal: BOOLEAN] =
   -- Checks for equality of values of objects
   BEGIN
      IF ob1.Type#ob2.Type THEN RETURN[FALSE];
      WITH o1:ob1 SELECT FROM
         NullType       => RETURN[TRUE];
         IntegerType    => RETURN[o1.IntegerVal=
                              JaMTypeChkDefs.DescIntegerType[ob2].IntegerVal];
         LongIntegerType   => RETURN[o1.LongIntegerVal=
                     JaMTypeChkDefs.DescLongIntegerType[ob2].LongIntegerVal];
         RealType       => RETURN[o1.RealVal=
                              JaMTypeChkDefs.DescRealType[ob2].RealVal];
         BooleanType    => RETURN[o1.BooleanVal=
                              JaMTypeChkDefs.DescBooleanType[ob2].BooleanVal];
         StringType        => RETURN[Match[o1,
                              JaMTypeChkDefs.DescStringType[ob2]]];
         StreamType        => RETURN[o1.SHandle=
                              JaMTypeChkDefs.DescStreamType[ob2].SHandle];
         CommandType    => RETURN[o1.Command=
                              JaMTypeChkDefs.DescCommandType[ob2].Command];
         DictType       => RETURN[o1.Address=
                              JaMTypeChkDefs.DescDictType[ob2].Address];
         ArrayType         => RETURN[o1.ArrayPtr=
                              JaMTypeChkDefs.DescArrayType[ob2].ArrayPtr];
         StackType         => RETURN[o1.StkPtr=
                              JaMTypeChkDefs.DescStackType[ob2].StkPtr];
         FrameType         => RETURN[o1.FrmPtr=
                              JaMTypeChkDefs.DescFrameType[ob2].FrmPtr];
         ENDCASE;
   END;

   Match: PUBLIC PROCEDURE [s1,s2: StringType Object] RETURNS[BOOLEAN] =
BEGIN
i: CARDINAL;
IF s1.Length#s2.Length THEN RETURN[FALSE];
FOR i IN [0..s1.Length) DO
   IF JaMVMDefs.GetCharVM[s1.Address,s1.Offset,i]#
   JaMVMDefs.GetCharVM[s2.Address,s2.Offset,i]
   THEN RETURN[FALSE];
ENDLOOP;
RETURN[TRUE];
END;

   EnterInCache: PROCEDURE[tuple: Tuple, hash: CARDINAL, cacheloc: INTEGER] =
   BEGIN
      IF cacheloc=-1
      THEN  BEGIN
            ClearCache[];
            cacheloc ← BITAND[hash, cachesize-1];
            END;
      Cache[cacheloc] ← tuple;
   END;

   ClearCache: PROCEDURE =
   BEGIN --could be speeded up using bitblt
      i: CARDINAL;
      FOR i IN [0..cachesize) DO Cache[i] ← nullTuple; ENDLOOP;
      cachecurLen ← 0;
   END;

   HighHalfLP: PROCEDURE [LONG POINTER] RETURNS [CARDINAL] =
      MACHINE CODE BEGIN Mopcodes.zEXCH; Mopcodes.zPOP END;

   LowHalfLP: PROCEDURE [LONG POINTER] RETURNS [CARDINAL] =
      MACHINE CODE BEGIN Mopcodes.zPOP END;
```

```
   HighHalfRE: PROCEDURE [REAL] RETURNS [CARDINAL] =
      MACHINE CODE BEGIN Mopcodes.zEXCH; Mopcodes.zPOP END;

   LowHalfLI: PROCEDURE [LONG INTEGER] RETURNS [CARDINAL] =
      MACHINE CODE BEGIN Mopcodes.zPOP END;


--*** JaM INTRINSICS ***

   DictDict: PUBLIC PROCEDURE =
-- Expects opstk: (maxLength)
-- Returns opstk: (new dictionary to hold up to maxLength objects)
   BEGIN OPEN JaMFnsDefs;
      PushDict[Dictionary[PopInteger[]]];
      END;

--"DictLength" is handled in JaMAttributes

   DictMaxLength: PUBLIC PROCEDURE =
-- Expects opstk: (dictionary)
-- Returns opstk: (maximum number of objects in dictionary)
   BEGIN OPEN JaMFnsDefs;
      PushInteger[MaxLength[PopDict[]]];
      END;

   DictKnown: PUBLIC PROCEDURE =
-- Expects opstk: (dictionary, key)
-- Returns opstk: (boolean)
   BEGIN OPEN JaMFnsDefs;
      key: Object = PopObject[];
      dict: DictType Object = PopDict[];
      PushBoolean[Known[dict,key]];
   END;

   DictWhere: PUBLIC PROCEDURE =
-- Expects opstk: (key), dictstk: a dictionary stack
-- Returns opstk: (dictionary(iff key known), boolean("known"))
--          dictstk: unchanged
   BEGIN OPEN JaMFnsDefs;
      key: Object = PopObject[];
      known: BOOLEAN;
      dict: DictType Object;
      [known,dict] ← Where[JaMControlDefs.GetCurrentFrame[].dictstk, key];
      IF known THEN PushDict[dict];
      PushBoolean[known];
   END;

   DictGet: PUBLIC PROCEDURE =
-- Expects opstk: (dictionary, key)
-- Returns opstk: (value looked up in dictionary)
   BEGIN OPEN JaMFnsDefs;
      key: Object = PopObject[];
      dict: DictType Object = PopDict[];
      PushObject[Get[dict,key]];
   END;

   DictPut: PUBLIC PROCEDURE =
-- Expects opstk: (dictionary, key, value)
-- Enters <key,value> into dictionary
-- Returns opstk: ()
-- No indication of whether an object with that key already existed.
   BEGIN OPEN JaMFnsDefs;
      value: Object = PopObject[];
      key: Object = PopObject[];
      dict: DictType Object = PopDict[];
      Put[dict,key,value];
   END;

   DictDefine: PUBLIC PROCEDURE =
-- Expects opstk: (key, value), dictstk: (dictionary)
-- Enters <key,value> into dictionary
-- Returns opstk: (), dictstk: (dictionary)
-- No indication of whether an object with that key already existed.
   BEGIN OPEN JaMFnsDefs;
      value: Object = PopObject[];
      key: Object = PopObject[];
```

```
      dict: DictType Object =
         JaMTypeChkDefs.DescDictType[
            JaMStackDefs.Top[JaMControlDefs.GetCurrentFrame[].dictstk]];
      Put[dict,key,value];
   END;

   DictLoad: PUBLIC PROCEDURE =
-- Expects opstk: (key), dictstk: a dictionary stack
-- Returns opstk: (value looked up in highest dictionary having key in dictstk),
--           dictstk: unchanged
   BEGIN OPEN JaMFnsDefs;
      key: Object = PopObject[];
      PushObject[Load[JaMControlDefs.GetCurrentFrame[].dictstk,key]];
   END;

   DictStore: PUBLIC PROCEDURE =
-- Expects opstk: (key, value), dictstk: a dictionary stack
-- Enters <key,value> into highest dictionary having key in dictstk,
--    or into dict on top of dictstk
-- Returns opstk: (), dictstk: unchanged
   BEGIN OPEN JaMFnsDefs;
      value: Object = PopObject[];
      key: Object = PopObject[];
      Store[JaMControlDefs.GetCurrentFrame[].dictstk,key,value];
   END;

   DictDelete: PUBLIC PROCEDURE =
-- Expects opstk: (dictionary, key)
-- Deletes object key from dictionary
-- Returns opstk: ()
   BEGIN OPEN JaMFnsDefs;
      key: Object = PopObject[];
      dict: DictType Object = PopDict[];
      Delete[dict,key];
   END;

   DictClear: PUBLIC PROCEDURE =
-- Expects opstk: (dictionary)
-- Deletes all objects from dictionary
-- Returns opstk: ()
   BEGIN OPEN JaMFnsDefs;
      Clear[PopDict[]];
   END;

   DictForall: PUBLIC PROCEDURE =
-- Expects opstk: (dictionary)(object)
-- For each tuple in dictionary put (key)(value) onto opstk and execute object
-- Returns opstk: ()
   BEGIN OPEN JaMFnsDefs;
      mark: MarkType Object←[nolit,MarkType[]];
      frm: Frame = JaMControlDefs.GetCurrentFrame[];
      obj: Object = PopObject[];
      dict: DictType Object = PopDict[];
   -- save them on exec stack
      JaMStackDefs.Push[mark,frm.execstk];
      JaMStackDefs.Push[dict,frm.execstk];
      JaMStackDefs.Push[obj,frm.execstk];
   -- prime state
      JaMStackDefs.Push[[lit,LongIntegerType[LOOPHOLE[LONG[NIL]]]],frm.execstk];
   -- start it
      DFAProc[];
   END;

   DFAProcObject: CommandType Object = [nolit,CommandType[DFAProc]];

   DFAProc: PROCEDURE =
   BEGIN OPEN JaMFnsDefs;
      frm: Frame = JaMControlDefs.GetCurrentFrame[];
      tupleptr: LONG POINTER ←
         LOOPHOLE[JaMTypeChkDefs.DescLongIntegerType[
                  JaMStackDefs.Pop[frm.execstk]].LongIntegerVal];
      obj: Object = JaMStackDefs.Pop[frm.execstk];
      dict: DictType Object = JaMTypeChkDefs.DescDictType[JaMStackDefs.Pop[frm.execstk]];
      key,value: Object;
   -- get tuple onto opstk
      [key,value,tupleptr] ← NextTuple[dict,tupleptr];
```

```
      IF tupleptr=NIL
      THEN  BEGIN
            [] ← JaMStackDefs.Pop[frm.execstk];  --remove mark
            RETURN;
            END;
      JaMStackDefs.Push[key,frm.opstk];
      JaMStackDefs.Push[value,frm.opstk];
   -- set up stack
      JaMStackDefs.Push[dict,frm.execstk];
      JaMStackDefs.Push[obj,frm.execstk];
      JaMStackDefs.Push[[lit,LongIntegerType[
                           LOOPHOLE[tupleptr,LONG INTEGER]]],
                        frm.execstk];
      JaMStackDefs.Push[DFAProcObject,frm.execstk];
      JaMStackDefs.Push[obj,frm.execstk];
   -- and let it happen
   END;

   DictBegin: PUBLIC PROCEDURE =
-- Expects opstk: (dictionary), dictstk: ()
-- Returns opstk: (), dictstk: (dictionary)
   BEGIN
      frm: Frame = JaMControlDefs.GetCurrentFrame[];
      Begin[frm.dictstk,PopDict[]];
   END;

   DictEnd: PUBLIC PROCEDURE =
-- Expects dictstk: (dictionary)
-- Returns dictstk: ()
   BEGIN
      End[JaMControlDefs.GetCurrentFrame[].dictstk];
   END;

--Private

PushDict: PROCEDURE[d: DictType Object] =
BEGIN
   JaMStackDefs.Push[d,JaMControlDefs.GetCurrentFrame[].opstk];
END;

PopDict: PROCEDURE RETURNS[DictType Object] =
BEGIN
   stack: Stack ← JaMControlDefs.GetCurrentFrame[].opstk;
   ob: Object = JaMStackDefs.Pop[stack];
   WITH dob:ob SELECT FROM
   DictType    => BEGIN
                  dict: DictType Object = dob;
                  RETURN[dict];
                  END;
   ENDCASE     => BEGIN
                  JaMStackDefs.Push[ob,stack];
                  JaMExecDefs.JaMError[TypeChk,TRUE];
                  END;
END;

-- Error string objects
undefkey: StringType Object = JaMLiteralDefs.MakeStringObject[".undefkey"];
dictfull: StringType Object = JaMLiteralDefs.MakeStringObject[".dictfull"];
TypeChk: StringType Object ← JaMLiteralDefs.MakeStringObject[".typechk"];
StkUndFlw: StringType Object = JaMLiteralDefs.MakeStringObject[".stkundflw"];

   END.
```

```
--file JaMExec.mesa
--Written by John Warnock, January, 1978.
--Last Updated: March 19, 1979  3:08 PM by MN


DIRECTORY

JaMMasterDefs: FROM "JaMMasterDefs",
JaMFnsDefs: FROM "JaMFnsDefs",
JaMScannerDefs: FROM "JaMScannerDefs",
JaMExecDefs: FROM "JaMExecDefs",
JaMControlDefs: FROM "JaMControlDefs",
JaMTypeChkDefs: FROM "JaMTypeChkDefs",
JaMLiteralDefs: FROM "JaMLiteralDefs",
JaMArrayDefs: FROM "JaMArrayDefs",
JaMIODefs: FROM "JaMIODefs",
JaMDictionaryDefs: FROM "JaMDictionaryDefs",
JaMStackDefs: FROM "JaMStackDefs";

JaMExec: PROGRAM  IMPORTS JaMScannerDefs,JaMFnsDefs,JaMTypeChkDefs,
JaMDictionaryDefs,JaMControlDefs,JaMLiteralDefs,JaMArrayDefs,JaMStackDefs EXPORTS JaMExecDefs =
BEGIN
OPEN JaMMasterDefs,JaMFnsDefs,JaMExecDefs,JaMControlDefs,JaMTypeChkDefs,
JaMScannerDefs,JaMIODefs,JaMStackDefs;

-- The following routines provide for the control of execution. It is
-- assumed that the current stack is the execution stack.

--"Execute" executes the undescriminated object.

Execute: PUBLIC PROCEDURE =
BEGIN
-- Some Definitions:
SingStep: PROCEDURE [ob:Object] =
   BEGIN
   Push[ob,frame.opstk];
   Push[asstep,frame.execstk];
   Push[step,frame.execstk];
   Push[arfree,frame.execstk];
   END;
ob:Object;

-- Procedure Starts here.

frame:Frame ← JaMControlDefs.GetCurrentFrame[];
UNTIL frame.execstk↑ = NIL
DO BEGIN ENABLE JaMError =>
BEGIN
Push[execute,frame.execstk];
RETRY;
END;
IF WakeUpFlag THEN
BEGIN
Push[Interrupt,frame.execstk];
WakeUpFlag ← FALSE;
END;
ob← Pop[frame.execstk];
IF ob.litflag = lit THEN BEGIN
      Push[ob,frame.opstk];
      IF stepflag THEN SingStep[ob];
      LOOP;
      END;
WITH discrimob:ob SELECT FROM
   IntegerType,
   LongIntegerType,
   RealType,
   BooleanType,
   DictType => BEGIN
         Push[ob,frame.opstk];
         IF stepflag THEN SingStep[ob];
         END;
   StreamType   =>   BEGIN
         dob:StreamType Object ←discrimob;
         IF StreamToken[dob,frame.execstk
               ! JaMError =>
               BEGIN
```

```
                Push[dob,frame.opstk];
                Push[execute,frame.execstk];
                CONTINUE;
                END] THEN
          BEGIN
          obj: Object = Pop[frame.execstk];
          Push[JaMDictionaryDefs.Load[frame.dictstk,obj
                ! JaMError =>
                BEGIN
                Push[obj,frame.opstk];
                Push[execute,frame.execstk];
                CONTINUE;
                END],
             frame.execstk];
             IF stepflag THEN SingStep[obj];
          END;
          END;
   StringType =>
          BEGIN
          dob:StringType Object ←discrimob;
          IF StringToken[dob,frame.execstk
                ! JaMError =>
                BEGIN
                Push[dob,frame.opstk];
                Push[execute,frame.execstk];
                CONTINUE;
                END] THEN
          BEGIN
          obj: Object = Pop[frame.execstk];
          Push[JaMDictionaryDefs.Load[frame.dictstk,obj
                ! JaMError =>
                BEGIN
                Push[obj,frame.opstk];
                Push[execute,frame.execstk];
                CONTINUE;
                END],
             frame.execstk];
             IF stepflag THEN SingStep[obj];
          END;
          END;
   CommandType => BEGIN
             freemark:StackLink ← FreeLoc[];
             discrimob.Command[! JaMError =>
             BEGIN
             IF restore THEN
             BEGIN
             RestoreStk[freemark,frame.opstk];
             END;
             Push[discrimob,frame.opstk];
             Push[execute,frame.execstk];
             CONTINUE;
             END];
          END;

   ArrayType      => BEGIN
          dob:ArrayType Object ← discrimob;
          []←JaMArrayDefs.ArrayAtom[dob,frame.execstk];
          END;
   StackType      => BEGIN
          END;

   FrameType      => BEGIN
          END;


   ENDCASE;
END;
ENDLOOP;

END;


--"If" is the implementation of the testing instruction.
--Two operands are required: an Object and a Boolean. If the
--Boolean is TRUE then the object is executed otherwise the
--object is popped.
```

```
If: PUBLIC PROCEDURE=
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
ob:Object ← Pop[frame.opstk];
IF PopBoolean[] THEN Push[ob,frame.execstk];
END;

--"IfElse" is the implementation of the two way conditional execution
--instruction. Three operands are required: Object1,Object2, and a  Boolean.
-- If the Boolean is TRUE then Object2 is executed otherwise Object1 is
--executed.

IfElse: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
obF:Object ← Pop[frame.opstk];
obT:Object ← Pop[frame.opstk];
Push[IF PopBoolean[] THEN obT ELSE obF,frame.execstk];
END;

--"Rept" is the "loop for count" instruction. Two operands are required:
-- an IntegerType and an Object. The Object is executed for the number
-- of times indicated by the Integer. (0 for Negative).

Rept: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
ob:Object ← Pop[frame.opstk];
i:IntegerType Object ← DescIntegerType[Pop[frame.opstk]];
m:MarkType Object←[nolit,MarkType[]];
Push[m,frame.execstk];
Push[ob,frame.execstk];
Push[i,frame.execstk];
Push[reptc,frame.execstk];
END;


CRept: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
i:IntegerType Object ← DescIntegerType[Pop[frame.execstk]];
ob:Object ← Pop[frame.execstk];
m:Object←Pop[frame.execstk];
IF i.IntegerVal <= 0 THEN RETURN ELSE
BEGIN
i.IntegerVal ← i.IntegerVal -1;
Push[m,frame.execstk];
Push[ob,frame.execstk];
Push[i,frame.execstk];
Push[reptc,frame.execstk];
Push[ob,frame.execstk];
END;
END;

--"Loop" is the "loop forever" instruction. One operand is required:
-- The Object is executed until an ".exit" command is executed.

Loop: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
ob:Object ← Pop[frame.opstk];
m:MarkType Object←[nolit,MarkType[]];
Push[m,frame.execstk];
Push[ob,frame.execstk];
Push[loopc,frame.execstk];
END;


CLoop: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
ob:Object ← Pop[frame.execstk];
Push[ob,frame.execstk];
Push[loopc,frame.execstk];
Push[ob,frame.execstk];
```

```
END;

--"Exec" Moves the top of the operand stack to the top of the execution stack.


Exec: PUBLIC PROCEDURE =
BEGIN
frame:Frame ←JaMControlDefs.GetCurrentFrame[];
JaMStackDefs.MoveStkOps[frame.opstk,frame.execstk,1];
END;

--"Stop" procedure clears the executionstack.

Stop: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
ClrStk[frame.execstk];
END;



--"Exit" procedure pops the executionstack until the inner most loop is
--terminated. (until a mark is encountered).

Exit: PUBLIC PROCEDURE =
BEGIN
ob:Object;
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
DO
ob ← Pop[frame.execstk];
WITH dob:ob SELECT FROM
MarkType => RETURN;
ENDCASE;
ENDLOOP;
END;

--"SingleStep" Sets stepflag in the execution module.  Each token is put on the operand stack
-- and ".step" is executed followed by the token.


SingleStep: PUBLIC PROCEDURE =
BEGIN
astepflag ← TRUE;
stepflag ← TRUE;
END;


--"FreeRun" Resets stepflag in the execution module.  Each token is put on the operand stack
-- and ".step" is executed followed by the token.


FreeRun: PUBLIC PROCEDURE =
BEGIN
astepflag ← FALSE;
stepflag ← FALSE;
END;

--"ASingleStep" Sets astepflag in the execution module.

ASingleStep: PUBLIC PROCEDURE =
BEGIN
stepflag ← TRUE AND astepflag;
END;


--"AFreeRun" Resets stepflag in the execution module.

AFreeRun: PUBLIC PROCEDURE =
BEGIN
stepflag ← FALSE;
END;

--"JaMError" is an error that is caught by execution control.  The
-- restore parameter is used to indicate to execution control if it
-- is safe to restore the operand stack. execute is the StringType Object
-- that will be pushed onto the execution stack.
```

```
JaMError: PUBLIC ERROR [execute:StringType Object,restore:BOOLEAN]=CODE;

--this is module initialization code.

WakeUpFlag:PUBLIC BOOLEAN ← FALSE;
step:StringType Object ← JaMLiteralDefs.MakeStringObject[".step"L];
asstep:CommandType Object ← [nolit,CommandType[ASingleStep]];
arfree:CommandType Object ← [nolit,CommandType[AFreeRun]];
sstep:CommandType Object ← [nolit,CommandType[SingleStep]];
rfree:CommandType Object ← [nolit,CommandType[FreeRun]];
Interrupt:StringType Object ←JaMLiteralDefs.MakeStringObject[".interrupt"L];
stepflag:BOOLEAN ← FALSE;
astepflag:BOOLEAN ← FALSE;
reptc:CommandType Object←[nolit,CommandType[CRept]];
loopc:CommandType Object←[nolit,CommandType[CLoop]];

RegisterCommand[".if"L,If];
RegisterCommand[".ifelse"L,IfElse];
RegisterCommand[".rept"L,Rept];
RegisterCommand[".loop"L,Loop];
RegisterCommand[".exit"L,Exit];
RegisterCommand[".stop"L,Stop];
RegisterCommand[".interrupt"L,Stop];
RegisterCommand[".exec"L,Exec];
RegisterCommand[".singlestep"L,SingleStep];
RegisterCommand[".runfree"L,FreeRun];


END.
```

```
--file JaMIO.mesa
--Written by John Warnock, January, 1979.
--Last Updated: April 11, 1979  2:16 PM by MN

DIRECTORY

EdStreamDefs: FROM "EdStreamDefs",
AltoDefs: FROM "AltoDefs",
RectangleDefs: FROM "RectangleDefs",
WindowDefs: FROM "WindowDefs",
DisplayDefs: FROM "DisplayDefs",
JaMMasterDefs: FROM "JaMMasterDefs",
JaMFnsDefs: FROM "JaMFnsDefs",
JaMStackDefs: FROM "JaMStackDefs",
JaMTypeChkDefs: FROM "JaMTypeChkDefs",
JaMExecDefs: FROM "JaMExecDefs",
JaMLiteralDefs: FROM "JaMLiteralDefs",
JaMVMDefs: FROM "JaMVMDefs",
JaMControlDefs: FROM "JaMControlDefs",
InlineDefs: FROM "InlineDefs",
JaMIODefs: FROM "JaMIODefs",
IODefs: FROM "IODefs",
SegmentDefs: FROM "SegmentDefs",
StreamDefs: FROM "StreamDefs";

JaMIO: PROGRAM
IMPORTS  EdStreamDefs,RectangleDefs,WindowDefs,DisplayDefs,
   JaMFnsDefs,IODefs,JaMTypeChkDefs,
   JaMLiteralDefs,JaMExecDefs,
   JaMVMDefs,JaMControlDefs,JaMStackDefs,
   SegmentDefs,StreamDefs
EXPORTS JaMIODefs
SHARES StreamDefs =
BEGIN
OPEN JaMMasterDefs,IODefs,JaMTypeChkDefs,JaMLiteralDefs,
JaMVMDefs,JaMControlDefs,JaMIODefs,JaMStackDefs,StreamDefs,InlineDefs;

-- RdLine and WrtString get and put StringType Objects on the stack.
ReadLine:PUBLIC PROCEDURE =
BEGIN
c:CHARACTER;
i:CARDINAL←0;
frame:Frame ←GetCurrentFrame[];
stringob:StringType Object←[lit,StringType[.,]];
streamob: StreamType Object ← DescStreamType[Pop[frame.opstk]];
[stringob.Address,stringob.Offset] ←AllocateCharsVM[0];
DO
IF streamob.SHandle.endof[streamob.SHandle] THEN
   BEGIN
   IF i = 0 THEN
      BEGIN
      JaMFnsDefs.PushBoolean[FALSE];
      streamob.SHandle.destroy[streamob.SHandle];
      RETURN;
      END
   ELSE
      BEGIN
      stringob.Length←i;
      [.]←AllocateCharsVM[i];
      Push[stringob,frame.opstk];
      JaMFnsDefs.PushBoolean[TRUE];
      RETURN;
      END;
   END
ELSE
   BEGIN
   c←streamob.SHandle.get[streamob.SHandle];
   IF c = 15C THEN
      BEGIN
      stringob.Length←i;
      [.]←AllocateCharsVM[i];
      Push[stringob,frame.opstk];
      JaMFnsDefs.PushBoolean[TRUE];
      RETURN;
      END;
   PutCharVM[c,stringob.Address,stringob.Offset,i];
```

```
   i←i+1;
END;
ENDLOOP;
END;

WrtString:PUBLIC PROCEDURE [stack: Stack]=
BEGIN
i:CARDINAL;
stringob: StringType Object ← DescStringType[Pop[stack]];
stream: StreamHandle ← DescStreamType[Pop[stack]].SHandle;
FOR i IN [0..stringob.Length)
DO
  stream.put[stream,GetCharVM[stringob.Address,stringob.Offset,i]];
ENDLOOP;
END;


WriteByteStream:PUBLIC PROCEDURE  =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
WrtString[frame.opstk];
END;



Print:PUBLIC PROCEDURE  =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
StreamLit[GetOutputStream[],frame.opstk];
Exch[frame.opstk];
WrtString[frame.opstk];
END;

Run:PUBLIC PROCEDURE  =
BEGIN
ob: Object;
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
JaMFnsDefs.PushInteger[1];
NByteStream[];
ob ← Pop[frame.opstk];
ob.litflag ← nolit;
Push[ob,frame.execstk];
END;


-- NByteStream, given the file name and options, returns a stream on the
-- stack.

NByteStream: PUBLIC PROCEDURE =
BEGIN
s:STRING ← [256];
frame:Frame ←JaMControlDefs.GetCurrentFrame[];
i:INTEGER ← JaMFnsDefs.PopInteger[];
ob:StringType Object ← DescStringType[Pop[frame.opstk]];
strm: StreamDefs.StreamHandle;
IF ob.Length > 256 THEN JaMExecDefs.JaMError[LongFileName,TRUE];
GetCharsVM[ob.Address,ob.Offset,@s.text,0,ob.Length];
s.length ← ob.Length;
strm ← EdStreamDefs.CreateBufferedStream[NewByteStream[s,i !
   SegmentDefs.FileNameError => JaMExecDefs.JaMError[BadFileName,TRUE]]];
Push[[lit,StreamType[strm]],frame.opstk];
END;


NKeyStream: PUBLIC PROCEDURE =
BEGIN OPEN JaMFnsDefs;
height: RectangleDefs.yCoord ← PopInteger[];
y0: RectangleDefs.yCoord ← PopInteger[];
width: RectangleDefs.xCoord ← PopInteger[];
x0: RectangleDefs.xCoord ← PopInteger[];
name: STRING ← [100];
strm: StreamHandle;
PopString[name ! IODefs.LineOverflow =>
      BEGIN
      name ← "anon";
      CONTINUE;
```

```
      END];
strm ← EdStreamDefs.CreateEditedStream[NIL, IODefs.ESC,
   name, BitmapHandle, x0, width, y0, height];
Push[[lit,StreamType[strm]],JaMControlDefs.GetCurrentFrame[].opstk];
END;


DestroyStream: PUBLIC PROCEDURE =
BEGIN
s:StreamHandle ← DescStreamType[Pop[JaMControlDefs.GetCurrentFrame[].opstk]].SHandle;
s.destroy[s];
END;

SetOutput: PUBLIC PROCEDURE =
BEGIN
IODefs.SetOutputStream[JaMFnsDefs.PopStream[]];
END;

GetOutput: PUBLIC PROCEDURE =
BEGIN
streamob: StreamType Object ← [lit,StreamType[IODefs.GetOutputStream[]]];
JaMFnsDefs.PushObject[streamob];
END;

   BitmapHeightDots: CARDINAL=350;
   BitmapWidthDots:  CARDINAL=512;
   BitmapWidthWords: CARDINAL =
      (BitmapWidthDots+15)/16+(((BitmapWidthDots+15)/16) MOD 2);
   BitmapAreaWords: CARDINAL = BitmapWidthWords*BitmapHeightDots;
   BitmapAreaPages: CARDINAL =
      (BitmapAreaWords+AltoDefs.PageSize-1)/AltoDefs.PageSize;
   Bitmap: RectangleDefs.BMptr;
   BitmapHandle: PUBLIC RectangleDefs.BMHandle ← NIL;

   SetUpScreen: PROCEDURE =    -- Set up centered bitmap display
   BEGIN OPEN RectangleDefs;
      W1: WindowDefs.WindowHandle;
      edstrm: StreamDefs.StreamHandle;
      DisplayDefs.DisplayOff[white];
      START RectanglesB[BitmapAreaPages,BitmapWidthWords];
      BitmapHandle ← GetDefaultBitmap[];
      BitmapHandle.indenting ← (38-BitmapWidthWords)/2;
      [] ← UpdateBitmap[BitmapHandle];
      Bitmap ← BitmapHandle.addr;

      START WindowDefs.WindowsB["JaM.typescript"];
      W1 ← WindowDefs.GetCurrentDisplayWindow[];
      edstrm ←
         EdStreamDefs.CreateEditedStream[W1,IODefs.ESC,NIL,NIL,0,0,0,0];
      IODefs.SetOutputStream[edstrm];
      IODefs.SetInputStream[edstrm];
      WindowDefs.RepaintDisplayWindows[BitmapHandle];
   END;


--START Code--

LongFileName:StringType Object ← MakeStringObject[".longname"];
BadFileName:StringType Object ← MakeStringObject[".badname"];

SetUpScreen[];

RegisterCommand[".print"L,Print];
RegisterCommand[".readline"L,ReadLine];
RegisterCommand[".writebytes"L,WriteByteStream];
RegisterCommand[".bytestream"L,NByteStream];
RegisterCommand[".keystream"L,NKeyStream];
RegisterCommand[".killstream"L,DestroyStream];
RegisterCommand[".run"L,Run];
RegisterCommand[".setoutput"L,SetOutput];
RegisterCommand[".getoutput"L,GetOutput];

END.
```

```
--File: JaMLiteral.mesa
--Written by: John Warnock, January, 1979

DIRECTORY

JaMMasterDefs: FROM "JaMMasterDefs",
JaMStackDefs: FROM "JaMStackDefs",
StreamDefs: FROM "StreamDefs",
JaMVMDefs: FROM "JaMVMDefs",
JaMLiteralDefs: FROM "JaMLiteralDefs";

JaMLiteral: PROGRAM
IMPORTS
JaMLiteralDefs,
JaMStackDefs,
JaMVMDefs
EXPORTS
JaMLiteralDefs =

--The following routines implement a set of literal building routines. The resulting constructed litera
**l is pushed on the operand stack.

BEGIN
OPEN JaMMasterDefs,JaMStackDefs,JaMLiteralDefs,JaMVMDefs,StreamDefs;

IntegerLit: PUBLIC PROCEDURE [i:INTEGER,s:Stack] =
BEGIN
Push[Object[lit,IntegerType[i]],s];
END;

LongIntegerLit: PUBLIC PROCEDURE [l:LONG INTEGER,s:Stack] =
BEGIN
Push[Object[lit,LongIntegerType[l]],s];
END;

RealLit: PUBLIC PROCEDURE [r:REAL,s:Stack] =
BEGIN
Push[Object[lit,RealType[r]],s];
END;

BooleanLit: PUBLIC PROCEDURE [b:BOOLEAN,s:Stack] =
BEGIN
Push[Object[lit,BooleanType[b]],s];
END;

StringLit: PUBLIC PROCEDURE [string:STRING,s:Stack] =
BEGIN
st:StringType Object ←MakeStringObject[string];
st.litflag ← lit;
Push[st,s];
END;

MakeStringObject: PUBLIC PROCEDURE [string:STRING]
      RETURNS [ob:StringType Object] =
BEGIN
ob← [nolit,StringType[,,]];
[ob.Address,ob.Offset]←AllocateCharsVM[string.length];
ob.Length←string.length;
PutCharsVM[ob.Address,ob.Offset,@string.text,0,string.length];
END;

StreamLit: PUBLIC PROCEDURE [stream:StreamDefs.StreamHandle,s:Stack] =
BEGIN
Push[Object[lit,StreamType[stream]],s];
END;


END.
```

```
--File: JaMScanner.mesa
--Written by: John Warnock, March 7, 1979
--Last updated: April 6, 1979  12:17 PM

DIRECTORY

JaMMasterDefs: FROM "JaMMasterDefs",
JaMLiteralDefs: FROM "JaMLiteralDefs",
JaMScannerDefs: FROM "JaMScannerDefs",
JaMControlDefs: FROM "JaMControlDefs",
JaMExecDefs: FROM "JaMExecDefs",
JaMStackDefs: FROM "JaMStackDefs",
JaMVMDefs: FROM "JaMVMDefs",
StreamDefs: FROM "StreamDefs",
InlineDefs: FROM "InlineDefs",
Mopcodes: FROM "mopcodes",
FloatDefs: FROM "FloatDefs",
IODefs: FROM "IODefs";


--This program implements a very simple lexical scanner which works
--in the following way. When the routine "StreamToken" or "StringToken" is called
--then the first token is returned on the top of stack followed by the
--remainder of the string at next-on-stack.


-- The implementation strategy is based on a state transition table.
-- The indexes into the table are: "charclass" and "state". These two indexes
-- determine a procedure which is called to do the appropriate thing on a
-- given state transition.


JaMScanner: PROGRAM
IMPORTS JaMLiteralDefs,JaMScannerDefs,JaMExecDefs,JaMControlDefs,
FloatDefs,JaMStackDefs,JaMVMDefs
EXPORTS JaMScannerDefs =

BEGIN
OPEN JaMMasterDefs,JaMLiteralDefs,JaMStackDefs,JaMScannerDefs,
InlineDefs,JaMLiteralDefs,FloatDefs,JaMVMDefs;

-- StreamToken is the version of the scanner that takes a stream as its
-- input.

StreamToken: PUBLIC PROCEDURE [streamobject: StreamType Object,sStack:Stack] RETURNS [BOOLEAN] =
   BEGIN
   -- This is the body of the stream token routine.
   charcount ← 0;
   ivalue ← 0;
   token ←FALSE;
   NameFlag ← FALSE;
   positive←TRUE;
   streamsource←streamobject;
   state ← NuetSt;
   stack←sStack;
   string←[streamsource.litflag,StringType[0,0,]];
   DO
   IF streamsource.SHandle.endof[streamsource.SHandle]
      THEN
      BEGIN
      charclass ← 9;
      streamsource.SHandle.destroy[streamsource.SHandle];
      END
      ELSE
      BEGIN
      char ← streamsource.SHandle.get[streamsource.SHandle];
      charclass ← CharClass[LOOPHOLE[char]];
      END;
   StreamDispatch [state][charclass][];
   IF token THEN RETURN [NameFlag];
   ENDLOOP;
   END;


-- StringToken is the version of the scanner that takes a string as its
```

```
-- input.

StringToken: PUBLIC PROCEDURE [stringobject:StringType Object,sStack:Stack] RETURNS [BOOLEAN] =
   BEGIN
   -- This is the body of the string token routine.
   curcnt:CARDINAL←0;
   tsoff:CARDINAL←4;
   tempstring:STRING ← [8];
   charcount ← 0;
   ivalue ← 0;
   token ←FALSE;
   NameFlag ← FALSE;
   positive←TRUE;
   state ← NuetSt;
   stringsource←stringobject;
   stack←sStack;
   string←[stringsource.litflag,StringType[0,0,]];
   GetCharsVM[stringsource.Address,stringsource.Offset,@tempstring.text,0,8];
   DO
   IF  charcount >= stringsource.Length
      THEN charclass ← 9
      ELSE
   BEGIN
   -- the buffering scheme implemented here is not logically
   -- necessary but is here for efficiency.
   char ← tempstring[curcnt];
   IF (curcnt←curcnt+1) >= 8 THEN
    BEGIN
    GetCharsVM[stringsource.Address+tsoff,stringsource.Offset,@tempstring.text,0,8];
    tsoff←tsoff+4;
    curcnt←0;
    END;
   charclass ← CharClass[LOOPHOLE[char]];
   END;
   charcount ← charcount + 1;
   StringDispatch [state][charclass][];
   IF token THEN RETURN [NameFlag];
   ENDLOOP;
   END;


CR:CHARACTER = IODefs.CR;
stringsource:StringType Object;
streamsource:StreamType Object;
stack:Stack;

ClassRange:TYPE = [0..10];
State:TYPE= [0..7];

CharClass:ARRAY [0..127] OF ClassRange←
[4,4,4,4,4,4,4,4,4,           -- NUL through ↑H
 0,0,            -- ↑I(TAB), ↑J(LF)
 4,4,            -- ↑K,↑L
 8,              -- ↑M (CR) Delim. and end of Bravo fld.
 4,4,4,4,4,4,4,4,4,4,4,4,       -- ↑N through ↑Y
 7,              -- ↑Z (bravo delimiter)
 10,             -- ESC
 4,4,4,4,              -- 34C through 37C
 0,              -- SPACE (SP)
 4,4,4,4,4,4,4,           -- 41C through 47C
 5,              -- '( (Left Paren)
 6,              -- ') (Right Paren)
 4,              -- 52C
 1,              -- '+ (PLUS)
 0,              -- ', (COMMA)
 1,              -- '- (MINUS)
 2,              -- '. (PERIOD)
 4,              -- 57C
 3,3,3,3,3,3,3,3,3,3,          -- '0 through '9 (NUMERIC)
 4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
 4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4];   --The rest: 72C through 177C
**

NuetSt:State = 0;
PreNum:State = 1;
```

```
PreFrt:State = 2;
ValInt:State = 3;
ValFrt:State = 4;
Ident :State = 5;
StrLit:State = 6;
BravoS:State = 7;

StringDispatch: ARRAY State OF ARRAY ClassRange OF PROCEDURE←

   --Delm, +- ,  . ,Numb,Othr, (  ,  ) , ↑Z , CR ,EOS  ,ESC--

--NuetSt--  [[Null,PrNm,Frc1,V11I,Nam1,StBl,SErr,BrvF,Null,RetN,Null],
--PreNum--   [NsR0,Name,Frac,VlIn,Name,NsR1,SErr,NsR1,NsR0,NsR1,NsR0],
--PreFrt--   [NsR0,Name,Name,V1F2,Name,NsR1,SErr,NsR1,NsR0,NsR1,NsR0],
--ValInt--   [NIR0,Name,V1F1,VlIn,Name,NIR1,SErr,NIR1,NIR0,NIR1,NIR0],
--ValFrt--   [NFR0,Name,Name,VlFr,Name,NFR1,SErr,NFR1,NFR0,NFR1,NFR0],
--Ident --   [NMR0,Name,Name,Name,Name,NMR1,SErr,NMR1,NMR0,NMR1,NMR0],
--StrLit--   [Null,Null,Null,Null,Null,StC1,CDR0,BrvF,Null,SErr,Null],
--BravoS--   [Null,Null,Null,Null,Null,Null,Null,Null,NNR0,NNR0,Null]];

-- The following are the action routines associated with the above state transitions.


Null:PROCEDURE=
BEGIN
END;

PrNm:PROCEDURE=
BEGIN
state←PreNum;
valnum ← FALSE;
Mark[];
IF char= '- THEN positive ← FALSE;
END;

Frac:PROCEDURE=
BEGIN
state←PreFrt;
dval←rval;
rvalue ← ivalue;
valnum ← FALSE;
END;

Frc1:PROCEDURE=
BEGIN
Mark[];
Frac[];
END;

V1F1:PROCEDURE=
BEGIN
state←ValFrt;
rvalue←ivalue;
END;

V1F2:PROCEDURE=
BEGIN
state←ValFrt;
rvalue←ivalue;
valnum ← TRUE;
AppendFractDigit[char];
END;

VlFr:PROCEDURE=
BEGIN
valnum ← TRUE;
AppendFractDigit[char];
END;

VlIn:PROCEDURE=
BEGIN
state←ValInt;
valnum ← TRUE;
IF ~ AppendDigit[char] THEN ConvertOverflow[];
END;
```

```
V11I:PROCEDURE=
BEGIN
Mark[];
VlIn[];
END;

StB1:PROCEDURE=
BEGIN
state←StrLit;
nestcount←1;
Mark[];
END;

NsR0:PROCEDURE=
BEGIN
ReturnStringRemainder[0];
ReturnStringToken[];
END;

NsR1:PROCEDURE=
BEGIN
ReturnStringRemainder[1];
ReturnStringToken[];
END;

NMR0:PROCEDURE=
BEGIN
ReturnStringRemainder[0];
ReturnStringToken[];
END;

NMR1:PROCEDURE=
BEGIN
ReturnStringRemainder[1];
ReturnStringToken[];
END;

Name:PROCEDURE=
BEGIN
state←Ident;
END;

Nam1:PROCEDURE=
BEGIN
state←Ident;
Mark[];
END;

BrvF:PROCEDURE=
BEGIN
savestate←state;
state←BravoS;
END;

NIR0:PROCEDURE=
BEGIN
ReturnStringRemainder[0];
ReturnNumber[];
END;

NIR1:PROCEDURE=
BEGIN
ReturnStringRemainder[1];
ReturnNumber[];
END;


NFR0:PROCEDURE=
BEGIN
ReturnStringRemainder[0];
ReturnReal[];
END;

NFR1:PROCEDURE=
BEGIN
ReturnStringRemainder[1];
```

```
ReturnReal[];
END;


StC1:PROCEDURE=
BEGIN
nestcount ← nestcount +1;
END;


CDR0:PROCEDURE=
BEGIN
nestcount ← nestcount - 1;
IF nestcount = 0 THEN
   BEGIN
   ReturnStringRemainder[0];
   ReturnStringLitToken[];
   END;
END;

NNR0:PROCEDURE=
BEGIN
state←savestate;
END;

RetN:PROCEDURE=
BEGIN
ReturnNull[];
END;

SErr:PROCEDURE=
BEGIN
BadSyntax[];
END;


StreamDispatch: ARRAY State OF ARRAY ClassRange OF PROCEDURE←

   --Delm , +-  ,  .  ,Numb ,Othr , (   ,  )  , ↑Z  , CR  ,EOS  ,ESC --

--NuetSt--  [[NullS,PrNmS,Frc1S,Vl1IS,Nam1S,StBlS,SErrS,BrvFS,NsPsS,RetNS,NullS],
--PreNum--   [NsR0S,NameS,FracS,VlInS,NameS,NsR1S,SErrS,NsR1S,NsR0S,NsRS,NsR0S],
--PreFrt--   [NsR0S,NameS,NameS,VlF2S,NameS,NsR1S,SErrS,NsR1S,NsR0S,NsRS,NsR0S],
--ValInt--   [NIR0S,NameS,VlF1S,VlInS,NameS,NIR1S,SErrS,NIR1S,NIR0S,NIRS,NIR0S],
--ValFrt--   [NFR0S,NameS,NameS,VlFrS,NameS,NFR1S,SErrS,NFR1S,NFR0S,NFRS,NFR0S],
--Ident --   [NMR0S,NameS,NameS,NameS,NameS,NMR1S,SErrS,NMR1S,NMR0S,NMRS,NMR0S],
--StrLit--   [StLtS,StLtS,StLtS,StLtS,StLtS,StC1S,CDR0S,BrvFS,StLtS,SErr,NullS],
--BravoS--   [NullS,NullS,NullS,NullS,NullS,NullS,NullS,NullS,NNR0S,NNR0S,NullS]];

-- The following are the action routines associated with the above state transitions.

NullS: PROCEDURE =
BEGIN
END;


NsPsS:PROCEDURE=
BEGIN
ReturnStreamRemainder[0];
END;

PrNmS:PROCEDURE=
BEGIN
state←PreNum;
valnum ← FALSE;
SMark[];
StoreChar[];
IF char= '- THEN positive ← FALSE;
END;

FracS:PROCEDURE=
BEGIN
state←PreFrt;
rvalue ← ivalue;
dval←rval;
valnum ← FALSE;
```

```
StoreChar[];
END;

Frc1S:PROCEDURE=
BEGIN
SMark[];
FracS[];
END;

V1F2S:PROCEDURE=
BEGIN
state←ValFrt;
rvalue←ivalue;
valnum ← TRUE;
AppendFractDigit[char];
StoreChar[];
END;

V1F1S:PROCEDURE=
BEGIN
state←ValFrt;
rvalue←ivalue;
StoreChar[];
END;

V1FrS:PROCEDURE=
BEGIN
valnum ← TRUE;
AppendFractDigit[char];
StoreChar[];
END;

V1InS:PROCEDURE=
BEGIN
state←ValInt;
valnum ← TRUE;
StoreChar[];
IF ~ AppendDigit[char] THEN ConvertOverflow[];
END;

V11IS:PROCEDURE=
BEGIN
SMark[];
V1InS[];
END;

StB1S:PROCEDURE=
BEGIN
state←StrLit;
nestcount←1;
SMark[];
END;

NsRS:PROCEDURE=
BEGIN
ReturnStreamToken[];
END;

NsR0S:PROCEDURE=
BEGIN
ReturnStreamRemainder[0];
ReturnStreamToken[];
END;

NsR1S:PROCEDURE=
BEGIN
ReturnStreamRemainder[1];
ReturnStreamToken[];
END;

NMRS:PROCEDURE=
BEGIN
ReturnStreamToken[];
END;

NMR0S:PROCEDURE=
```

```
BEGIN
ReturnStreamRemainder[0];
ReturnStreamToken[];
END;

NMR1S:PROCEDURE=
BEGIN
ReturnStreamRemainder[1];
ReturnStreamToken[];
END;

NameS:PROCEDURE=
BEGIN
state←Ident;
StoreChar[];
END;

Nam1S:PROCEDURE=
BEGIN
state←Ident;
SMark[];
StoreChar[];
END;

BrvFS:PROCEDURE=
BEGIN
savestate←state;
state←BravoS;
END;

NIRS:PROCEDURE=
BEGIN
ReturnNumber[];
END;

NIR0S:PROCEDURE=
BEGIN
ReturnStreamRemainder[0];
ReturnNumber[];
END;

NIR1S:PROCEDURE=
BEGIN
ReturnStreamRemainder[1];
ReturnNumber[];
END;

NFRS:PROCEDURE=
BEGIN
ReturnReal[];
END;

NFR0S:PROCEDURE=
BEGIN
ReturnStreamRemainder[0];
ReturnReal[];
END;

NFR1S:PROCEDURE=
BEGIN
ReturnStreamRemainder[1];
ReturnReal[];
END;

StLtS:PROCEDURE=
BEGIN
StoreChar[];
END;


StC1S:PROCEDURE=
BEGIN
nestcount ← nestcount +1;
StoreChar[];
END;
```

```
CDROS:PROCEDURE=
BEGIN
nestcount ← nestcount - 1;
IF nestcount = 0 THEN
   BEGIN
   ReturnStreamRemainder[0];
   ReturnStreamLitToken[];
   END
   ELSE StoreChar[];
END;

NNROS:PROCEDURE=
BEGIN
state←savestate;
END;

RetNS:PROCEDURE=
BEGIN
ReturnNull[];
END;

SErrS:PROCEDURE=
BEGIN
BadSyntax[];
END;


--These support the above dispatch functions:


ReturnNumber: PROCEDURE   =
BEGIN
IF ~positive THEN ivalue ← -ivalue;
IF ivalue < 32768 AND ivalue >= -32768
      THEN BEGIN
          Push[[lit,IntegerType[LowHalf[ivalue]]],stack];
          END
      ELSE BEGIN
         Push[[lit,LongIntegerType[ivalue]],stack];
         END;
token←TRUE;
NameFlag←FALSE;
END;

ReturnReal: PROCEDURE   =
BEGIN
Push[[lit,RealType[IF ~positive THEN -rvalue ELSE rvalue]],stack];
token←TRUE;
END;

LowHalf: PROCEDURE [LONG INTEGER] RETURNS [INTEGER] =
    MACHINE CODE BEGIN Mopcodes.zPOP END;


ReturnNull: PROCEDURE =
BEGIN
ob:NullType Object;
ob ← [nolit,NullType[]];
Push[ob,stack];
token←TRUE;
END;

Mark: PROCEDURE =
BEGIN
savecount←charcount;
END;

SMark: PROCEDURE =
BEGIN
[string.Address,string.Offset]←AllocateCharsVM[0];
string.Length←0;
END;

StoreChar: PROCEDURE =
BEGIN
```

```
PutCharVM[char,string.Address,string.Offset,string.Length];
string.Length ← string.Length+1;
END;

ReturnStreamRemainder:PROCEDURE [lst:[0..1] ] =
BEGIN
IF lst = 1 THEN streamsource.SHandle.putback[streamsource.SHandle,char];
Push[streamsource,stack];
--IF char = CR THEN Push[prompt,stack];
END;

ReturnStringRemainder:PROCEDURE [lst:[0..1] ] =
BEGIN
string.Length ← stringsource.Length-(charcount-lst);
string.Address ← stringsource.Address+BITSHIFT[stringsource.Offset+charcount-lst,-1];
string.Offset ← BITAND[stringsource.Offset+charcount-lst,1];
Push[string,stack];
RETURN;
END;

ReturnStreamToken:PROCEDURE=
BEGIN
[,]←AllocateCharsVM[string.Length];
Push[string,stack];
NameFlag ← TRUE;
token←TRUE;
END;

ReturnStreamLitToken:PROCEDURE=
BEGIN
st:StringType Object← [lit,StringType[
   Offset: string.Offset,
   Length: string.Length,
   Address: string.Address]];
[,]←AllocateCharsVM[string.Length];
Push[st,stack];
token←TRUE;
END;

ReturnStringToken:PROCEDURE=
BEGIN
st:StringType Object ← [nolit,StringType[,,]];
st.Length ← charcount - savecount;
st.Address ← stringsource.Address + BITSHIFT[stringsource.Offset + savecount-1,-1];
st.Offset ← BITAND[stringsource.Offset + savecount-1,1];
Push[st,stack];
NameFlag ← TRUE;
token←TRUE;
END;

ReturnStringLitToken:PROCEDURE=
BEGIN
st:StringType Object ← [lit,StringType[,,]];
st.Length ← charcount - savecount-1;
st.Address ← stringsource.Address + BITSHIFT[stringsource.Offset + savecount,-1];
st.Offset ← BITAND[stringsource.Offset + savecount,1];
Push[st,stack];
token←TRUE;
END;

AppendDigit: PROCEDURE [d:CHARACTER] RETURNS [b:BOOLEAN] =
BEGIN
MaxVal:LONG INTEGER = 214748364;
MaxD: CARDINAL = 7;
v:INTEGER ← LOOPHOLE[d - '0];
IF (ivalue < MaxVal OR v <= MaxD) THEN
BEGIN
ivalue ← ivalue*10 + v; RETURN[TRUE];
END
ELSE
RETURN[FALSE];
END;

AppendFractDigit: PROCEDURE [d:CHARACTER] =
BEGIN
v:REAL ← d - '0;
```

```
rvalue ← rvalue + v*dval;
dval←dval*rval;
END;

ConvertOverflow:PROCEDURE =
BEGIN
JaMExecDefs.JaMError[OverFlowErr,TRUE];
END;

BadSyntax:PROCEDURE =
BEGIN
JaMExecDefs.JaMError[SyntaxErr,TRUE];
END;

Token: PUBLIC PROCEDURE =
   BEGIN
   frame:Frame ←JaMControlDefs.GetCurrentFrame[];
   ob: Object ← Pop[frame.opstk];
   WITH dob:ob SELECT FROM
   StringType  => BEGIN
      strng:StringType Object ← dob;
      []←StringToken[strng,frame.opstk];
      NullChk[];
      END;
   StreamType  =>BEGIN
      strm:StreamType Object ← dob;
      []←StreamToken[strm,frame.opstk];
      NullChk[];
      END;
   ENDCASE  =>BEGIN
      JaMExecDefs.JaMError[TypeErr,TRUE];
      END;
   END;

NullChk: PUBLIC PROCEDURE =
   BEGIN
   frame:Frame ←JaMControlDefs.GetCurrentFrame[];
   ob: Object ← Top[frame.opstk];
   WITH dob:ob SELECT FROM
   NullType => BEGIN
      []←Pop[frame.opstk];
      Push[false,frame.opstk];
      END;
   ENDCASE  =>BEGIN
      Push[true,frame.opstk];
      END;
   END;
-- Some global variables known to the routines.

rvalue:REAL;      -- initial value for scanned real.
ivalue:LONG INTEGER ← 0;   -- initial value for scanned integer.
string:StringType Object;
TypeErr:StringType Object;
SyntaxErr:StringType Object;
OverFlowErr:StringType Object;
--prompt:StringType Object;
true:BooleanType Object;
false:BooleanType Object;
charclass:ClassRange;   --Current Character Class
state:State;       --Current State
savestate:State;  --Saved state for Bravo format
nestcount:INTEGER ←0;       --Balanced parentheses count.
char:CHARACTER ← 40C;   --Set Up Initially with SP.
charcount,savecount:CARDINAL ← 0; --Character positions in string.
positive:BOOLEAN← TRUE; --the sign of the current number being scanned.
dval:REAL←StringToFloat[".1"];   --constant used for real conversion.
rval:REAL=dval;
valnum:BOOLEAN ← TRUE;  --valid number indicator.
token:BOOLEAN ←TRUE;
NameFlag:BOOLEAN ← TRUE;
--Set up prompt string object
TypeErr ← MakeStringObject[".typechk"];
SyntaxErr ← MakeStringObject[".syntaxerr"];
OverFlowErr ←MakeStringObject[".overflow"];
true ← [lit,BooleanType[TRUE]];
false ← [lit,BooleanType[FALSE]];
```

```
--prompt ← MakeStringObject[".prompt"];
JaMControlDefs.RegisterCommand[".token",Token];
END.
```

```
--file JaMStack.mesa
--Written by John Warnock, December 1978.
-- Last Edit: March 19, 1979  12:51 PM by MN

DIRECTORY

JaMMasterDefs: FROM "JaMMasterDefs",
JaMFnsDefs: FROM "JaMFnsDefs",
JaMVMDefs: FROM "JaMVMDefs",
JaMControlDefs: FROM "JaMControlDefs",
JaMExecDefs: FROM "JaMExecDefs",
JaMLiteralDefs: FROM "JaMLiteralDefs",
SystemDefs: FROM "SystemDefs" USING [AllocateResidentPages],
JaMStackDefs: FROM "JaMStackDefs",
StreamDefs: FROM "StreamDefs";

JaMStack: PROGRAM
IMPORTS JaMFnsDefs,JaMStackDefs,JaMControlDefs,JaMVMDefs,JaMLiteralDefs,JaMExecDefs,
SystemDefs EXPORTS JaMStackDefs =

BEGIN
OPEN JaMMasterDefs,JaMFnsDefs,JaMControlDefs,JaMLiteralDefs,JaMExecDefs,JaMStackDefs,SystemDefs;
StackAreaPtr: TYPE = POINTER TO StackArea;
StackArea: TYPE = ARRAY [0..stackcount) OF StackEntry;

i:INTEGER [0..stackcount);
stackcount: INTEGER = 1000;
FreeListPtr:StackLink;

--"Push" pushes a descriptor onto the current operand stack.

Push: PUBLIC PROCEDURE [ob:Object,stack:Stack] =
BEGIN
TempPointer:StackLink;
IF (TempPointer ← FreeListPtr.nextEntry) = NIL THEN
BEGIN
-- A stack overflow, in the JaM machine, is a serious error
-- For this reason all three stacks are packaged into arrays
-- and stored on the operand stack.
   obj:Object;
   frame: Frame ← JaMControlDefs.GetCurrentFrame[];
   ostk:ArrayType Object←MakeStackArray[frame.opstk];
   dstk:ArrayType Object←MakeStackArray[frame.dictstk];
   estk:ArrayType Object←MakeStackArray[frame.execstk];
Push[ostk, frame.opstk];
Push[dstk, frame.opstk];
Push[estk, frame.opstk];
JaMVMDefs.GetWordsVM[dstk.ArrayPtr,@obj,SIZE[Object]];
Push[obj,frame.dictstk];
JaMError[StkOvrFlw,FALSE];
END;
FreeListPtr↑ ← StackEntry[ob,stack↑];
stack↑ ← FreeListPtr;
FreeListPtr ← TempPointer;
END;

MakeStackArray: PROCEDURE [stack:Stack] RETURNS [ArrayType Object]=
BEGIN
i:CARDINAL ← CountStk[stack];
val: Object;
array: ArrayType Object ←
[lit,ArrayType[i,JaMVMDefs.AllocateWordsVM[i*SIZE[Object]]]];
j: CARDINAL ← array.Length*SIZE[Object];
THROUGH [0..array.Length)
DO j ← j - SIZE[Object];
   val ← Pop[stack];
   JaMVMDefs.PutWordsVM[array.ArrayPtr + j, @val,SIZE[Object]];
ENDLOOP;
RETURN[array];
END;

--"Pop" returns the descriptor from the top of the operand stk.

Pop: PUBLIC PROCEDURE [stack:Stack] RETURNS [Ob:Object] =
BEGIN
TempPointer:StackLink;
```

```
IF stack↑ = NIL THEN JaMError[StkUndFlw,TRUE];
Ob ← stack.ob;
TempPointer ← stack.nextEntry;
stack.nextEntry ← FreeListPtr;
FreeListPtr ← stack↑;
stack↑ ← TempPointer;
END;


--"Top" returns the descriptor from the top of the operand stk without popping.

Top: PUBLIC PROCEDURE [stack:Stack] RETURNS [Ob:Object] =
BEGIN
IF stack↑ = NIL THEN JaMError[StkUndFlw,TRUE];
Ob ← stack.ob;
END;

--"Dup" Duplicates the top descriptor onto the stack.

Dup: PUBLIC PROCEDURE [stack:Stack] =
BEGIN
TDesc1:Object ← Top [stack];
Push [TDesc1,stack];
END;

--"Exch" Exchanges the top entries on the stack.

Exch: PUBLIC PROCEDURE [stack:Stack] =
BEGIN
TDesc1:Object ← Pop [stack];
TDesc2:Object ← Pop [stack];
Push [TDesc1,stack]; Push [TDesc2,stack];
END;

-- "ClrStk" moves all entries from the stack to the freelist.

ClrStk: PUBLIC PROCEDURE [stack:Stack] =
BEGIN
tptr0:StackLink;
tptr:StackLink ← stack↑;
IF tptr = NIL THEN RETURN;
UNTIL tptr = NIL
DO
tptr0←tptr;
tptr←tptr.nextEntry;
ENDLOOP;
tptr0.nextEntry←FreeListPtr;
FreeListPtr←stack↑;
stack↑ ← NIL;
END;

-- "CopyStk" copies the top n entries on the given stack.

CopyStk:PUBLIC PROCEDURE [ n:CARDINAL,stack:Stack] =
BEGIN
ob:Object;
tptr:StackLink←NIL;
tstk:Stack←@tptr;
tempptr:StackLink ← stack↑;
IF n = 0 THEN RETURN;
THROUGH [0..n)
DO
IF tempptr = NIL THEN BEGIN JaMError[StkUndFlw,TRUE]; END;
Push[tempptr.ob,tstk];
tempptr←tempptr.nextEntry;
ENDLOOP;
UNTIL tstk↑ = NIL
DO
ob←Pop[tstk];
Push[ob,stack];
ENDLOOP;
END;

-- "RollStk" rolls the top n entries on the given stack by k places.
-- k>0 is like pop, k<0 is like push
```

```
RollStk:PUBLIC PROCEDURE [stack: Stack, n: CARDINAL, k: INTEGER] =
BEGIN
curr: StackLink ← stack↑;
first,kth: StackLink;
kk: CARDINAL;
IF n = 0 THEN RETURN;
k ← k MOD n;  --make |k|<n
IF k<0 THEN k ← n+k;  --make k positive
IF k = 0 THEN RETURN;
kk ← k;
--Find relevant entries, checking for underflow but making no changes
first ← curr;
THROUGH [1..kk)
DO IF curr=NIL THEN JaMError[StkUndFlw,TRUE];
   curr ← curr.nextEntry;
ENDLOOP;
kth ← curr;
THROUGH [kk..n)
DO IF curr=NIL THEN JaMError[StkUndFlw,TRUE];
   curr ← curr.nextEntry;
ENDLOOP;
--do it
stack↑ ← kth.nextEntry;
kth.nextEntry ← curr.nextEntry;
curr.nextEntry ← first;
END;


-- "CountStk" counts the entries on the given stack.

CountStk:PUBLIC PROCEDURE [stack:Stack] RETURNS [n:CARDINAL] =
BEGIN
tempptr:StackLink ← stack↑;
n←0;
UNTIL tempptr = NIL
DO
tempptr←tempptr.nextEntry;
n←n+1;
ENDLOOP;
END;

-- "CountToMark" counts the entries on the given stack to the first mark.

CountToMark:PUBLIC PROCEDURE [stack:Stack] RETURNS [n:CARDINAL] =
BEGIN
tempptr:StackLink ← stack↑;
n←0;
UNTIL tempptr = NIL
DO
WITH tempptr.ob SELECT FROM
MarkType => EXIT;
ENDCASE;
tempptr←tempptr.nextEntry;
n←n+1;
ENDLOOP;
END;

ClrToMark:PUBLIC PROCEDURE [stack:Stack]=
BEGIN
ob:Object;
DO
ob←Top[stack];
WITH ob SELECT FROM
MarkType => EXIT;
ENDCASE  => []←Pop[stack];
ENDLOOP;
END;


-- "MoveStkOps" transfers a number of descriptors from one stack
-- to another. This is an internal procedure.

MoveStkOps: PUBLIC PROCEDURE [ From, To:Stack,Count:CARDINAL] =
BEGIN
TPtr:StackLink;
TPtr2:StackLink;
```

```
IF Count <= 0 THEN RETURN;
IF (TPtr ← From↑) = NIL THEN JaMError[StkUndFlw,TRUE];
THROUGH [1..Count)
DO
IF (TPtr ← TPtr.nextEntry) = NIL THEN JaMError[StkUndFlw,TRUE];
ENDLOOP;
TPtr2 ← To↑;
To↑← From↑;
From↑← TPtr.nextEntry;
TPtr.nextEntry ← TPtr2;
END;

-- "StackForall" executes a given mesa procedure for each element on the
-- given stack,and returns true if the procedure returns true.

StackForall:PUBLIC PROCEDURE [ stack:Stack,proc:PROCEDURE[obj:Object]
RETURNS [done:BOOLEAN]] RETURNS [BOOLEAN] =
BEGIN
tptr:StackLink;
tptr ← stack↑;
UNTIL tptr = NIL
DO
IF proc[tptr.ob] THEN RETURN [TRUE];
tptr←tptr.nextEntry;
ENDLOOP;
RETURN [FALSE];
END;


-- The next set of routines make the above routines into intrinsics.

PopOpStk: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
[]←Pop[frame.opstk];
END;

DupOpStk: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
Dup[frame.opstk];
END;

ExchOpStk: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
Exch[frame.opstk];
END;

ClearOpStk: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
ClrStk[frame.opstk];
END;

CopyOpStk: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
i:INTEGER ← JaMFnsDefs.PopInteger[];
IF i < 0  THEN i ← 0;
CopyStk[i,frame.opstk];
END;

RollOpStk: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
k:INTEGER ← JaMFnsDefs.PopInteger[];
n:INTEGER ← JaMFnsDefs.PopInteger[];
RollStk[frame.opstk, n, k];
END;

CountOpStk: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
i:INTEGER ← CountStk[frame.opstk];
```

```
IntegerLit[i,frame.opstk];
END;

CountToMrk: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
i:INTEGER ← CountToMark[frame.opstk];
IntegerLit[i,frame.opstk];
END;

ClearToMrk: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
ClrToMark[frame.opstk];
END;

Mark: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
m:MarkType Object ← [lit,MarkType[]];
Push[m,frame.opstk];
END;

--The next two procedures are used in conjunction with error recovery.

FreeLoc: PUBLIC PROCEDURE RETURNS [sl:StackLink]=
BEGIN
RETURN[FreeListPtr];
END;

RestoreStk: PUBLIC PROCEDURE [sl:StackLink,s:Stack]=
BEGIN
tmp:StackLink;
UNTIL sl = FreeListPtr
DO
IF FreeListPtr = NIL THEN RETURN;
tmp ←FreeListPtr;
FreeListPtr ← FreeListPtr.nextEntry;
tmp.nextEntry ← s↑;
s↑ ← tmp;
ENDLOOP;
END;

--The next definitions are the error objects to be executed by execution
-- control.

StkUndFlw: StringType Object ← MakeStringObject[".stkundflw"];
StkOvrFlw: StringType Object ← MakeStringObject[".stkovrflw"];
TypeChk: StringType Object ← MakeStringObject[".typechk"];

-- The following is the module initialization code.


stackareaptr:StackAreaPtr ← AllocateResidentPages [(SIZE[StackArea]+255)/256];
FOR i IN [0..stackcount)
 DO stackareaptr[i].nextEntry ← @stackareaptr[i+1];
 ENDLOOP;
stackareaptr[stackcount -1].nextEntry ← NIL;
FreeListPtr ← @stackareaptr[0];

END.
```

```
--file JaMStart.mesa
--Written by John Warnock, Feb., 1979.
--Last Edit April 11, 1979  2:17 PM by MN
DIRECTORY

FloatDefs: FROM "FloatDefs",
ImageDefs: FROM "ImageDefs",
IODefs: FROM "IODefs",
JaMIODefs: FROM "JaMIODefs",
JaMMasterDefs: FROM "JaMMasterDefs",
JaMControlDefs: FROM "JaMControlDefs",
JaMExecDefs: FROM "JaMExecDefs",
JaMStringDefs: FROM "JaMStringDefs",
JaMAttributesDefs: FROM "JaMAttributesDefs",
JaMStackDefs: FROM "JaMStackDefs",
JaMMathDefs: FROM "JaMMathDefs",
JaMArrayDefs: FROM "JaMArrayDefs",
JaMLiteralDefs: FROM "JaMLiteralDefs",
JaMInterruptDefs: FROM "JaMInterruptDefs",
StreamDefs: FROM "StreamDefs";

JaMStart: PROGRAM
IMPORTS FloatDefs,ImageDefs,IODefs,JaMIODefs,JaMControlDefs,
   JaMExecDefs,JaMInterruptDefs,JaMMathDefs,JaMLiteralDefs,
   JaMStringDefs,JaMArrayDefs,JaMStackDefs,JaMAttributesDefs =
BEGIN
OPEN JaMMasterDefs,StreamDefs;


-- This is the main control program for JaM. This program creates an edited stream, and then transfers
**execution control.

ImageDefs.MakeImage["JaM.image"];
START FloatDefs.Float;
START JaMControlDefs.JaMControl;
START JaMIODefs.JaMIO;
START JaMAttributesDefs.JaMAttributes;
START JaMMathDefs.JaMMath;
START JaMArrayDefs.JaMArray;
START JaMStringDefs.JaMString;
BEGIN
stream: StreamType Object←[nolit,StreamType[IODefs.GetInputStream[]]];
--prmpt: StringType Object←JaMLiteralDefs.MakeStringObject[".prompt"L];
Title: StringType Object←JaMLiteralDefs.MakeStringObject["( JaM   Version 1.0 April 11, 1979

).print"L];
frame: Frame ← JaMControlDefs.GetCurrentFrame[];
JaMInterruptDefs.InitJaMBreak[];
JaMStackDefs.Push[stream,frame.execstk];
JaMStackDefs.Push[Title,frame.execstk];
DO
   JaMExecDefs.Execute[];
   JaMStackDefs.Push[stream,frame.execstk];
ENDLOOP;
END;

END.
```

```
--File: JaMString.mesa
--Written by: John Warnock, January, 1979

DIRECTORY

DebugRealDefs: FROM "DebugRealDefs",
StringDefs: FROM "StringDefs",
InlineDefs: FROM "InlineDefs",
JaMMasterDefs: FROM "JaMMasterDefs",
JaMFnsDefs: FROM "JaMFnsDefs",
JaMControlDefs: FROM "JaMControlDefs",
JaMExecDefs: FROM "JaMExecDefs",
JaMTypeChkDefs: FROM "JaMTypeChkDefs",
JaMLiteralDefs: FROM "JaMLiteralDefs",
JaMStackDefs: FROM "JaMStackDefs",
JaMStringDefs: FROM "JaMStringDefs",
JaMVMDefs: FROM "JaMVMDefs";

JaMString: PROGRAM
IMPORTS DebugRealDefs,StringDefs,JaMFnsDefs,JaMExecDefs,JaMTypeChkDefs,JaMControlDefs,JaMLiteralDefs,Ja
**MStackDefs,
JaMVMDefs
EXPORTS JaMStringDefs =

--The following routines implement a set of string manipulations.

BEGIN
OPEN DebugRealDefs,StringDefs,JaMLiteralDefs,JaMMasterDefs,
JaMStackDefs,JaMControlDefs,JaMVMDefs;

String: PUBLIC PROCEDURE =
   BEGIN
   frame:Frame ← JaMControlDefs.GetCurrentFrame[];
   i:INTEGER ← JaMFnsDefs.PopInteger[];
   s:StringType Object← [lit,StringType[,,]];
   s.Length ← i;
   [s.Address,s.Offset]←AllocateCharsVM[i];
   Push[s,frame.opstk];
   END;


StringCompare: PUBLIC PROCEDURE [s1,s2: StringType Object] RETURNS[r:JaMStringDefs.Relation] =
BEGIN
c1:CHARACTER;
c2:CHARACTER;
i: CARDINAL;
FOR i IN [0..MIN[s1.Length,s2.Length]) DO
   c1 ←GetCharVM[s1.Address,s1.Offset,i];
   c2 ←GetCharVM[s2.Address,s2.Offset,i];
   IF c1 > c2 THEN RETURN[greater];
   IF c1 < c2 THEN RETURN[less];
ENDLOOP;
IF s1.Length > s2.Length THEN RETURN[greater];
IF s1.Length < s2.Length THEN RETURN[less];
RETURN[equal];
END;

StringSearch: PUBLIC PROCEDURE =
BEGIN
compare:BOOLEAN;
i: CARDINAL;
j: CARDINAL;
frame:Frame ← GetCurrentFrame[];
s:StringType Object ← JaMTypeChkDefs.DescStringType[Pop[frame.opstk]];
t:StringType Object ← JaMTypeChkDefs.DescStringType[Pop[frame.opstk]];
FOR j IN [0..t.Length) DO
IF t.Length - j < s.Length THEN BEGIN Push[t,frame.opstk]; JaMFnsDefs.PushBoolean[FALSE];  RETURN; END;
**
compare ← TRUE;
FOR i IN [0..s.Length) DO
   IF GetCharVM[s.Address,s.Offset,i] # GetCharVM[t.Address,t.Offset,j+i] THEN BEGIN compare ← FALSE; E
**XIT; END;
ENDLOOP;
IF compare THEN
      BEGIN
      sob:StringType Object←[lit,StringType[,,]];
```

```
      sob.Address ← t.Address + InlineDefs.BITSHIFT[t.Offset+j+s.Length,-1];
      sob.Offset ← InlineDefs.BITAND[t.Offset+j+s.Length,1];
      sob.Length ← t.Length-(j+s.Length);
      Push[sob,frame.opstk];
      sob.Address ← t.Address + InlineDefs.BITSHIFT[t.Offset + j,-1];
      sob.Offset ← InlineDefs.BITAND[t.Offset+j,1];
      sob.Length ← s.Length;
      Push[sob,frame.opstk];
      sob.Address ← t.Address;
      sob.Offset ← t.Offset;
      sob.Length ← j;
      Push[sob,frame.opstk];
      JaMFnsDefs.PushBoolean[TRUE];
      RETURN;
      END;
ENDLOOP;
Push[t,frame.opstk];
JaMFnsDefs.PushBoolean[FALSE];
RETURN;
END;


SubString: PUBLIC PROCEDURE =
   BEGIN
   frame:Frame ← JaMControlDefs.GetCurrentFrame[];
   cnt:INTEGER ← JaMFnsDefs.PopInteger[];
   strt:INTEGER ← JaMFnsDefs.PopInteger[];
   s:StringType Object← JaMTypeChkDefs.DescStringType[Pop[frame.opstk]];
   IF cnt < 0 OR strt < 0 OR strt+cnt > s.Length THEN RangeErr[];
   s.Address ← s.Address +InlineDefs.BITSHIFT[s.Offset+strt,-1];
   s.Offset ← InlineDefs.BITAND[s.Offset+strt,1];
   s.Length ← cnt;
   Push[s,frame.opstk];
   END;

PutString: PUBLIC PROCEDURE =
BEGIN
i:CARDINAL;
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
s:StringType  Object ← JaMTypeChkDefs.DescStringType[Pop[frame.opstk]];
strt: INTEGER ← JaMFnsDefs.PopInteger[];
t:StringType  Object ← JaMTypeChkDefs.DescStringType[Pop[frame.opstk]];
IF strt < 0 OR strt +s.Length > t.Length THEN RangeErr[];
FOR i IN [0..s.Length) DO
   PutCharVM[GetCharVM[s.Address,s.Offset,i],t.Address,t.Offset,i+strt];
ENDLOOP;
Push[t,frame.opstk];
END;

ConvertToString: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
ob:Object←Pop[frame.opstk];
s:STRING←[64];
WITH dob:ob SELECT FROM
   IntegerType =>BEGIN
         AppendDecimal[s,dob.IntegerVal];
         StringLit[s,frame.opstk];
         END;
   LongIntegerType   =>BEGIN
         AppendLongNumber[s,dob.LongIntegerVal,10];
         StringLit[s,frame.opstk];
         END;
   RealType =>BEGIN
         AppendRealNumber[s,dob.RealVal];
         StringLit[s,frame.opstk];
         END;
   BooleanType =>BEGIN
         IF dob.BooleanVal THEN Push[true,frame.opstk]
         ELSE Push[false,frame.opstk];
         END;
   StringType  =>BEGIN
         Push[dob,frame.opstk];
         END;
   ENDCASE     =>BEGIN
         Push[novalue,frame.opstk];
```

```
        END;

END;


ConvertIntoString: PUBLIC PROCEDURE  =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
sob:StringType Object ← JaMTypeChkDefs.DescStringType[Pop[frame.opstk]];
ob:Object←Pop[frame.opstk];
s:STRING←[20];
WITH dob:ob SELECT FROM
   IntegerType =>BEGIN
         AppendDecimal[s,dob.IntegerVal];
         IF sob.Length >= s.length THEN
         BEGIN
         PutCharsVM[sob.Address,sob.Offset,@s.text,0,s.length];
         sob.Length ← s.length;
         Push[sob,frame.opstk];
         END
         ELSE
         SizeErr[];
         END;
   LongIntegerType   =>BEGIN
         AppendLongNumber[s,dob.LongIntegerVal,10];
         IF sob.Length >= s.length THEN
         BEGIN
         PutCharsVM[sob.Address,sob.Offset,@s.text,0,s.length];
         sob.Length ← s.length;
         Push[sob,frame.opstk];
         END
         ELSE
         SizeErr[];
         END;
   RealType =>BEGIN
         AppendRealNumber[s,dob.RealVal];
         IF sob.Length >= s.length THEN
         BEGIN
         PutCharsVM[sob.Address,sob.Offset,@s.text,0,s.length];
         sob.Length ← s.length;
         Push[sob,frame.opstk];
         END
         ELSE
         SizeErr[];
         END;
   BooleanType =>BEGIN
         IF dob.BooleanVal THEN Push[true,frame.opstk]            ELSE Push[false,frame.opstk];
         END;
   StringType  =>BEGIN
         Push[dob,frame.opstk];
         END;
   ENDCASE     =>BEGIN
         Push[novalue,frame.opstk];
         END;

END;

SizeErr:PROCEDURE = BEGIN JaMExecDefs.JaMError[sizechk,TRUE]; END;
RangeErr:PROCEDURE = BEGIN JaMExecDefs.JaMError[rangechk,TRUE]; END;

true:StringType Object ← MakeStringObject[".true"];
false:StringType Object ← MakeStringObject[".false"];
novalue:StringType Object ← MakeStringObject["--nostringval--"];
sizechk:StringType Object ← MakeStringObject[".sizechk"];
rangechk:StringType Object ← MakeStringObject[".rangechk"];
--Strings
RegisterCommand[".search"L,StringSearch];
RegisterCommand[".string"L,String];
RegisterCommand[".substring"L,SubString];
RegisterCommand[".putstring"L,PutString];
RegisterCommand[".cvs"L,ConvertToString];
RegisterCommand[".cvis"L,ConvertIntoString];

END.
```

```
--file JaMTypeChk.mesa
--Written by John Warnock, January, 1979.  Edited February 12, 1979  10:06 AM
DIRECTORY

JaMMasterDefs: FROM "JaMMasterDefs",
JaMExecDefs: FROM "JaMExecDefs",
JaMLiteralDefs: FROM "JaMLiteralDefs",
JaMTypeChkDefs: FROM "JaMTypeChkDefs";

JaMTypeChk: PROGRAM
IMPORTS JaMLiteralDefs,JaMExecDefs EXPORTS JaMTypeChkDefs =
BEGIN
OPEN JaMMasterDefs,JaMLiteralDefs,JaMExecDefs,JaMTypeChkDefs;

DescIntegerType: PUBLIC PROCEDURE [ob:Object] RETURNS [iob:IntegerType Object] =
BEGIN
WITH dob:ob SELECT FROM
IntegerType    =>iob←dob;
ENDCASE        =>JaMTypeErr[];
END;

DescLongIntegerType: PUBLIC PROCEDURE [ob:Object] RETURNS [iob:LongIntegerType Object] =
BEGIN
WITH dob:ob SELECT FROM
LongIntegerType   =>iob←dob;
ENDCASE        =>JaMTypeErr[];
END;

DescRealType: PUBLIC PROCEDURE [ob:Object] RETURNS [iob:RealType Object] =
BEGIN
WITH dob:ob SELECT FROM
RealType       =>iob←dob;
ENDCASE        =>JaMTypeErr[];
END;

DescBooleanType: PUBLIC PROCEDURE [ob:Object] RETURNS [iob:BooleanType Object] =
BEGIN
WITH dob:ob SELECT FROM
BooleanType    =>iob←dob;
ENDCASE        =>JaMTypeErr[];
END;

DescStringType: PUBLIC PROCEDURE [ob:Object] RETURNS [iob:StringType Object] =
BEGIN
WITH dob:ob SELECT FROM
StringType     =>iob←dob;
ENDCASE        =>JaMTypeErr[];
END;

DescStreamType: PUBLIC PROCEDURE [ob:Object] RETURNS [iob:StreamType Object] =
BEGIN
WITH dob:ob SELECT FROM
StreamType     =>iob←dob;
ENDCASE        =>JaMTypeErr[];
END;

DescCommandType: PUBLIC PROCEDURE [ob:Object] RETURNS [iob:CommandType Object] =
BEGIN
WITH dob:ob SELECT FROM
CommandType    =>iob←dob;
ENDCASE        =>JaMTypeErr[];
END;

DescDictType: PUBLIC PROCEDURE [ob:Object] RETURNS [iob:DictType Object] =
BEGIN
WITH dob:ob SELECT FROM
DictType       =>iob←dob;
ENDCASE        =>JaMTypeErr[];
END;

DescArrayType: PUBLIC PROCEDURE [ob:Object] RETURNS [iob:ArrayType Object] =
BEGIN
WITH dob:ob SELECT FROM
ArrayType      =>iob←dob;
ENDCASE        =>JaMTypeErr[];
END;
```

```
DescStackType: PUBLIC PROCEDURE [ob:Object] RETURNS [iob:StackType Object] =
BEGIN
WITH dob:ob SELECT FROM
StackType       =>iob←dob;
ENDCASE      =>JaMTypeErr[];
END;

DescFrameType: PUBLIC PROCEDURE [ob:Object] RETURNS [iob:FrameType Object] =
BEGIN
WITH dob:ob SELECT FROM
FrameType       =>iob←dob;
ENDCASE      =>JaMTypeErr[];
END;

JaMTypeErr: PROCEDURE =
BEGIN
JaMError[TypeChk,TRUE];
END;


TypeChk: StringType Object ← MakeStringObject[".typechk"];

END.
```

```
--file JaMAttributes.mesa
--Written by John Warnock/Martin Newell, February, 1979.
--Edited February 21, 1979  10:17 AM
DIRECTORY

JaMMasterDefs: FROM "JaMMasterDefs",
JaMLiteralDefs: FROM "JaMLiteralDefs",
JaMControlDefs: FROM "JaMControlDefs",
JaMDictionaryDefs: FROM "JaMDictionaryDefs",
JaMTypeChkDefs: FROM "JaMTypeChkDefs",
JaMStackDefs: FROM "JaMStackDefs",
JaMAttributesDefs: FROM "JaMAttributesDefs";

JaMAttributes: PROGRAM
IMPORTS JaMLiteralDefs,JaMControlDefs,JaMDictionaryDefs,JaMStackDefs
EXPORTS JaMAttributesDefs =
BEGIN
OPEN JaMMasterDefs,JaMLiteralDefs,JaMStackDefs;

Length: PUBLIC PROCEDURE =
BEGIN
SLength[JaMControlDefs.GetCurrentFrame[].opstk];
END;

SLength: PUBLIC PROCEDURE [stack: Stack] =
--Does stack: (object) => (length of object)
BEGIN
ob: Object ← Pop[stack];
WITH dob:ob SELECT FROM
StringType  => IntegerLit[dob.Length,stack];
ArrayType   => IntegerLit[dob.Length,stack];
DictType => BEGIN
      dict: DictType Object = dob;
      IntegerLit[JaMDictionaryDefs.Length[dict],stack];
      END;
StackType   => BEGIN
      stk: StackType Object = dob;
      IntegerLit[JaMStackDefs.CountStk[stk.StkPtr],stack];
      END;
ENDCASE  => IntegerLit[1,stack];
END;

LitType: PUBLIC PROCEDURE =
BEGIN
frame:Frame ←JaMControlDefs.GetCurrentFrame[];
ob: Object ← Pop[frame.opstk];
IF ob.litflag = lit THEN Push[LiteralOb,frame.opstk] ELSE
Push[ExecOb,frame.opstk];
END;

Type: PUBLIC PROCEDURE =
BEGIN
frame:Frame ←JaMControlDefs.GetCurrentFrame[];
ob: Object ← Pop[frame.opstk];
WITH dob:ob SELECT FROM
      IntegerType =>    Push[IntegerOb,frame.opstk];
      LongIntegerType =>   Push[LongIntegerOb,frame.opstk];
      RealType =>    Push[RealOb,frame.opstk];
      BooleanType =>    Push[BooleanOb,frame.opstk];
      StringType =>  Push[StringOb,frame.opstk];
      StreamType =>  Push[StreamOb,frame.opstk];
      CommandType =>    Push[CommandOb,frame.opstk];
      DictType =>    Push[DictOb,frame.opstk];
      ArrayType =>   Push[ArrayOb,frame.opstk];
      StackType =>   Push[StackOb,frame.opstk];
      FrameType =>   Push[FrameOb,frame.opstk];
      MarkType  =>   Push[MarkOb,frame.opstk];
      NullType => Push[NullOb,frame.opstk];
      ENDCASE;
END;

ConvertToLiteral: PUBLIC PROCEDURE =
BEGIN
frame:Frame ←JaMControlDefs.GetCurrentFrame[];
ob: Object ← Pop[frame.opstk];
ob.litflag ← lit;
```

```
Push[ob,frame.opstk];
END;

ConvertToExec: PUBLIC PROCEDURE =
BEGIN
frame:Frame ←JaMControlDefs.GetCurrentFrame[];
ob: Object ← Pop[frame.opstk];
ob.litflag ← nolit;
Push[ob,frame.opstk];
END;

IntegerOb:StringType Object←
      JaMLiteralDefs.MakeStringObject[".integertype"L];
LongIntegerOb:StringType Object←
      JaMLiteralDefs.MakeStringObject[".longintegertype"L];
RealOb:StringType Object←
      JaMLiteralDefs.MakeStringObject[".realtype"L];
BooleanOb:StringType Object←
      JaMLiteralDefs.MakeStringObject[".booleantype"L];
StringOb:StringType Object←
      JaMLiteralDefs.MakeStringObject[".stringtype"L];
StreamOb:StringType Object←
      JaMLiteralDefs.MakeStringObject[".streamtype"L];
CommandOb:StringType Object←
      JaMLiteralDefs.MakeStringObject[".commandtype"L];
DictOb:StringType Object←
      JaMLiteralDefs.MakeStringObject[".dicttype"L];
ArrayOb:StringType Object←
      JaMLiteralDefs.MakeStringObject[".arraytype"L];
StackOb:StringType Object←
      JaMLiteralDefs.MakeStringObject[".stacktype"L];
FrameOb:StringType Object←
      JaMLiteralDefs.MakeStringObject[".frametype"L];
MarkOb:StringType Object←
      JaMLiteralDefs.MakeStringObject[".marktype"L];
NullOb:StringType Object←
      JaMLiteralDefs.MakeStringObject[".nulltype"L];
LiteralOb:StringType Object←
      JaMLiteralDefs.MakeStringObject[".literal"L];
ExecOb:StringType Object←
      JaMLiteralDefs.MakeStringObject[".executable"L];


JaMControlDefs.RegisterCommand[".litchk",LitType];
JaMControlDefs.RegisterCommand[".type",Type];
JaMControlDefs.RegisterCommand[".length",Length];
JaMControlDefs.RegisterCommand[".cvlit",ConvertToLiteral];
JaMControlDefs.RegisterCommand[".cvx",ConvertToExec];

END.
```

```
-- JaMVM.mesa --
-- Written by Martin Newell, February 1979
-- Updated February 23, 1979  2:58 PM by MN

-- Virtual memory is paged onto an Alto file system file, using the DMS Core
-- system.  Due to the addressing used in the Core system package, VM
-- addresses are mapped directly onto words in the file i.e. use of VM
-- word n allocates space on the file for VM words [0..n].  A different page
-- transfer package may not make this requirement.
-- VM address 0 maps onto the first word of page 2 (counting from 0) of the
-- file, because page 0 is used by the Alto file system, and page 1 is used
-- by the VM package for restart information.   A simple marching allocator
-- is provided, but it is logically separate from the VM package itself
-- (i.e. the first command can be GetWordVM[...]).

DIRECTORY
   SystemDefs: FROM "SystemDefs",
   MiscDefs: FROM "MiscDefs",
   InlineDefs: FROM "InlineDefs",
   BitBltDefs: FROM "BitBltDefs",
   Mopcodes: FROM "Mopcodes",
   crD: FROM "CoreDefs",
   ovD: FROM "OverviewDefs",
   JaMVMDefs: FROM "JaMVMDefs";

JaMVM: PROGRAM
IMPORTS SystemDefs,MiscDefs,crD,JaMVMDefs
EXPORTS JaMVMDefs =
BEGIN OPEN InlineDefs,JaMVMDefs;

--*** GLOBALS ***

PageSizeb: CARDINAL = 8; --Must be log2(PageSizew)
PageSizew: CARDINAL = 256; --in words - Must be: 2**PageSizeb
PageSizec: CARDINAL = 512; --in chars - Must be: 2*PageSizew
MaxPages: CARDINAL; --# pages available in VM
MaxWords: LONG POINTER; --Must be MaxPages*PageSizew
Words: LONG POINTER; --# words allocated
Offset: CARDINAL; --# bytes allocated in last Word

-- paging buffers
BufferDesc: TYPE = POINTER TO BufferDescRecord;
BufferDescRecord: TYPE = RECORD[
         page: CARDINAL,
         dirty: BOOLEAN,
         buffer: POINTER,
         next: BufferDesc];
swapBuffer: BufferDesc;
firstBuffer: BufferDesc;

-- Page table:
PageTableEntry: TYPE = BufferDesc;
nullPageTableEntry: PageTableEntry = NIL;
PageTable: DESCRIPTOR FOR ARRAY OF PageTableEntry;

-- Paging file
uFH: crD.UFileHandle ← NIL;

--*** PROCEDURES ***

   InitializeVM: PUBLIC PROCEDURE[filename: STRING, nPages,nbuffers: CARDINAL] =
-- Clear and initialize VM to page from file filename, not to
-- exceed nPages, using nbuffers buffers
   BEGIN
      exists: BOOLEAN = RestartVM[filename, nPages, nbuffers];
      IF exists
      THEN  BEGIN
            IF crD.UFileTruncate[0,0,uFH]#ovD.ok THEN ERROR VMFileProblem;
            Words ← LOOPHOLE[LONG[0]];
            Offset ← 0;
            END;
   END;

   RestartVM: PUBLIC PROCEDURE[filename: STRING, nPages,nbuffers: CARDINAL]
                  RETURNS[restarted: BOOLEAN] =
-- Restart VM to page from file filename, not to exceed nPages, using
```

```
-- nbuffers buffers
-- Page buffers are allocated off the Heap, and are pointed to by a ring
-- of descriptors, BufferDesc.  swapBuffer points to the element of
-- the list which will be swapped next.  swapBuffer is simply stepped on every
-- swap (see MemAddress).
   BEGIN
      i: CARDINAL;
      bD: BufferDesc;
      lastPage: CARDINAL;
      erc: ovD.ErrorCode;
      buffer: POINTER;
      [erc,uFH] ← crD.OpenFile[[["JaM"],[""]],[filename],update];
      IF erc#ovD.ok THEN ERROR VMFileProblem;
      swapBuffer ← NIL;
      buffer ← SystemDefs.AllocateResidentPages[nbuffers];
      THROUGH [1..nbuffers]   -- Build list of bD and buffers
      DO bD ← SystemDefs.AllocateHeapNode[SIZE[BufferDescRecord]];
         bD↑ ← [  page: 0,
                  dirty: FALSE,
                  buffer: buffer,
                  next: swapBuffer];
         buffer ← buffer + PageSizew;
         IF swapBuffer=NIL THEN firstBuffer ← bD;
         swapBuffer ← bD;
      ENDLOOP;
      firstBuffer.next ← swapBuffer; --close the ring
-- Initialize page table
      [,lastPage,] ← crD.UFileLength[uFH];
      MaxPages ← MAX[nPages,lastPage];
      MaxWords ← LOOPHOLE[LONG[MaxPages]*LONG[PageSizew]];
      PageTable ← DESCRIPTOR[
         SystemDefs.AllocateResidentPages[
            (MaxPages*SIZE[PageTableEntry]+255)/256],
         MaxPages];
      FOR i IN [0..MaxPages) DO PageTable[i] ← nullPageTableEntry; ENDLOOP;
-- and some other state:
      IF lastPage=0
      THEN  BEGIN --New file
            restarted ← FALSE;
            Words ← LOOPHOLE[LONG[0]];
            Offset ← 0;
            END
      ELSE  BEGIN --Old file
            restarted ← TRUE;
            ReadPage[swapBuffer.buffer,0];   --borrow a buffer
            COPY[swapBuffer.buffer,2,@Words];
            Offset ← (swapBuffer.buffer+2)↑;
            END;
   END;

   FlushVM: PUBLIC PROCEDURE =
-- Write up page buffers and deallocate them
   BEGIN
      bD: BufferDesc;
      nextbD: BufferDesc;
--save state in page "0"
      WriteUpPageBuffer[swapBuffer];   --borrow a buffer
      COPY[@Words,2,swapBuffer.buffer];
      (swapBuffer.buffer+2)↑ ← Offset;
      WritePage[swapBuffer.buffer,0];
--write up buffers and deallocate
      SystemDefs.FreePages[firstBuffer.buffer];
      FOR bD ← swapBuffer, nextbD
      DO WriteUpPageBuffer[bD];
         nextbD ← bD.next;
         SystemDefs.FreeHeapNode[bD];
         IF nextbD=swapBuffer THEN EXIT;
      ENDLOOP;
      IF crD.CloseFile[uFH]#ovD.ok THEN ERROR VMFileProblem;
      uFH ← NIL;
   END;

   AllocateWordsVM: PUBLIC PROCEDURE[length: CARDINAL] RETURNS[location: LONG POINTER] =
-- Increment allocataion pointer by length words
-- AllocateWordsVM[0] aligns on a word boundary
-- Generates VMFull:
```

```
   BEGIN
      IF Offset#0
      THEN  BEGIN
            Words ← Words+1;
            Offset ← 0;
      END;
      location ← Words;
      Words ← Words + length;
      IF Words-MaxWords>=0 THEN ERROR VMFull;
   END;

   AllocateCharsVM: PUBLIC PROCEDURE[length: CARDINAL] RETURNS[location: LONG POINTER, offset: CARDINAL
**] =
-- Increment allocataion pointer by length chars
-- Generates VMFull:
   BEGIN
      frac: CARDINAL ← Offset+BITAND[length,1];
      location ← Words;
      offset ← Offset;
      Words ← Words + BITSHIFT[length,-1]+BITSHIFT[frac,-1];    --mind overflow!
      Offset ← BITAND[frac,1];
      IF Words-MaxWords>=0 THEN ERROR VMFull;
   END;

   GetWordVM: PUBLIC PROCEDURE[VMAddr: LONG POINTER, index: CARDINAL]
      RETURNS[value: WORD] =
-- Get a word from VM address VMAddr+index
-- Generates VMBadPointer:
   BEGIN
      page: CARDINAL;
      addrInPage: CARDINAL;
      [page,addrInPage] ← PageNumber[VMAddr+index];
      RETURN[(MemAddress[page,FALSE] + addrInPage)↑];
   END;

   GetCharVM: PUBLIC PROCEDURE[VMAddr: LONG POINTER, offset: CARDINAL, index: CARDINAL] RETURNS[char: C
**HARACTER] =
-- Get a character from VM address VMAddr
-- Generates VMBadPointer:
   BEGIN --wierd arithmetic is to avoid overflow
      frac: CARDINAL ← offset+BITAND[index,1];
      word: WORD ← GetWordVM[VMAddr,BITSHIFT[index,-1]+BITSHIFT[frac,-1]];
      IF BITAND[frac,1]=0
      THEN char ← LOOPHOLE[BITSHIFT[word,-8]]
      ELSE char ← LOOPHOLE[BITAND[word,377B]];
   END;

   PutWordVM: PUBLIC PROCEDURE[value: WORD, VMAddr: LONG POINTER, index: CARDINAL] =
-- Put value to VM address VMAddr+index
-- Generates VMBadPointer:
   BEGIN
      page: CARDINAL;
      addrInPage: CARDINAL;
      [page,addrInPage] ← PageNumber[VMAddr+index];
      (MemAddress[page,TRUE] + addrInPage)↑ ← value;
   END;

   PutCharVM: PUBLIC PROCEDURE[char: CHARACTER,
                     VMAddr: LONG POINTER, offset: CARDINAL, index: CARDINAL] =
-- Put a character to VM address VMAddr
-- Generates VMBadPointer:
   BEGIN
      frac: CARDINAL ← offset+BITAND[index,1];
      off: CARDINAL ← BITSHIFT[index,-1]+BITSHIFT[frac,-1];
      word: WORD ← GetWordVM[VMAddr,off];
      IF BITAND[frac,1]=0
      THEN word ← BITOR[BITAND[word,377B],BITSHIFT[LOOPHOLE[char],8]]
      ELSE word ← BITOR[BITAND[word,177400B],BITAND[LOOPHOLE[char],377B]];
      PutWordVM[word,VMAddr,off];
   END;

   GetWordsVM: PUBLIC PROCEDURE[VMAddr: LONG POINTER, realAddr: POINTER, nwords: CARDINAL] =
-- Copy nwords words from VM address VMAddr to actual memory address realAddr
-- Generates VMBadPointer:
   BEGIN
      firstpage,addrInFirstPage: CARDINAL;
```

```
      lastpage,addrInLastPage: CARDINAL;
      [firstpage,addrInFirstPage] ← PageNumber[VMAddr];
      [lastpage,addrInLastPage] ← PageNumber[VMAddr+nwords-1];
      IF firstpage=lastpage
      THEN COPY[MemAddress[firstpage,FALSE] + addrInFirstPage,
                                    nwords, realAddr]
      ELSE  BEGIN
            r: POINTER ← realAddr;
            page: CARDINAL;
            -- first page
            COPY[MemAddress[firstpage,FALSE] + addrInFirstPage,
                  PageSizew-addrInFirstPage, realAddr];
            r ← r + PageSizew-addrInFirstPage;
            -- middle pages
            FOR page IN (firstpage..lastpage)
            DO COPY[MemAddress[page,FALSE], PageSizew, r];
               r ← r + PageSizew;
            ENDLOOP;
            -- last page
            COPY[MemAddress[lastpage,FALSE], addrInLastPage+1, r];
      END;
   END;

   GetCharsVM: PUBLIC PROCEDURE[VMAddr: LONG POINTER, VMOffset: CARDINAL,
                                          realAddr: POINTER, realOffset: CARDINAL,
                                          nchars: CARDINAL] =
-- Copy nchars chars from VM address VMAddr to actual memory address realAddr
-- Generates VMBadPointer:
   BEGIN
      firstpage,addrInFirstPage: CARDINAL;
      lastpage,addrInLastPage: CARDINAL;
      frac: CARDINAL = VMOffset+BITAND[nchars,1];
      nw: CARDINAL = BITSHIFT[nchars,-1]+BITSHIFT[frac+1,-1];
      [firstpage,addrInFirstPage] ← PageNumber[VMAddr];
      [lastpage,addrInLastPage] ← PageNumber[VMAddr+nw-1];
      IF firstpage=lastpage
      THEN ByteBlt[MemAddress[firstpage,FALSE] + addrInFirstPage, VMOffset,
                        nchars, realAddr, realOffset]
      ELSE  BEGIN
            r: POINTER ← realAddr;
            roffset: CARDINAL ← realOffset;
            page: CARDINAL;
            firstnchars: CARDINAL =
               BITSHIFT[PageSizew-addrInFirstPage,1]-VMOffset;
            lastnchars: CARDINAL =
               BITSHIFT[addrInLastPage+1,1]-BITAND[frac,1];
            dc: CARDINAL;
            -- first page
            ByteBlt[MemAddress[firstpage,FALSE] + addrInFirstPage, VMOffset,
                     firstnchars, realAddr, realOffset];
            dc ← realOffset + firstnchars;
            r ← r + BITSHIFT[dc,-1];
            roffset ← BITAND[dc,1];
            -- middle pages
            FOR page IN (firstpage..lastpage)
            DO ByteBlt[MemAddress[page,FALSE], 0, PageSizec, r, roffset];
               r ← r + PageSizew;
            ENDLOOP;
            -- last page
            ByteBlt[MemAddress[lastpage,FALSE], 0, lastnchars, r, roffset];
      END;
   END;

   PutWordsVM: PUBLIC PROCEDURE[VMAddr: LONG POINTER, realAddr: POINTER, nwords: CARDINAL] =
-- Copy nwords words from actual memory to VM
-- Generates VMBadPointer:
   BEGIN
      firstpage,addrInFirstPage: CARDINAL;
      lastpage,addrInLastPage: CARDINAL;
      [firstpage,addrInFirstPage] ← PageNumber[VMAddr];
      [lastpage,addrInLastPage] ← PageNumber[VMAddr+nwords-1];
      IF firstpage=lastpage
      THEN COPY[realAddr, nwords,
                  MemAddress[firstpage,TRUE] + addrInFirstPage]
      ELSE  BEGIN
            r: POINTER ← realAddr;
```

```
            page: CARDINAL;
            -- first page
            COPY[realAddr, PageSizew-addrInFirstPage,
                  MemAddress[firstpage,TRUE] + addrInFirstPage];
            r ← r + PageSizew-addrInFirstPage;
            -- middle pages
            FOR page IN (firstpage..lastpage)
            DO COPY[r, PageSizew, MemAddress[page,TRUE]];
               r ← r + PageSizew;
            ENDLOOP;
            -- last page
            COPY[r, addrInLastPage+1, MemAddress[lastpage,TRUE]];
      END;
   END;

   PutCharsVM: PUBLIC PROCEDURE[VMAddr: LONG POINTER, VMOffset: CARDINAL,
                                realAddr: POINTER, realOffset: CARDINAL,
                                nchars: CARDINAL] =
-- Copy nchars chars from actual memory to VM
-- Generates VMBadPointer:
   BEGIN
      firstpage,addrInFirstPage: CARDINAL;
      lastpage,addrInLastPage: CARDINAL;
      frac: CARDINAL = VMOffset+BITAND[nchars,1];
      nw: CARDINAL = BITSHIFT[nchars,-1]+BITSHIFT[frac+1,-1];
      [firstpage,addrInFirstPage] ← PageNumber[VMAddr];
      [lastpage,addrInLastPage] ← PageNumber[VMAddr+nw-1];
      IF firstpage=lastpage
      THEN ByteBlt[realAddr, realOffset, nchars,
                  MemAddress[firstpage,TRUE] + addrInFirstPage, VMOffset]
      ELSE  BEGIN
            r: POINTER ← realAddr;
            roffset: CARDINAL ← realOffset;
            page: CARDINAL;
            firstnchars: CARDINAL =
               BITSHIFT[PageSizew-addrInFirstPage,1]-VMOffset;
            lastnchars: CARDINAL =
               BITSHIFT[addrInLastPage+1,1]-BITAND[frac,1];
            dc: CARDINAL;
            -- first page
            ByteBlt[realAddr, realOffset, firstnchars,
                  MemAddress[firstpage,TRUE] + addrInFirstPage, VMOffset];
            dc ← realOffset + firstnchars;
            r ← r + BITSHIFT[dc,-1];
            roffset ← BITAND[dc,1];
            -- middle pages
            FOR page IN (firstpage..lastpage)
            DO ByteBlt[r, roffset, PageSizec, MemAddress[page,TRUE], 0];
               r ← r + PageSizew;
            ENDLOOP;
            -- last page
            ByteBlt[r, roffset, lastnchars, MemAddress[lastpage,TRUE], 0];
      END;
   END;

-- PRIVATE PROCEDURES --

   MemAddress: PROCEDURE [page: CARDINAL, forWrite: BOOLEAN]
                  RETURNS [POINTER] =
--Get memory address of page, swapping etc. as necessary
   BEGIN
      bD: BufferDesc;
      IF page>=MaxPages THEN ERROR VMBadPointer;
      bD ← PageTable[page];
      IF bD=NIL
      THEN  BEGIN --its not in core
            WriteUpPageBuffer[swapBuffer];
            bD ← swapBuffer;
            ReadPage[bD.buffer,page+1];
            bD.page ← page;
            PageTable[page] ← bD;
            swapBuffer ← swapBuffer.next; --step next swap candidate
            END;
      bD.dirty ← bD.dirty OR forWrite;
      RETURN[bD.buffer];
   END;
```

```
   WriteUpPageBuffer: PROCEDURE[bD: BufferDesc] =
--Write up page buffer bD
   BEGIN
         IF bD.dirty
         THEN  BEGIN
               WritePage[bD.buffer,bD.page+1];
               bD.dirty ← FALSE;
               END;
         IF PageTable[swapBuffer.page]=swapBuffer --test init case
         THEN  PageTable[swapBuffer.page] ← NIL;
   END;

--The following three procedures depend on the secondary storage device

   ReadPage: PROCEDURE [memaddr: POINTER, page: CARDINAL] =
--Read a page to memory address memaddr
   BEGIN
      erc: ovD.ErrorCode;
      bytesRead: CARDINAL;
      [erc,bytesRead] ← crD.ReadPages[memaddr, PageSizec, page, uFH];
      IF erc#ovD.ok THEN ERROR VMFileProblem;
      IF bytesRead=0 THEN MiscDefs.Zero[memaddr,PageSizew];
   END;

   WritePage: PROCEDURE [memaddr: POINTER, page: CARDINAL] =
--Write a page from memory address memaddr
   BEGIN
      IF crD.WritePages[memaddr, PageSizec, page, uFH]#ovD.ok
      THEN ERROR VMFileProblem;
   END;

--

   PageNumber: PROCEDURE [addr: LONG POINTER]
                  RETURNS [pageNum,addrInPage: CARDINAL] =
   -- Returns 16 bit page # and address in page from 24 bits of addr -
   -- specifically for 256 word pages
   BEGIN
      RETURN[BITSHIFT[LowHalf[addr],-8]+BITSHIFT[HighHalf[addr],8],
               BITAND[LowHalf[addr], PageSizew-1]];
   END;

   BBp: BitBltDefs.BBptr ← LOOPHOLE[BITAND[LOOPHOLE[
      SystemDefs.AllocateHeapNode[SIZE[BitBltDefs.BBTable]+1]+1],-2]];

   ByteBlt: PROCEDURE [from: POINTER, fromoffset: [0..1], nbytes: CARDINAL,
                       to: POINTER, tooffset: [0..1]] =
   BEGIN OPEN BitBltDefs;
      BBp↑ ←
      [  pad: 0,
         sourcealt: FALSE,
         destalt: FALSE,                  -- TRUE to use alternate memory bank
         sourcetype: block,
         function: replace,
         unused: 0,
         dbca: to,                        -- destination BaseCoreAddress
         dbmr: 77777B,                    -- destination raster width(in words)
         dlx: BITSHIFT[tooffset,3],       -- destination left x
         dty: 0,                          -- destination top y
         dw: BITSHIFT[nbytes,3],
         dh: 1,
         sbca: from,                      -- source BaseCoreAddress
         sbmr: 77777B,                    -- source raster width(in words)
         slx: BITSHIFT[fromoffset,3],     -- source left x
         sty: 0,                          -- source top y
         gray0: 0,                        -- four words of "gray"
         gray1: 0,
         gray2: 0,
         gray3: 0];
      BITBLT[BBp];
   END;

   HighHalf: PROCEDURE [LONG POINTER] RETURNS [INTEGER] =
      MACHINE CODE BEGIN Mopcodes.zEXCH; Mopcodes.zPOP END;
```

```
   LowHalf: PROCEDURE [LONG POINTER] RETURNS [INTEGER] =
      MACHINE CODE BEGIN Mopcodes.zPOP END;

   VMBadPointer:  PUBLIC ERROR = CODE;
   VMFull:        PUBLIC ERROR = CODE;
   VMFileProblem: PUBLIC ERROR = CODE;

END.
```

```
--File: JaMMath.mesa
--Written by: John Warnock, February 27, 1979
--Last Updated:February 27, 1979  4:03 PM
DIRECTORY

JaMMasterDefs: FROM "JaMMasterDefs",
JaMFnsDefs: FROM "JaMFnsDefs",
JaMStackDefs: FROM "JaMStackDefs",
JaMStringDefs: FROM "JaMStringDefs",
JaMExecDefs: FROM "JaMExecDefs",
StreamDefs: FROM "StreamDefs",
JaMMathDefs: FROM "JaMMathDefs",
JaMControlDefs: FROM "JaMControlDefs",
JaMVMDefs: FROM "JaMVMDefs",
JaMLiteralDefs: FROM "JaMLiteralDefs";

JaMMath: PROGRAM
IMPORTS
JaMFnsDefs,
JaMControlDefs,
JaMLiteralDefs,
JaMStringDefs,
JaMExecDefs,
JaMStackDefs
EXPORTS
JaMMathDefs =

--The following routines implement a set of literal building routines. The resulting constructed litera
**l is pushed on the operand stack.

BEGIN
OPEN JaMControlDefs,JaMMasterDefs,JaMStackDefs,JaMLiteralDefs,JaMExecDefs,StreamDefs;

TypeError: PROCEDURE = BEGIN JaMError[TypeChk,TRUE] END;

MixedMode: PUBLIC PROCEDURE [ob1,ob2:POINTER TO Object,
PInt:PROCEDURE [i,j:INTEGER, o: POINTER TO Object],
PLongInt:PROCEDURE [i,j:LONG INTEGER, o: POINTER TO Object],
PReal:PROCEDURE [i,j:REAL, o: POINTER TO Object]]
RETURNS [rs:Object] =
BEGIN
rsptr: POINTER TO Object ← @rs;
WITH dob1:ob1↑ SELECT FROM
IntegerType    =>WITH dob2:ob2↑ SELECT FROM
      IntegerType =>  PInt[dob1.IntegerVal,dob2.IntegerVal, rsptr];
      LongIntegerType   =>BEGIN
            li:LONG INTEGER ← dob1.IntegerVal;
            PLongInt[li,dob2.LongIntegerVal, rsptr];
            END;
      RealType =>BEGIN
            r:REAL←dob1.IntegerVal;
            PReal[r,dob2.RealVal, rsptr];
            END;
      ENDCASE     => TypeError[];
LongIntegerType   =>WITH dob2:ob2↑ SELECT FROM
      IntegerType =>BEGIN
            li:LONG INTEGER ← dob2.IntegerVal;
            PLongInt[dob1.LongIntegerVal,li, rsptr];
            END;
      LongIntegerType   =>BEGIN
            PLongInt[dob1.LongIntegerVal,dob2.LongIntegerVal, rsptr];
            END;
      RealType =>BEGIN
            r:REAL←dob1.LongIntegerVal;
            PReal[r,dob2.RealVal, rsptr];
            END;
      ENDCASE     => TypeError[];
RealType    =>WITH dob2:ob2↑ SELECT FROM
      IntegerType =>BEGIN
            r:REAL←dob2.IntegerVal;
            PReal[dob1.RealVal,r, rsptr];
            END;
      LongIntegerType   =>BEGIN
            r:REAL←dob2.LongIntegerVal;
            PReal[dob1.RealVal,r, rsptr];
            END;
```

```
      RealType =>BEGIN
            PReal[dob1.RealVal,dob2.RealVal, rsptr];
            END;
      ENDCASE     => TypeError[];
ENDCASE     => TypeError[];
END;

Add: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
ob2:Object ← Pop[frame.opstk];
ob1:Object ← Pop[frame.opstk];
Push [MixedMode[@ob1,@ob2,AddI,AddLI,AddR],frame.opstk];
END;

AddI: PROCEDURE [i,j:INTEGER, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,IntegerType [i+j]];
END;

AddLI: PROCEDURE [i,j: LONG INTEGER, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,LongIntegerType [i+j]];
END;

AddR: PROCEDURE [i,j:REAL, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,RealType [i+j]];
END;


Sub: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
ob2:Object ← Pop[frame.opstk];
ob1:Object ← Pop[frame.opstk];
Push [MixedMode[@ob1,@ob2,SubI,SubLI,SubR],frame.opstk];
END;

SubI: PROCEDURE [i,j:INTEGER, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,IntegerType [i-j]];
END;

SubLI: PROCEDURE [i,j: LONG INTEGER, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,LongIntegerType [i-j]];
END;

SubR: PROCEDURE [i,j:REAL, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,RealType [i-j]];
END;


Mul: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
ob2:Object ← Pop[frame.opstk];
ob1:Object ← Pop[frame.opstk];
Push [MixedMode[@ob1,@ob2,MulI,MulLI,MulR],frame.opstk];
END;

MulI: PROCEDURE [i,j:INTEGER, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,IntegerType [i*j]];
END;

MulLI: PROCEDURE [i,j: LONG INTEGER, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,LongIntegerType [i*j]];
END;

MulR: PROCEDURE [i,j:REAL, o: POINTER TO Object]=
BEGIN
```

```
o↑ ← [lit,RealType [i*j]];
END;


Div: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
ob2:Object ← Pop[frame.opstk];
ob1:Object ← Pop[frame.opstk];
Push [MixedMode[@ob1,@ob2,DivI,DivLI,DivR],frame.opstk];
END;

DivI: PROCEDURE [i,j:INTEGER, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,IntegerType [i/j]];
END;

DivLI: PROCEDURE [i,j: LONG INTEGER, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,LongIntegerType [i/j]];
END;

DivR: PROCEDURE [i,j:REAL, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,RealType [i/j]];
END;

Neg: PUBLIC PROCEDURE =
BEGIN OPEN JaMFnsDefs;
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
ob:Object ← Pop[frame.opstk];
WITH dob:ob SELECT FROM
IntegerType =>BEGIN
      i:INTEGER←dob.IntegerVal;
      PushInteger[-i];
      RETURN;
      END;
LongIntegerType =>BEGIN
      l:LONG INTEGER←dob.LongIntegerVal;
      PushReal[-l];
      RETURN;
      END;
RealType    =>BEGIN
      r:REAL←dob.RealVal;
      PushReal[-r];
      RETURN;
      END;
ENDCASE =>TypeError[];
END;

Equal: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
ob2:Object ← Pop[frame.opstk];
ob1:Object ← Pop[frame.opstk];
WITH sob1:ob1 SELECT FROM
StringType =>BEGIN
      WITH sob2:ob2 SELECT FROM
      StringType =>BEGIN
            IF JaMStringDefs.StringCompare[sob1,sob2] = equal
            THEN JaMFnsDefs.PushBoolean[TRUE]
            ELSE JaMFnsDefs.PushBoolean[FALSE];
            END;
      ENDCASE => TypeError[];
      END;
ENDCASE  => Push [MixedMode[@ob1,@ob2,EqI,EqLI,EqR],frame.opstk];
END;

EqI: PROCEDURE [i,j:INTEGER, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,BooleanType [i = j]];
END;

EqLI: PROCEDURE [i,j: LONG INTEGER, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,BooleanType [i = j]];
```

```
END;

EqR: PROCEDURE [i,j:REAL, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,BooleanType [i = j]];
END;


LessThan: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
ob2:Object ← Pop[frame.opstk];
ob1:Object ← Pop[frame.opstk];
WITH sob1:ob1 SELECT FROM
StringType =>BEGIN
      WITH sob2:ob2 SELECT FROM
      StringType =>BEGIN
            IF JaMStringDefs.StringCompare[sob1,sob2] = less
            THEN JaMFnsDefs.PushBoolean[TRUE]
            ELSE JaMFnsDefs.PushBoolean[FALSE];
            END;
      ENDCASE => TypeError[];
      END;
ENDCASE  =>Push [MixedMode[@ob1,@ob2,LtI,LtLI,LtR],frame.opstk];
END;

LtI: PROCEDURE [i,j:INTEGER, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,BooleanType [i < j]];
END;

LtLI: PROCEDURE [i,j: LONG INTEGER, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,BooleanType [i < j]];
END;

LtR: PROCEDURE [i,j:REAL, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,BooleanType [i < j]];
END;


GreaterThan: PUBLIC PROCEDURE =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
ob2:Object ← Pop[frame.opstk];
ob1:Object ← Pop[frame.opstk];
WITH sob1:ob1 SELECT FROM
StringType =>BEGIN
      WITH sob2:ob2 SELECT FROM
      StringType =>BEGIN
            IF JaMStringDefs.StringCompare[sob1,sob2] = greater
            THEN JaMFnsDefs.PushBoolean[TRUE]
            ELSE JaMFnsDefs.PushBoolean[FALSE];
            END;
      ENDCASE => TypeError[];
      END;
ENDCASE  =>Push [MixedMode[@ob1,@ob2,GtI,GtLI,GtR],frame.opstk];
END;

GtI: PROCEDURE [i,j:INTEGER, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,BooleanType [i > j]];
END;

GtLI: PROCEDURE [i,j: LONG INTEGER, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,BooleanType [i > j]];
END;

GtR: PROCEDURE [i,j:REAL, o: POINTER TO Object]=
BEGIN
o↑ ← [lit,BooleanType [i > j]];
END;
```

```
Not: PUBLIC PROCEDURE =
BEGIN OPEN JaMFnsDefs;
PushBoolean[~ PopBoolean[]];
END;

And: PUBLIC PROCEDURE =
BEGIN OPEN JaMFnsDefs;
b1:BOOLEAN ← PopBoolean[];
b2:BOOLEAN ← PopBoolean[];
PushBoolean[ b2 AND b1];
END;

Or: PUBLIC PROCEDURE =
BEGIN OPEN JaMFnsDefs;
b1:BOOLEAN ← PopBoolean[];
b2:BOOLEAN ← PopBoolean[];
PushBoolean[ b2 OR b1];
END;

Xor: PUBLIC PROCEDURE =
BEGIN OPEN JaMFnsDefs;
b1:BOOLEAN ← PopBoolean[];
b2:BOOLEAN ← PopBoolean[];
PushBoolean[ (b2 AND b1) OR (~b2 AND ~b1)];
END;


TypeChk: StringType Object ← MakeStringObject[".typechk"];


-- register the commands

RegisterCommand[".add",Add];
RegisterCommand[".sub",Sub];
RegisterCommand[".mul",Mul];
RegisterCommand[".div",Div];
RegisterCommand[".neg",Neg];
RegisterCommand[".eq",Equal];
RegisterCommand[".lt",LessThan];
RegisterCommand[".gt",GreaterThan];
RegisterCommand[".not",Not];
RegisterCommand[".and",And];
RegisterCommand[".or",Or];
RegisterCommand[".xor",Xor];

END.
```

```
--file jaminterrupt.mesa
--Written by John Warnock, April 10, 1979
--Last Edit April 10, 1979  8:35 AM by jw
DIRECTORY

ProcessDefs: FROM "ProcessDefs",
FrameDefs: FROM "FrameDefs",
InlineDefs: FROM "InlineDefs",
KeyDefs: FROM "KeyDefs",
JaMInterruptDefs: FROM "JaMInterruptDefs",
JaMExec: FROM "JaMExec",
JaMMasterDefs: FROM "JaMMasterDefs";

JaMInterrupt: MONITOR
IMPORTS JaMExec,ProcessDefs,FrameDefs EXPORTS JaMInterruptDefs =
BEGIN
OPEN JaMMasterDefs;
level:ProcessDefs.InterruptLevel = 11;
Keywakeup:CONDITION;
JaMBreak: ENTRY PROCEDURE =
BEGIN
ProcessDefs.SetPriority[ProcessDefs.DefaultPriority +1];
DO
WAIT Keywakeup;
IF KeyDefs.Keys.Spare2 = down THEN JaMExec.WakeUpFlag ← TRUE;
ENDLOOP;
END;

InitJaMBreak: PUBLIC PROCEDURE =
BEGIN
FrameDefs.MakeCodeResident[FrameDefs.GlobalFrame[JaMBreak]];
ProcessDefs.DisableTimeout[@Keywakeup];
ProcessDefs.CV[level] ← @Keywakeup;
ProcessDefs.DIW↑ ← InlineDefs.BITOR[ProcessDefs.DIW↑,InlineDefs.BITSHIFT[1,level]];
ProcessDefs.Detach[FORK JaMBreak[]];
END;

END.
```

```
--File: JaMArray.mesa
--Written by: John Warnock,March 2, 1979
DIRECTORY

JaMMasterDefs: FROM "JaMMasterDefs",
JaMFnsDefs: FROM "JaMFnsDefs",
JaMStackDefs: FROM "JaMStackDefs",
JaMLiteralDefs: FROM "JaMLiteralDefs",
JaMControlDefs: FROM "JaMControlDefs",
JaMTypeChkDefs: FROM "JaMTypeChkDefs",
JaMExecDefs: FROM "JaMExecDefs",
JaMVMDefs: FROM "JaMVMDefs",
JaMArrayDefs: FROM "JaMArrayDefs";

JaMArray: PROGRAM
IMPORTS
JaMArrayDefs,
JaMStackDefs,
JaMTypeChkDefs,
JaMLiteralDefs,
JaMControlDefs,
JaMExecDefs,
JaMFnsDefs,
JaMVMDefs
EXPORTS
JaMArrayDefs =

--The following routines implement a set of literal building routines. The resulting constructed litera
**l is pushed on the operand stack.

BEGIN
OPEN JaMArrayDefs,JaMMasterDefs,JaMFnsDefs,JaMVMDefs;

Array: PUBLIC PROCEDURE =
BEGIN
null:Object ← [lit,NullType[]];
i:INTEGER ← PopInteger[];
j:INTEGER ← 0;
array:ArrayType Object ← [lit,ArrayType[i,AllocateWordsVM[i*SIZE[Object]]]];
FOR j IN [0..i-1]
DO
PutWordsVM[array.ArrayPtr + SIZE[Object]*j,@null,SIZE[Object]];
ENDLOOP;
PushObject[array];
END;

ArrayPut: PUBLIC PROCEDURE =
BEGIN
val:Object ← PopObject[];
i:INTEGER ← PopInteger[];
array:ArrayType Object ← JaMTypeChkDefs.DescArrayType[PopObject[]];
PutWordsVM[array.ArrayPtr + SIZE[Object]*i,@val,SIZE[Object]];
END;

ArrayGet: PUBLIC PROCEDURE =
BEGIN
val:Object;
i:INTEGER ← PopInteger[];
array:ArrayType Object ← JaMTypeChkDefs.DescArrayType[PopObject[]];
GetWordsVM[array.ArrayPtr + SIZE[Object]*i,@val,SIZE[Object]];
PushObject[val];
END;


ArrayPart: PUBLIC PROCEDURE =
BEGIN
i:INTEGER ← PopInteger[];
j:INTEGER ← PopInteger[];
array:ArrayType Object ← JaMTypeChkDefs.DescArrayType[PopObject[]];
IF i < 0 OR j < 0 THEN RangeErr[];
IF ABS[i+j] >= array.Length THEN RangeErr[];
array.ArrayPtr ← array.ArrayPtr + j*SIZE[Object];
array.Length← i;
PushObject[array];
END;
```

```
ArrayAtom: PUBLIC PROCEDURE [aob:ArrayType Object,stk:JaMMasterDefs.Stack]
   RETURNS [BOOLEAN] =
BEGIN
ob:Object;
IF aob.Length <= 0  THEN RETURN [FALSE];
GetWordsVM[aob.ArrayPtr,@ob,SIZE[Object]];
aob.ArrayPtr ← aob.ArrayPtr + SIZE[Object];
aob.Length← aob.Length - 1;
JaMStackDefs.Push[aob,stk];
JaMStackDefs.Push[ob,stk];
RETURN [TRUE];
END;

ArrayStore: PUBLIC PROCEDURE =
-- expects opstk: (ob0, ob1, ... , obn-1, array)  array of length n
-- returns opstk: (array)
BEGIN
   array: ArrayType Object ← JaMTypeChkDefs.DescArrayType[PopObject[]];
   frame: Frame ← JaMControlDefs.GetCurrentFrame[];
   val: Object;
   i: CARDINAL ← array.Length*SIZE[Object];
   IF array.Length > JaMStackDefs.CountStk[frame.opstk]
   THEN JaMExecDefs.JaMError[StkUndFlw,TRUE];
   THROUGH [0..array.Length)
   DO i ← i - SIZE[Object];
      val ← PopObject[];
      PutWordsVM[array.ArrayPtr + i, @val,SIZE[Object]];
   ENDLOOP;
   JaMStackDefs.Push[array, frame.opstk];
END;

ArrayLoad: PUBLIC PROCEDURE =
-- expects opstk: (array)  array of length n
-- returns opstk: (array[0], array[1], ... , array[n-1], array)
BEGIN
   array: ArrayType Object ← JaMTypeChkDefs.DescArrayType[PopObject[]];
   frame: Frame ← JaMControlDefs.GetCurrentFrame[];
   val: Object;
   i: CARDINAL;
   FOR i ← 0, i+SIZE[Object] UNTIL i = array.Length*SIZE[Object]
   DO GetWordsVM[array.ArrayPtr + i, @val,SIZE[Object]];
      PushObject[val];
   ENDLOOP;
   JaMStackDefs.Push[array, frame.opstk];
END;


--"ArrayForall" is the "do for each array element" instruction. Two operands are required:
-- an Object. The Object is executed for each member of the
-- given array.

ArrayForall: PUBLIC PROCEDURE =
BEGIN OPEN JaMStackDefs;
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
ob:Object ← Pop[frame.opstk];
array:ArrayType Object ← JaMTypeChkDefs.DescArrayType[Pop[frame.opstk]];
m:MarkType Object←[nolit,MarkType[]];
Push[m,frame.execstk];
Push[ob,frame.execstk];
Push[array,frame.execstk];
Push[arrayc,frame.execstk];
END;


CArrayForall: PUBLIC PROCEDURE =
BEGIN
OPEN JaMStackDefs;
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
array:ArrayType Object ← JaMTypeChkDefs.DescArrayType[Pop[frame.execstk]];
ob:Object ← Pop[frame.execstk];
m:Object←Pop[frame.execstk];
IF ~ArrayAtom[array,frame.opstk] THEN RETURN ELSE
BEGIN
Push[m,frame.execstk];
Push[ob,frame.execstk];
```

```
Exch[frame.opstk];
MoveStkOps[frame.opstk,frame.execstk,1];
Push[arrayc,frame.execstk];
Push[ob,frame.execstk];
END;
END;

arrayc:CommandType Object ← [nolit,CommandType[CArrayForall]];

RangeErr:PROCEDURE =
BEGIN
JaMExecDefs.JaMError[rangechk,TRUE];
END;

StkUndFlw: StringType Object ← JaMLiteralDefs.MakeStringObject[".stkundflw"];
rangechk:StringType Object ← JaMLiteralDefs.MakeStringObject[".rangechk"];

JaMControlDefs.RegisterCommand[".array",Array];
JaMControlDefs.RegisterCommand[".aget",ArrayGet];
JaMControlDefs.RegisterCommand[".aput",ArrayPut];
JaMControlDefs.RegisterCommand[".subarray",ArrayPart];
JaMControlDefs.RegisterCommand[".arrayforall",ArrayForall];
JaMControlDefs.RegisterCommand[".astore",ArrayStore];
JaMControlDefs.RegisterCommand[".aload",ArrayLoad];


END.
```

```
-- JaM.config
-- Written by Martin Newell, February 1979
-- Last edited by MN on April 11, 1979  2:12 PM
DIRECTORY

  ovD: FROM "OverViewDefs",
  crD: FROM "CoreDefs",
  gsD: FROM "GlobalStorageDefs";

JaM: CONFIGURATION
   IMPORTS SystemDefs,StreamDefs,ImageDefs,IODefs,FloatDefs,
   StringDefs,MiscDefs,ovD,crD,gsD,FrameDefs,DiskKDDefs,
   DiskDefs,SegmentDefs,ProcessDefs,DirectoryDefs,BFSDefs,
   RectangleDefs,WindowDefs,MenuDefs,EdStreamDefs,DisplayDefs
   EXPORTS JaMFnsDefs
   CONTROL JaMStart =
   BEGIN
    Float; FloatIO; FloatDebug; EdStream;
    CoreSS; CoreIO; CoreCom; GlobalStorage;
    JaMFns; JaMVM; JaMDictionary; JaMString; JaMExec; JaMLiteral; JaMStack;
    JaMAttributes; JaMTypeChk; JaMControl; JaMScanner; JaMIO; JaMMath;
    JaMArray; JaMInterrupt; Display; JaMWindEx; WindowPackage; JaMStart;
   END.
```

```
--File EdStreamDefs.mesa
--Written by Martin Newell, April, 1979.
--Updated April 9, 1979  2:27 PM by MN

DIRECTORY

StreamDefs: FROM "StreamDefs" USING[OtherStreamHandle, StreamHandle],
RectangleDefs: FROM "RectangleDefs" USING[BMHandle, xCoord, yCoord],
WindowDefs: FROM "WindowDefs" USING[WindowHandle];

DEFINITIONS FROM StreamDefs,RectangleDefs,WindowDefs;

EdStreamDefs: DEFINITIONS =
BEGIN

  EditedHandle: TYPE = OtherStreamHandle;

        CreateEditedStream: PUBLIC PROCEDURE [window: WindowHandle,
                                                        eolchar: CHARACTER,
                                                        name: STRING, bitmap: BMHandle,
                                                        x0,width: xCoord, y0,height: yCoord]
                                        RETURNS [stream: EditedHandle];
-- Creates a single edited stream for input and output.
-- Use of the stream for input yields characters from current
--              selections that have been 'doit'ed by typing eolchar.
--      When used for output the stream behaves as a vanilla display stream.
-- If window#NIL and is a scriptfile window then it is used and
--              other parameters are ignored, otherwise a new window is created
--              using remaining parameters.

        CreateBufferedStream: PUBLIC PROCEDURE [stream: StreamHandle]
                                        RETURNS [bstream: StreamHandle];
-- Create a buffered version of the input stream, for doing
--              putback to any depth

END.
```

```
--File EdStream.mesa
--Written by Martin Newell, April, 1979.
--Last Updated: April 9, 1979  2:27 PM by MN

DIRECTORY

JaMWindExDefs: FROM "JaMWindExDefs" USING[ReadEditedString],
RectangleDefs: FROM "RectangleDefs" USING[Rptr, CreateRectangle,
        DestroyRectangle, BMHandle, xCoord, yCoord],
WindowDefs: FROM "WindowDefs" USING[WindowHandle, CreateDisplayWindow,
        DestroyDisplayWindow, SetCurrentDisplayWindow, GetCurrentDisplayWindow],
IODefs: FROM "IODefs" USING[ControlC, CR, GetInputStream,
        GetOutputStream, SetInputStream, SetOutputStream, LineOverflow],
StringDefs: FROM "StringDefs" USING[AppendChar, AppendString,
        StringBoundsFault],
SystemDefs: FROM "SystemDefs" USING[AllocateHeapNode, FreeHeapNode,
        AllocateHeapString, FreeHeapString],
StreamDefs: FROM "StreamDefs" USING[StreamHandle, StreamObject,
        StreamError, CreateDisplayStream, CreateKeyStream],
EdStreamDefs: FROM "EdStreamDefs" USING[EditedHandle];

DEFINITIONS FROM JaMWindExDefs,RectangleDefs,WindowDefs,IODefs,StringDefs,
        SystemDefs,StreamDefs,EdStreamDefs;

EdStream: PROGRAM
IMPORTS JaMWindExDefs,RectangleDefs,WindowDefs,IODefs,StringDefs,
        SystemDefs,StreamDefs
EXPORTS EdStreamDefs
SHARES StreamDefs =
BEGIN

        edData: TYPE = POINTER TO edDataRecord;
        edDataRecord: TYPE = RECORD[
                window: WindowHandle,
                eolchar: CHARACTER,
                line: STRING,
                chptr: CARDINAL];
        defaultLineLength: CARDINAL = 64;

  eofchar: CHARACTER = ControlC;

        CreateEditedStream: PUBLIC PROCEDURE [window: WindowHandle,
                                                eolchar: CHARACTER,
                                                name: STRING, bitmap: BMHandle,
                                                x0,width: xCoord, y0,height: yCoord]
                                        RETURNS [stream: EditedHandle] =
-- Creates a single edited stream for input and output.
-- Use of the stream for input yields characters from current
--              selections that have been 'doit'ed by typing eolchar.
--      When used for output the stream behaves as a vanilla display stream.
-- If window#NIL and is a scriptfile window then it is used and
--              other parameters are ignored, otherwise a new window is created
--              using remaining parameters.
        BEGIN
                eD: edData ← AllocateHeapNode[SIZE[edDataRecord]];
                IF window=NIL OR window.type#scriptfile
                THEN    BEGIN
                                rectangle: Rptr ← CreateRectangle[bitmap,x0,width,y0,height];
                                window ← CreateDisplayWindow[scriptfile,
                                rectangle, CreateDisplayStream[rectangle],
                                CreateKeyStream[],name];
                                END;
                eD↑ ← [
                        window: window,
                        eolchar:        eolchar,
                        line:           AllocateHeapString[defaultLineLength],
                        chptr:  1];
                stream ← AllocateHeapNode[SIZE[StreamObject]];
                stream↑ ← [
                        reset: EdReset,
                        get: EdGet,
                        putback: EdPutBack,
                        put: EdPut,
                        endof: EdEndOf,
                        destroy: EdDestroy,
                        body: Other[data: eD]];
```

```
        END;

        EdReset: PROCEDURE [stream: StreamHandle] =
        BEGIN
        WITH s:stream SELECT FROM
        Other =>
                BEGIN
                        eD: edData ← s.data;
                        eD.line.length ← 0;
                        eD.window.ks.reset[eD.window.ks];
                END;
        ENDCASE;
        END;

        EdGet: PROCEDURE [stream: StreamHandle]
                                RETURNS [ch: CHARACTER] =
        BEGIN
        lastch: CHARACTER;
        WITH s:stream SELECT FROM
        Other =>
                BEGIN
                        eD: edData ← s.data;
                        EndLine: PROCEDURE [char: CHARACTER] RETURNS [BOOLEAN] =
                                BEGIN RETURN[char=eD.eolchar OR char=eofchar]; END;
                        SELECT TRUE FROM
                        eD.chptr<eD.line.length => ch ← eD.line[eD.chptr];
                        eD.chptr=eD.line.length => ch ← eD.eolchar;
                        ENDCASE =>
                                BEGIN
                                        si: StreamHandle ← GetInputStream[];
                                        so: StreamHandle ← GetOutputStream[];
                                        IF GetCurrentDisplayWindow[]#eD.window
                                        THEN SetCurrentDisplayWindow[eD.window];
                                        lastch ← ReadEditedString[eD.line,EndLine,TRUE !
                                                LineOverflow =>
                                                        BEGIN
                                                        nl: STRING ←
                                                                AllocateHeapString[2*eD.line.maxlength]
**;
                                                        AppendString[nl,eD.line];
                                                        FreeHeapString[eD.line];
                                                        eD.line ← nl;
                                                        RESUME[nl];
                                                        END];
                                        IF lastch=eofchar
                                        THEN    AppendChar[eD.line,lastch !
                                                        StringBoundsFault =>
                                                                BEGIN
                                                                nl: STRING ←
                                                                        AllocateHeapString[eD.l
**ine.maxlength + 1];
                                                                AppendString[nl,eD.line];
                                                                FreeHeapString[eD.line];
                                                                eD.line ← nl;
                                                                RESUME[nl];
                                                                END];
                                        eD.window.ds.put[eD.window.ds,CR];  --dubious
                                        SetInputStream[si];
                                        SetOutputStream[so];
                                        eD.chptr ← 0;
                                        ch ← IF eD.line.length=0 THEN eD.eolchar ELSE eD.line[0];
                                END;
                        IF ch=eofchar
                        THEN ERROR StreamError[stream,StreamAccess];
                        eD.chptr ← eD.chptr + 1;
                END;
        ENDCASE;
        END;

        EdPutBack: PROCEDURE [stream: StreamHandle, ch: CHARACTER] =
        BEGIN
        WITH s:stream SELECT FROM
        Other =>
                BEGIN
                        eD: edData ← s.data;
                        IF eD.chptr=0
```

```
                        THEN    BEGIN --not quite right? - will generate eolchar after it...
                                        eD.line[0] ← ch;
                                        eD.line.length ← 1;
                                        END
                        ELSE    BEGIN
                                        eD.chptr ← eD.chptr-1;
                                        eD.line[eD.chptr] ← ch;
                                        END;
                END;
        ENDCASE;
        END;

        EdPut: PROCEDURE [stream: StreamHandle, ch: CHARACTER] =
        BEGIN
        WITH s:stream SELECT FROM
        Other =>
                BEGIN
                        eD: edData ← s.data;
                        eD.window.ds.put[eD.window.ds,ch];
                END;
        ENDCASE;
        END;

        EdEndOf: PROCEDURE [stream: StreamHandle]
                                        RETURNS [BOOLEAN] =
        BEGIN
        WITH s:stream SELECT FROM
        Other =>
                BEGIN
                        eD: edData ← s.data;
                        SELECT TRUE FROM
                        eD.chptr<eD.line.length => RETURN[eD.line[eD.chptr]=eofchar];
                        eD.chptr=eD.line.length => RETURN[FALSE];       --eolchar next
                        eD.chptr>eD.line.length =>
                        BEGIN
                                        BEGIN   --try to read one
                                        end: BOOLEAN ← FALSE;
                                        EdPutBack[stream,EdGet[stream
                                                        ! StreamError =>
                                                                BEGIN
                                                                end ← TRUE;
                                                                CONTINUE;
                                                                END]];
                                        RETURN[end];
                                        END;
                        END;
                        ENDCASE;
                END;
        ENDCASE;
        END;

        EdDestroy: PROCEDURE [stream: StreamHandle] =
        BEGIN
        WITH s:stream SELECT FROM
        Other =>
                BEGIN
                        eD: edData ← s.data;
                        eD.window.ks.destroy[eD.window.ks];
                        eD.window.ds.destroy[eD.window.ds];
                        DestroyRectangle[eD.window.rectangle];
                        DestroyDisplayWindow[eD.window];
                        FreeHeapString[eD.line];
                        FreeHeapNode[eD];
                        FreeHeapNode[stream];
                END;
        ENDCASE;
        END;

-- ***Buffered Streams***

        bData: TYPE = POINTER TO bDataRecord;
        bDataRecord: TYPE = RECORD[
                stream: StreamHandle,
                buffer: STRING];
        defaultBufferLength: CARDINAL = 2;
```

```
        CreateBufferedStream: PUBLIC PROCEDURE [stream: StreamHandle]
                                        RETURNS [bstream: StreamHandle] =
-- Create a buffered version of the input stream, for doing
--              putback to any depth
        BEGIN
                bD: bData ← AllocateHeapNode[SIZE[bDataRecord]];
                bD↑ ← [
                        stream: stream,
                        buffer: AllocateHeapString[defaultLineLength]];
                bstream ← AllocateHeapNode[SIZE[StreamObject]];
                bstream↑ ← [
                        reset: BReset,
                        get: BGet,
                        putback: BPutBack,
                        put: BPut,
                        endof: BEndOf,
                        destroy: BDestroy,
                        body: Other[data: bD]];
        END;

        BReset: PROCEDURE [stream: StreamHandle] =
        BEGIN
        WITH s:stream SELECT FROM
        Other =>
                BEGIN
                        bD: bData ← s.data;
                        bD.buffer.length ← 0;
                        bD.stream.reset[bD.stream];
                END;
        ENDCASE;
        END;

        BGet: PROCEDURE [stream: StreamHandle]
                                RETURNS [ch: CHARACTER] =
        BEGIN
        WITH s:stream SELECT FROM
        Other =>
                BEGIN
                        bD: bData ← s.data;
                        IF bD.buffer.length = 0
                        THEN    RETURN[bD.stream.get[bD.stream]]
                        ELSE    BEGIN
                                        bD.buffer.length ← bD.buffer.length - 1;
                                        RETURN[bD.buffer[bD.buffer.length]];
                                        END;
                END;
        ENDCASE;
        END;

        BPutBack: PROCEDURE [stream: StreamHandle, ch: CHARACTER] =
        BEGIN
        WITH s:stream SELECT FROM
        Other =>
                BEGIN
                        bD: bData ← s.data;
                        AppendChar[bD.buffer,ch ! StringBoundsFault =>
                                                BEGIN
                                                nl: STRING ←
                                                        AllocateHeapString[2*bD.buffer.maxlengt
**h];
                                                AppendString[nl,bD.buffer];
                                                FreeHeapString[bD.buffer];
                                                bD.buffer ← nl;
                                                RESUME[nl];
                                                END];
                END;
        ENDCASE;
        END;

        BPut: PROCEDURE [stream: StreamHandle, ch: CHARACTER] =
        BEGIN
        WITH s:stream SELECT FROM
        Other =>
                BEGIN
                        bD: bData ← s.data;
```

```
                        bD.stream.put[bD.stream,ch];
                END;
        ENDCASE;
        END;

        BEndOf: PROCEDURE [stream: StreamHandle]
                                        RETURNS [BOOLEAN] =
        BEGIN
        WITH s:stream SELECT FROM
        Other =>
                BEGIN
                        bD: bData ← s.data;
                        RETURN[bD.buffer.length=0 AND bD.stream.endof[bD.stream]];
                END;
        ENDCASE;
        END;

        BDestroy: PROCEDURE [stream: StreamHandle] =
        BEGIN
        WITH s:stream SELECT FROM
        Other =>
                BEGIN
                        bD: bData ← s.data;
                        bD.stream.destroy[bD.stream];
                        FreeHeapString[bD.buffer];
                        FreeHeapNode[bD];
                        FreeHeapNode[stream];
                END;
        ENDCASE;
        END;

END.
```

```
--File:  JaMWindExDefs.mesa

DIRECTORY
  InlineDefs: FROM "inlinedefs" USING [LongDivMod, LongMult],
  KeyDefs: FROM "keydefs" USING [MouseButton],
  MenuDefs: FROM "menudefs" USING [MenuItem],
  Mopcodes: FROM "mopcodes" USING [zPOP],
  RectangleDefs: FROM "rectangledefs" USING [DCBptr, FAptr, leftmargin],
  WindowDefs: FROM "windowdefs" USING [
    BMHandle, DiskHandle, DisplayHandle, NullIndex, OriginIndex, Rptr,
    StreamHandle, StreamIndex, WindowHandle, xCoord, yCoord];

DEFINITIONS FROM MenuDefs, RectangleDefs, WindowDefs;

JaMWindExDefs:  DEFINITIONS =
BEGIN

-- some TYPE'S and POINTERS

  JaMWEDataHandle:  TYPE = POINTER TO JaMWEDataObject;
  AMouseButton: TYPE = KeyDefs.MouseButton;
  KeySet: TYPE = [0..37B];
  ButtonProc: PUBLIC TYPE = PROCEDURE[WindowHandle, xCoord, yCoord];
  ProcArray:  PUBLIC TYPE = ARRAY AMouseButton OF ButtonProc;
  CursorType: TYPE = {textpointer, arrow, bullseye, leftbutton,
    uparrow, downarrow, botharrow, hourglass, norm, menu};
  CursorArray: TYPE = ARRAY [0..15] OF CARDINAL;

  -- programs implementing the Window Executive

  JaMWEPosition: PROGRAM [JaMWEState:  JaMWEDataHandle];
  JaMWESelection: PROGRAM [JaMWEState:  JaMWEDataHandle];
  JaMWEWindows: PROGRAM [JaMWEState:  JaMWEDataHandle];
  JaMWEMain: PROGRAM [JaMWEState:  JaMWEDataHandle];
  JaMWEBreak: PROGRAM [JaMWEState: JaMWEDataHandle];
  JaMWEControl: PROGRAM;  -- Control Module

  -- procedures implementing the Window Executive

  ReadEditedString: PROCEDURE[s: STRING,
                        eol: PROCEDURE[CHARACTER]RETURNS[BOOLEAN],
                        unused: BOOLEAN]
                  RETURNS[CHARACTER];
  GetMouseButton:  PROCEDURE RETURNS[AMouseButton];
  GetKeySet:  PROCEDURE RETURNS[KeySet];
  InvertCursor: PUBLIC PROCEDURE RETURNS [BOOLEAN];
  CursorToRectangleCoords:  PROCEDURE [Rptr, xCoord, yCoord]
    RETURNS[xCoord, yCoord];
  SetCursor:  PROCEDURE [CursorType];
  SetJumpStripe:  PROCEDURE [WindowHandle, BOOLEAN];
  NullProc:  PROCEDURE [WindowHandle, xCoord, yCoord];
  NoteNameError:  PROCEDURE [WindowHandle, STRING];
  WriteMessageString:  PROCEDURE [WindowHandle, STRING];
  AssignScratchFile:  PROCEDURE RETURNS[STRING, INTEGER];
  LoadThisWindow: PROCEDURE [w: WindowHandle];
  LoadWindow: PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord];
  GrowWindow: PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord];
  MoveWindow: PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord] ;
  DestroyWindow: PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord];
  CreateWindow: PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord] ;
  PutSelect: PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord];
  WordSelect: PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord];
  TextSelect: PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord];
  MenuSelect: PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord];
  PositionFile: PROCEDURE[w: WindowHandle, x: xCoord, y: yCoord];
  ScrollUpFile: PROCEDURE[w: WindowHandle, x: xCoord, y: yCoord];
  ScrollDownFile: PROCEDURE[w: WindowHandle, x: xCoord, y: yCoord];
  NormalizeSelection: PROCEDURE[w: WindowHandle, x: xCoord, y: yCoord];
  ReadEditChar: PROCEDURE [w: WindowHandle, InString: STRING,
                        eol: PROCEDURE[CHARACTER]RETURNS[BOOLEAN]]
                  RETURNS[STRING];
  LDivMod: PROCEDURE [a: LONG INTEGER, b: CARDINAL] RETURNS [q, r: CARDINAL] =
    LOOPHOLE[InlineDefs.LongDivMod];
  LMult: PROCEDURE [a, b: CARDINAL] RETURNS [LONG INTEGER] =
    LOOPHOLE[InlineDefs.LongMult];
  Shorten: PROCEDURE [a: LONG INTEGER] RETURNS [CARDINAL] =
```

```
    MACHINE CODE BEGIN Mopcodes.zPOP END;

-- constants
  maxlines:  CARDINAL = 80;
  JumpStrip: CARDINAL = leftmargin;
  slop: CARDINAL = 10;
  maxscratch: CARDINAL = 4;
  OriginIndex: StreamIndex = WindowDefs.OriginIndex;
  NullIndex: StreamIndex = WindowDefs.NullIndex;

-- magic memory locations and contents
  DCBchainHead: DCBptr = LOOPHOLE[420B];
  xmouseloc: POINTER = LOOPHOLE[424B];
  ymouseloc: POINTER = LOOPHOLE[425B];
  xcursorloc: POINTER = LOOPHOLE[426B];
  ycursorloc: POINTER = LOOPHOLE[427B];
  buttonsloc: POINTER = LOOPHOLE[177030B];
  cursormap: POINTER = LOOPHOLE[431B];

--WindEx RECORD

JaMWEDataObject:  TYPE = RECORD [
  scratchfiles: ARRAY [0..maxscratch) OF DiskHandle,
  windows: ARRAY [0..4) OF WindowHandle,
  menuarray:  DESCRIPTOR FOR ARRAY OF MenuItem,  -- Filled in by JaMWEWindows
  defaultfont: FAptr,
  defaultlineheight: CARDINAL,
  defaultks: StreamHandle,
  defaultds: DisplayHandle,
  defaultmapdata: BMHandle,
  useKeyset: BOOLEAN,
  currentcursor:  CursorType,
  cxa, cya: [0..256),
  TextProcArray:  ProcArray,
  ScrollProcArray:  ProcArray,
  ButtonProcArray:  ProcArray];


END. of wmanagerdefs
```

```
--File: JaMWEBreak.mesa

DIRECTORY
  MenuDefs: FROM "menudefs" USING [
    ClearMenu, CreateMenu, DisplayMenu, MarkMenuItem, MenuItem],
  RectangleDefs: FROM "rectangledefs" USING [
    CursorToMapCoords],
  JaMWindExDefs: FROM "jamwindexdefs" USING [
    CursorToRectangleCoords,GetMouseButton, SetCursor,
    JaMWEDataHandle, xcursorloc, ycursorloc, NullProc],
  WindowDefs: FROM "windowdefs" USING [WindowHandle, xCoord, yCoord];

DEFINITIONS FROM WindowDefs, JaMWindExDefs;

JaMWEBreak: PROGRAM [JaMWEState: JaMWEDataHandle]
  IMPORTS MenuDefs, RectangleDefs, JaMWindExDefs
  EXPORTS JaMWindExDefs
  SHARES JaMWindExDefs =

BEGIN

OPEN JaMWEState;

CR: CHARACTER = 15C;

-- some externals

nCommands: CARDINAL = 12;

create: CARDINAL = 0;
destroy: CARDINAL = 1;
move: CARDINAL = 2;
grow: CARDINAL = 3;
load: CARDINAL = 4;
stuff: CARDINAL = 5;
find: CARDINAL = 6;
break: CARDINAL = 7;
clear: CARDINAL = 8;
trace: CARDINAL = 9;
position: CARDINAL = 10;
keys: CARDINAL = 11;

MenuSelect: PUBLIC PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord]=
  BEGIN OPEN MenuDefs;
  -- define locals
  index: INTEGER ← -1;
  mapx: xCoord;
  mapy: yCoord;
  defaultmenu:  DESCRIPTOR FOR ARRAY OF MenuItem =
    DESCRIPTOR[BASE[menuarray], LENGTH[menuarray]];
  -- check if a menu
  IF w.menu = NIL THEN w.menu ← CreateMenu[defaultmenu];
  -- paste it up there
  [mapx, mapy] ← RectangleDefs.CursorToMapCoords[defaultmapdata, x, y];
  mapy ← MIN[mapy, MAX[0,(w.rectangle.bitmap.height)
     -(LENGTH[w.menu.array]+LOOPHOLE[defaultlineheight,INTEGER]+2)]];
  SetCursor[menu];
  DisplayMenu[w.menu, w.rectangle.bitmap, mapx, mapy];
  -- while the button is down select menu items
  WHILE GetMouseButton[] = Blue DO
    -- convert to rectangle coords
    x ← xcursorloc↑;
    y ← ycursorloc↑;
    -- and see if in menu
    [x, y] ← CursorToRectangleCoords[w.menu.rectangle, x, y];
    index ← IF x > 0 AND x <= w.menu.rectangle.cw AND y > 0 AND
      y <= w.menu.rectangle.ch THEN y/defaultlineheight ELSE -1;
    IF index >= LENGTH[w.menu.array] THEN index ← -1;
    MarkMenuItem[w.menu, index];
  ENDLOOP;
  -- and restore menus region and contents underneath
  ClearMenu[w.menu];
  -- see if command selected
  IF index # -1 THEN w.menu.array[index].proc[w, xcursorloc↑, ycursorloc↑];
  END;
```

```
ButtonWait: PROCEDURE =
  BEGIN
  -- wait until all button are up
  UNTIL GetMouseButton[] = None DO NULL; ENDLOOP;
  RETURN;
  END;

-- initialization for windows module

InitBreak: PROCEDURE =
  BEGIN OPEN MenuDefs;
  menuarray[break] ← MenuItem["Set Brk", NullProc];
  menuarray[clear] ← MenuItem["Clr Brk", NullProc];
  menuarray[trace] ← MenuItem["Set Trc", NullProc];
  END;


InitBreak[];

END. of JaMWEBreak
```

```
--File:  JaMWEControl.mesa

DIRECTORY
  IODefs: FROM "IODefs" USING [LineOverflow, ControlA, ControlH,
    ControlW, ControlQ, SP, DEL, TAB, CR, LF],
  AltoFileDefs: FROM "AltoFileDefs" USING [FA],
  BitBltDefs: FROM "BitBltDefs" USING [BBptr, BITBLT, BBTable],
  InlineDefs: FROM "InlineDefs" USING [BITAND, BITSHIFT],
  --ProcessDefs: FROM "processdefs" USING [Detach],--
  RectangleDefs: FROM "rectangledefs" USING [
    ComputeCharWidth, GetDefaultBitmap, GetDefaultFont, leftmargin, Rptr,
    xCoord, yCoord, InvertBoxInRectangle],
  StreamDefs: FROM "streamdefs" USING [
    ClearDisplayChar, DisplayHandle, GrEqualIndex, EqualIndex,
    GetDefaultDisplayStream,
    GetDefaultKey, GetIndex, KeyboardHandle, ModifyIndex, SetIndex,
    StreamIndex, --ReadBlock, WriteBlock,-- StreamHandle,
    GetFA, JumpToFA],
  StringDefs: FROM "stringdefs" USING [AppendChar, AppendString],
  SystemDefs: FROM "systemdefs" USING [AllocateHeapString, FreeHeapString,
    AllocateHeapNode],
  JaMWindExDefs: FROM "jamwindexdefs" USING [
    maxscratch, OriginIndex, JaMWEBreak, JaMWEDataObject, JaMWEMain,
    JaMWEPosition, JaMWESelection, JaMWEWindows],
  WindowDefs: FROM "windowdefs" USING [
    AlterWindowType, GetCurrentDisplayWindow, MakeSelection,
    Selection, WindowHandle, PaintDisplayWindow];

DEFINITIONS FROM IODefs, BitBltDefs, InlineDefs, StreamDefs, StringDefs,
       SystemDefs, RectangleDefs, WindowDefs, JaMWindExDefs;

JaMWEControl:  PROGRAM
  IMPORTS IODefs, SystemDefs, StringDefs, StreamDefs,
    RectangleDefs, WindowDefs, JaMWindExDefs
  EXPORTS JaMWindExDefs
  SHARES JaMWindExDefs, StreamDefs =
  BEGIN

  -- common types

  WindowHandle: TYPE = WindowDefs.WindowHandle;
  DisplayHandle: TYPE = StreamDefs.DisplayHandle;
  KeyboardHandle: TYPE = StreamDefs.KeyboardHandle;
  StreamIndex: TYPE = StreamDefs.StreamIndex;
  Selection: TYPE = WindowDefs.Selection;
  Rptr: TYPE = RectangleDefs.Rptr;
  xCoord: TYPE = RectangleDefs.xCoord;
  yCoord: TYPE = RectangleDefs.yCoord;

  unDELchar: CHARACTER = LF;
  --

  ReadEditChar: PUBLIC PROCEDURE [w: WindowHandle, InString: STRING,
                       eol: PROCEDURE[CHARACTER]RETURNS[BOOLEAN]]
                RETURNS[STRING] =
  --read char and do editing.
  --If terminating char then current selection is returned
  --     with that char appended
    BEGIN
    ds:  DisplayHandle ← w.ds;
    eofindex:  StreamIndex ← w.eofindex;
    char:  CHARACTER ← w.ks.get[w.ks];
    IF InvertCursor[] THEN [] ← InvertCursor[];
    IF eol[char]
    THEN BEGIN
         DoInput[w];
         InString ← GetSel[w,InString];
         AppendChar[InString,char]; --always room, see GetSel
         RETURN[InString];
         END;
    IF EqualIndex[OriginIndex, eofindex] OR
       EqualIndex[w.selection.rightindex, ModifyIndex[eofindex,-1]]
    THEN BEGIN  -- at end of stream - do it now
         SELECT char FROM
         ControlA, ControlH => BackSpace[w];
         ControlW, ControlQ => BackWord[w];
```

```
        DEL => BackSelection[w];
        unDELchar => UnDelete[w];
        ENDCASE => MakeOrExtendSelection[w, char];
        END
    ELSE BEGIN  -- buffer input for batched execution
        PutArray[buffInput,buffIndex,char]; buffIndex ← buffIndex+1;
        IF w.ks.endof[w.ks] OR buffIndex>=buffInputLength THEN DoInput[w];
        END;
    RETURN[InString];
    END;

BackSpace: PROCEDURE[w: WindowHandle] =
  BEGIN
  index:  StreamIndex;
  IF EqualIndex[w.selection.leftindex,w.eofindex] THEN RETURN;
  IF InvertCursor[] THEN [] ← InvertCursor[];
  index ← ModifyIndex[w.eofindex, -1];
  IF w.ds.charx = leftmargin
  THEN BEGIN  -- rescan to find last character
       w.eofindex ← index;
       w.selection.rightindex ← ModifyIndex[index,-1];
       PaintDisplayWindow[w];
       END
  ELSE BEGIN
--       MarkSelection[w];
       SetIndex[w.file,index];
       ClearDisplayChar[w.ds, w.file.get[w.file]];
       w.selection.rightx ← w.ds.charx;
       w.selection.rightindex ← ModifyIndex[index, -1];
--       MarkSelection[w];
       w.eofindex ← index;
       END;
  END;

BackWord: PROCEDURE[w: WindowHandle] =
  BEGIN
  curind:  StreamIndex;
  gotNonSpace:  BOOLEAN ← FALSE;
  ch:  CHARACTER;
  IF InvertCursor[] THEN [] ← InvertCursor[];
  UNTIL EqualIndex[w.eofindex,w.selection.leftindex]
  DO curind ← ModifyIndex[w.eofindex,-1];
     SetIndex[w.file,curind];
     ch←w.file.get[w.file];
     IF ch=SP OR ch=TAB OR ch=CR
     THEN BEGIN IF gotNonSpace THEN EXIT END
     ELSE gotNonSpace ← TRUE;
     IF w.ds.charx#leftmargin
     THEN BEGIN
          ClearDisplayChar[w.ds,ch];
          w.selection.rightx ← w.ds.charx;
          END;
     w.selection.rightindex ← ModifyIndex[curind,-1];
     w.eofindex ← curind;
  ENDLOOP;
  IF w.ds.charx=leftmargin THEN PaintDisplayWindow[w];
  END;

BackSelection: PROCEDURE[w: WindowHandle] =
  BEGIN
  END;

UnDelete: PROCEDURE[w: WindowHandle] =
  BEGIN
  END;

MakeOrExtendSelection: PROCEDURE[w: WindowHandle, char: CHARACTER] =
  BEGIN OPEN JaMWEState;
  ds:  DisplayHandle ← w.ds;
  sel:  Selection;
  eofindex:  StreamIndex ← w.eofindex;
  -- now make/extend the current selection
  w.ds.put[w.ds, char];
  IF StreamDefs.EqualIndex[OriginIndex, eofindex] THEN
    BEGIN  --make this char the current selection
    sel ← [
```

```
          leftx:  ds.charx - RectangleDefs.ComputeCharWidth[char, ds.pfont],
          leftline:  ds.line,
          leftindex:  eofindex,
          rightx:  ds.charx ,
          rightline:  ds.line,
          rightindex:  eofindex
          ];
    END
  ELSE
    BEGIN -- extend selection to include this char
    sel ← Selection[
          leftx:  w.selection.leftx,
          leftline:  IF w.selection.leftline#0
                       THEN w.selection.leftline
                       ELSE ds.line, --kludge since null selection clobbered in SetDisplayWindow
          leftindex:  w.selection.leftindex,
          rightx:  ds.charx ,
          rightline:  ds.line,
          rightindex:  eofindex
          ];
    END;
  MakeSelection[w, @sel];
  END;

--buffers for modifying transcript file

  buffInputLength: CARDINAL = 16; --part for contiguous keyboard input
  buffFileLength: CARDINAL = 128; --part for file chunks
  buffInput: POINTER ← AllocateHeapNode[(buffInputLength+buffFileLength)/2];
  buffFile: POINTER ← buffInput+buffInputLength/2;
  buffIndex: CARDINAL ← 0;

DoInput: PROCEDURE [w: WindowHandle] =
-- Execute contents of buffInput
  BEGIN
  ind,curind:  StreamIndex;
  eofindex:  StreamIndex ← w.eofindex;
--  sel:  Selection;
  i,nRead: CARDINAL;
  delInd: INTEGER;
  char: CHARACTER;
  IF buffIndex=0 THEN RETURN;
  ind ← ModifyIndex[w.selection.rightindex,1];
  SetIndex[w.file,ind];
  --arrange even word boundary
--  IF BITAND[ind.byte,1]#0 THEN
--    BEGIN
--    nextchar: CHARACTER = w.file.get[w.file];
--    SetIndex[w.file,ind];
--    w.file.put[w.file,GetArray[buf,0]];
--    MoveBytes[buffInput,0,1,buffIndex-1,0,0];
--    PutArray[buf,buffIndex-1,nextchar];
--    ind ← GetIndex[w.file];
--    END;
  -- get first buffer-full of rest of file
  nRead ← ReadBytes[w.file,buffFile,
                MIN[buffFileLength,DiffIndex[eofindex,ind]]];
  SetIndex[w.file,ind];
  -- execute the chars
  FOR i IN [0..buffIndex)
  DO SELECT (char ← GetArray[buffInput,i]) FROM
     ControlA, ControlH =>
        BEGIN
        curind ← GetIndex[w.file];
        IF ~EqualIndex[curind,w.selection.leftindex]
        THEN SetIndex[w.file,ModifyIndex[curind,-1]];
        END;
     ControlW, ControlQ =>
        BEGIN
        gotNonSpace:  BOOLEAN ← FALSE;
        ch:  CHARACTER;
        curind ← GetIndex[w.file];
        UNTIL EqualIndex[curind,w.selection.leftindex]
        DO curind ← ModifyIndex[curind,-1];
           SetIndex[w.file,curind];
           ch←w.file.get[w.file];
```

```
            IF ch=SP OR ch=TAB OR ch=CR
            THEN BEGIN IF gotNonSpace THEN EXIT END
            ELSE gotNonSpace ← TRUE;
            SetIndex[w.file,curind];  --in case end of loop
         ENDLOOP;
         END;
      DEL => SetIndex[w.file,w.selection.leftindex];
      ENDCASE => w.file.put[w.file,char];
   ENDLOOP;
   --now replace rest of file, moved either forwards or backwards
   curind ← GetIndex[w.file];
   delInd ← DiffIndex[curind,ind];
   SELECT TRUE FROM
   delInd>0 =>
         BEGIN  --extra chars
         IF nRead=buffFileLength
         THEN BEGIN
                  SetIndex[w.file,ind];
                  [] ← ReadBytes[w.file,buffInput,delInd];
                  UNTIL GrEqualIndex[ind,eofindex]
                  DO BEGIN
                     MoveBytes[buffInput,buffInputLength/2,0,nRead,
                           delInd/2,BITAND[delInd,1]];
                     SetIndex[w.file,ind];
                     WriteBytes[w.file,buffInput,nRead];
                     ind ← GetIndex[w.file];
                     MoveBytes[buffInput,nRead/2,BITAND[nRead,1],delInd,0,0];
                     nRead ← ReadBytes[w.file,buffFile,
                         MIN[buffFileLength,DiffIndex[eofindex,ind]]];
                     END;
                  ENDLOOP;
                  WriteBytes[w.file,buffInput,delInd];
                  END
         ELSE WriteBytes[w.file,buffFile,nRead];
         END;
   delInd<0 =>
         BEGIN  --less chars
         ind ← curind;
         curind ← ModifyIndex[ind,nRead];
         UNTIL GrEqualIndex[curind,eofindex]
         DO BEGIN
            SetIndex[w.file,ind];
            WriteBytes[w.file,buffFile,nRead];
            ind ← GetIndex[w.file];
            curind ← ModifyIndex[ind,-delInd];
            SetIndex[w.file,curind];
            nRead ← ReadBytes[w.file,buffFile,
                        MIN[buffFileLength,DiffIndex[eofindex,curind]]];
            END;
         ENDLOOP;
         END;
   ENDCASE  => WriteBytes[w.file,buffFile,nRead];  --replace buffer
   w.eofindex ← ModifyIndex[eofindex,delInd];
   buffIndex ← 0;
   -- adjust selection
   w.selection.rightindex ← ModifyIndex[w.selection.rightindex,delInd];
   IF w.ks.endof[w.ks] THEN PaintDisplayWindow[w];
   END;

PutArray: PROCEDURE [base: POINTER, index: CARDINAL, ch: CHARACTER] =
-- Put ch into array, odd numbered elements at low sig end
   BEGIN
   IF BITAND[index,1]=0
   THEN (base+index/2)↑ ←
         BITSHIFT[LOOPHOLE[ch,WORD],8] + BITAND[(base+index/2)↑,377B]
   ELSE (base+index/2)↑ ←
         LOOPHOLE[ch,WORD] + BITAND[(base+index/2)↑,177400B];
   END;

GetArray: PROCEDURE [base: POINTER, index: CARDINAL]
         RETURNS[ch: CHARACTER] =
-- Get ch from array, odd numbered elements at low sig end
   BEGIN
   IF BITAND[index,1]=0
   THEN ch ← LOOPHOLE[BITSHIFT[(base+index/2)↑,-8]]
   ELSE ch ← LOOPHOLE[BITAND[(base+index/2)↑,377B]];
```

```
  END;

ReadBytes: PROCEDURE [strm: StreamDefs.StreamHandle,
                memaddr: POINTER, nbytes: CARDINAL]
            RETURNS[nread: CARDINAL] =
  BEGIN
--  nread ← nbytes;
--  IF nbytes>1 THEN [] ← ReadBlock[strm,memaddr,nbytes/2];
--  IF BITAND[nbytes,1]#0
--  THEN PutArray[memaddr,nbytes-1,strm.get[strm]];
  i: CARDINAL;
  FOR i IN [0..nbytes)
  DO BEGIN
     IF strm.endof[strm] THEN ERROR; --should never happen
     PutArray[memaddr,i,strm.get[strm]];
     END;
  ENDLOOP;
  nread ← nbytes;
  END;

WriteBytes: PROCEDURE [strm: StreamDefs.StreamHandle,
                memaddr: POINTER, nbytes: CARDINAL] =
  BEGIN
--  IF nbytes>1 THEN [] ← WriteBlock[strm,memaddr,nbytes/2];
--  IF BITAND[nbytes,1]#0
--  THEN strm.put[strm,GetArray[memaddr,nbytes-1]];
  i: CARDINAL;
  FOR i IN [0..nbytes) DO strm.put[strm,GetArray[memaddr,i]]; ENDLOOP;
  END;

DiffIndex: PROCEDURE [ind1,ind2: StreamDefs.StreamIndex]
                RETURNS[diff: INTEGER] =
--Compute difference of indices ind1-ind2
  BEGIN
  diff ← (LOOPHOLE[ind1.page,INTEGER]-ind2.page)*512 +
          LOOPHOLE[ind1.byte,INTEGER]-ind2.byte;
  END;

  GetSel: PROCEDURE [w: WindowHandle, string: STRING]
                RETURNS[STRING] =
  -- put the selection into the Heapstring string
  -- signals IODefs.LineOverflow for longer string (like
  --   IODefs.ReadEditedString does)
    BEGIN
    count: CARDINAL ← 0;
    fa: AltoFileDefs.FA;
    IF EqualIndex[w.selection.rightindex, OriginIndex] THEN RETURN[string];
    GetFA[w.file, @fa];
    count ← DiffIndex[w.selection.rightindex,w.selection.leftindex]+1;
    WHILE count+1 > string.maxlength  -- +1 to leave room for terminating char
      DO string ← SIGNAL LineOverflow[string]; ENDLOOP;
    string.length ← 0;
    SetIndex[w.file, w.selection.leftindex];
    THROUGH [0..count)
      DO AppendChar[string, w.file.get[w.file]]; ENDLOOP;
    JumpToFA[w.file, @fa];
    RETURN[string];
    END;

  cursorOn: BOOLEAN ← FALSE;

  InvertCursor: PUBLIC PROCEDURE RETURNS [BOOLEAN] =
    BEGIN
    w: WindowHandle = GetCurrentDisplayWindow[];
    IF w = NIL OR  w.type # scriptfile OR w.selection.leftline=0
      THEN RETURN[cursorOn];
    RectangleDefs.InvertBoxInRectangle[
      w.rectangle, w.selection.rightx-1, 2,
      w.selection.rightline*w.ds.lineheight+1, w.ds.lineheight-2];
    RETURN[cursorOn ← ~cursorOn]
    END;

NoteNameError: PUBLIC PROCEDURE [w:WindowHandle, str:  STRING] =
  BEGIN OPEN JaMWEState;
  i: INTEGER;
  scratchstr: STRING;
```

```
  -- convert window into scratch and tell bad name
  IF w.type # scratch THEN
    BEGIN
    [scratchstr, i] ← AssignScratchFile[];
    WindowDefs.AlterWindowType[w, scratch, scratchstr];
    scratchfiles[i] ← w.file;
    SystemDefs.FreeHeapString[scratchstr];
    END;
  WriteMessageString[w, str];
  WriteMessageString[w, "FileNameError!"L];
  END;

WriteMessageString: PUBLIC PROCEDURE [w:WindowHandle, str: STRING] =
  BEGIN
  i: CARDINAL;
  -- write message
  FOR i IN [0..str.length) DO
    w.ds.put[w.ds, str[i]];
    ENDLOOP;
  w.ds.put[w.ds, 15B];
  END;

AssignScratchFile: PUBLIC PROCEDURE RETURNS[STRING, INTEGER] =
  BEGIN OPEN JaMWEState;
  zero: CARDINAL = LOOPHOLE['0];
  i: INTEGER;
  str: STRING;
  -- loop through array looking for a free one
  FOR i IN [0..maxscratch) DO
    IF scratchfiles[i] = NIL THEN
      BEGIN
      str ← SystemDefs.AllocateHeapString[8];
      StringDefs.AppendString[str, "Scratch"L];
      StringDefs.AppendChar[str, LOOPHOLE[i+zero,CHARACTER]];
      RETURN[str, i];
      END;
    ENDLOOP;
  END;

-- initialization for wmanager

  InitConfiguration: PROCEDURE =
    BEGIN OPEN JaMWindExDefs;
    START JaMWESelection[@JaMWEState];
    START JaMWEWindows[@JaMWEState];
    START JaMWEPosition[@JaMWEState];
    START JaMWEBreak[@JaMWEState]; -- must be started after JaMWEWindows
    START JaMWEMain[@JaMWEState];
    END;

  InitManager: PROCEDURE =
    BEGIN OPEN JaMWEState;
    -- Declare Locals
    i: CARDINAL;
    w: WindowHandle ← WindowDefs.GetCurrentDisplayWindow[];
    -- process and save currently extant windows
    FOR i IN [0..4) DO
      windows[i] ← w;
      IF w.link = windows[0] THEN EXIT
      ELSE w ← w.link;
      ENDLOOP;
    FOR i IN [0..maxscratch) DO
      scratchfiles[i] ← NIL;
      ENDLOOP;
    -- now init some stuff for later
    defaultmapdata ← RectangleDefs.GetDefaultBitmap[];
    defaultds ← StreamDefs.GetDefaultDisplayStream[];
    defaultks ← StreamDefs.GetDefaultKey[];
    [defaultfont, defaultlineheight] ← RectangleDefs.GetDefaultFont[];
    currentcursor ← textpointer;
    -- setup External Button Procedures
    ButtonProcArray ← TextProcArray;
--    ProcessDefs.Detach[FORK WindowExecutive];
    END;

  BBp: BitBltDefs.BBptr ← LOOPHOLE[BITAND[LOOPHOLE[
```

```
                SystemDefs.AllocateHeapNode[SIZE[BitBltDefs.BBTable]+1]+1],-2]];

  MoveBytes: PROCEDURE [base: POINTER, from: CARDINAL, fromoffset: [0..1],
                        nbytes: CARDINAL, to: CARDINAL, tooffset: [0..1]] =
--unlike ByteBlt, this always does right thing even if blocks overlap
  BEGIN OPEN BitBltDefs;
    IF nbytes=0 THEN RETURN;
    IF nbytes>7777B THEN ERROR; --shouldn't be allowed to happen
    BBp↑ ←
        [ pad: 0,
          sourcealt: FALSE,
          destalt: FALSE,                -- TRUE to use alternate memory bank
          sourcetype: block,
          function: replace,
          unused: 0,
          dbca: base,                    -- destination BaseCoreAddress
          dbmr: 77777B,                  -- destination raster width(in words)
          dlx: BITSHIFT[to,4]+BITSHIFT[tooffset,3],     -- destination left x
          dty: 0,                        -- destination top y
          dw: BITSHIFT[nbytes,3],
          dh: 1,
          sbca: base,                    -- source BaseCoreAddress
          sbmr: 77777B,                  -- source raster width(in words)
          slx: BITSHIFT[from,4]+BITSHIFT[fromoffset,3], -- source left x
          sty: 0,                        -- source top y
          gray0: 0,                      -- four words of "gray"
          gray1: 0,
          gray2: 0,
          gray3: 0];
    BITBLT[BBp];
  END;

  JaMWEState: JaMWindExDefs.JaMWEDataObject;

-- MAIN BODY CODE

  InitConfiguration[];
  InitManager[];

END. of JaMWEControl
```

```
--File: JaMWEMain.mesa

DIRECTORY
  AltoDefs: FROM "altodefs" USING [BytesPerPage],
  InlineDefs: FROM "inlinedefs" USING [BITOR, BITXOR],
  IODefs: FROM "iodefs" USING [BS, CR, DEL, ESC],
  KeyDefs: FROM "keydefs" USING [Keys, Mouse],
  OsStaticDefs: FROM "osstaticdefs" USING [OsStatics],
  RectangleDefs: FROM "rectangledefs" USING [
    ClearBoxInRectangle, CursorToRecCoords, GrayArray, GrayPtr, leftmargin,
    Rptr, xCoord, yCoord],
  SegmentDefs: FROM "segmentdefs" USING [FileNameError, InvalidFP],
  StreamDefs: FROM "streamdefs" USING [ModifyIndex,
    DisplayHandle, GetIndex, KeyboardHandle, OpenKeyStream,
    StreamError, StreamIndex],
  JaMWindExDefs: FROM "jamwindexdefs" USING [
    AMouseButton, InvertCursor, CursorArray, cursormap, CursorType,
    JumpStrip, KeySet, LMult, NullIndex, ReadEditChar, Shorten,
    slop, JaMWEDataHandle, xcursorloc, ycursorloc],
  WindowDefs: FROM "windowdefs" USING [
    AlterWindowType, FindDisplayWindow, GetCurrentDisplayWindow,
    GetSelection, Selection, SetCurrentDisplayWindow, SetFileForWindow,
    WindowHandle, MakeSelection];

DEFINITIONS FROM JaMWindExDefs, RectangleDefs, SegmentDefs, StreamDefs, WindowDefs;

JaMWEMain: PROGRAM [JaMWEState: JaMWEDataHandle]
  IMPORTS SegmentDefs, StreamDefs, RectangleDefs, WindowDefs,
    JaMWindExDefs
  EXPORTS JaMWindExDefs
  SHARES JaMWindExDefs, StreamDefs =
  BEGIN

  -- common types

  WindowHandle: TYPE = WindowDefs.WindowHandle;
  DisplayHandle: TYPE = StreamDefs.DisplayHandle;
  KeyboardHandle: TYPE = StreamDefs.KeyboardHandle;
  StreamIndex: TYPE = StreamDefs.StreamIndex;
  Selection: TYPE = WindowDefs.Selection;
  Rptr: TYPE = RectangleDefs.Rptr;
  xCoord: TYPE = RectangleDefs.xCoord;
  yCoord: TYPE = RectangleDefs.yCoord;

  -- Primary external interface:

  ReadEditedString: PUBLIC PROCEDURE[s: STRING,
                        eol: PROCEDURE[CHARACTER]RETURNS[BOOLEAN],
                        unused: BOOLEAN]
                  RETURNS[CHARACTER] =
  -- Reads a string from the keystream attached to the current window
  -- The current window may be changed during this procedure
    BEGIN
    w: WindowHandle ← WindowDefs.GetCurrentDisplayWindow[];
    sel: Selection;
    timer: POINTER TO INTEGER ← LOOPHOLE[430B];
    blinktime: INTEGER ← timer↑+8;
    s.length ← 0;
    OpenKeyStream[w.ks];
    sel ← [
          leftx:  w.ds.charx,
          leftline:  w.ds.line,
          leftindex:  w.eofindex,
          rightx:  w.ds.charx ,
          rightline:  w.ds.line,
          rightindex:  ModifyIndex[w.eofindex,-1]
          ];
    MakeSelection[w,@sel];
    UNTIL s.length # 0
    DO
      s ← DoWork[s, eol ! StreamDefs.StreamError => CONTINUE];
      IF blinktime-timer↑ ~IN[0..13] THEN
        BEGIN [] ← InvertCursor[]; blinktime ← timer↑+13 END;
      ENDLOOP;
    IF InvertCursor[] THEN [] ← InvertCursor[];
    s.length ← s.length - 1;
```

```
    RETURN[s[s.length]];
    END;

  DoWork: PROCEDURE[string: STRING,
                    eol: PROCEDURE[CHARACTER]RETURNS[BOOLEAN]]
              RETURNS[STRING] =
    BEGIN OPEN JaMWEState;
    -- Declare Locals
    x, y: INTEGER;
    k: KeySet;
    inJumpStrip: BOOLEAN ← FALSE;
    mousewindow: WindowHandle;
    buttons: AMouseButton;
    cw: WindowHandle ← WindowDefs.GetCurrentDisplayWindow[];

    -- check if need to service KeyStream
    IF NOT cw.ks.endof[cw.ks] THEN
      BEGIN ENABLE StreamDefs.StreamError => CONTINUE;
      string ← ReadEditChar[cw, string, eol];
      IF string.length # 0 THEN RETURN[string];
      END;
    -- check if some part of cursor is in jump bar
    [x, y] ← CursorToRectangleCoords[cw.rectangle, xcursorloc↑, ycursorloc↑];
    inJumpStrip ← x+slop > 0 AND x <= JumpStrip AND y+slop > 0 AND
      y-slop <= cw.rectangle.ch;
    SetJumpStripe[cw, inJumpStrip];
    -- check mouse buttons
    buttons ← GetMouseButton[];
    -- look at the mouse and flip from one to the other
    [mousewindow, x, y] ←
      WindowDefs.FindDisplayWindow[xcursorloc↑+cxa, ycursorloc↑+cya];
    IF ~inJumpStrip AND cw # mousewindow AND mousewindow # NIL AND
      buttons # None THEN
      BEGIN
      IF InvertCursor[] THEN [] ← InvertCursor[];
      WindowDefs.SetCurrentDisplayWindow[mousewindow !
        SegmentDefs.InvalidFP =>
          BEGIN
          WindowDefs.SetFileForWindow[mousewindow, mousewindow.name !
            SegmentDefs.FileNameError =>
              BEGIN
              WindowDefs.AlterWindowType[mousewindow, mousewindow.type, NIL];
              CONTINUE;
              END];
          RETRY;
          END];
      StreamDefs.OpenKeyStream[mousewindow.ks];
      END
    ELSE IF buttons = None THEN
      BEGIN OPEN OsStaticDefs;
      IF useKeyset AND OsStatics.AltoVersion.engineeringnumber # 4 --DO
        AND (k ← GetKeySet[]) # 0 THEN
          SELECT k FROM
            1B => StuffSel[cw];
            2B => PutChar[IODefs.CR];
            3B => BEGIN PutChar[IODefs.CR]; StuffSel[cw]; END;
            4B => PutChar[IODefs.ESC];
            10B => PutChar[IODefs.DEL];
            20B => PutChar[IODefs.BS];
            ENDCASE
      ELSE BEGIN
        IF OsStatics.AltoVersion.engineeringnumber = 4 THEN useKeyset ← FALSE;
        IF FL4Down[] THEN
          BEGIN StuffSel[cw]; WHILE FL4Down[] DO NULL ENDLOOP; END;
        END;
      END
    ELSE
      BEGIN
      THROUGH [0..700) DO NULL ENDLOOP;  -- Debounce Mouse
      ButtonProcArray[GetMouseButton[]][cw, xcursorloc↑+cxa,ycursorloc↑+cya];
      END;
    RETURN[string];
    END;

  SetJumpStripe: PUBLIC PROCEDURE[w: WindowHandle, flag: BOOLEAN] =
    BEGIN OPEN RectangleDefs, JaMWEState;
```

```
  -- Declare Locals
   r: Rptr = w.rectangle;
   y1, y2: yCoord;
   bytepos, wendpos, eof: LONG INTEGER;
   longZero: LONG INTEGER ← 0;
   longOne: LONG INTEGER ← 1;
   barheight: CARDINAL = r.ch-defaultlineheight-2;
   barwidth: INTEGER = leftmargin-1;
   current, wendindex: StreamIndex;
   gray: GrayArray ← [125252B, 52525B, 125252B, 52525B];
   grayarray: GrayPtr = @gray;
   zeros: GrayArray ← [0,0,0,0];
   zeroarray: GrayPtr = @zeros;
  -- check if visible
   IF (w.rectangle.visible = FALSE) OR (w.file = NIL) THEN
     RETURN;
  -- now set or reset state
   IF flag THEN
     BEGIN
     IF currentcursor = botharrow THEN RETURN;
     SetCursor[botharrow];
     ButtonProcArray ← ScrollProcArray;
     ClearBoxInRectangle[r, 1, barwidth, defaultlineheight+1, barheight, grayarray];
     current ← IF w.tempindex = NullIndex THEN w.fileindex ELSE w.tempindex;
     -- compute position in file and paint(reset) position
     bytepos ← IF current.byte=177777B THEN longZero
       ELSE LMult[current.page, AltoDefs.BytesPerPage] + current.byte;
     wendindex ← StreamDefs.GetIndex[w.file];
     wendpos ← LMult[wendindex.page, AltoDefs.BytesPerPage] + wendindex.byte;
     eof ← IF w.eofindex.byte=177777B THEN longOne
       ELSE LMult[w.eofindex.page, AltoDefs.BytesPerPage] + w.eofindex.byte;
     IF wendpos > eof THEN eof ← wendpos;
     y1 ← Shorten[(bytepos*barheight)/eof];
     y2 ← Shorten[(wendpos*barheight)/eof];
     y2 ← MAX[y2-y1, 2];
     ClearBoxInRectangle[
       r, 3, barwidth-5, y1+defaultlineheight+1, y2, zeroarray];
     END
   ELSE
     BEGIN
     SetCursor[textpointer];
     ButtonProcArray ← TextProcArray;
     ClearBoxInRectangle[
       r, 1, barwidth, defaultlineheight+1, barheight, zeroarray]
     END;
   END;

 GetMouseButton: PUBLIC PROCEDURE RETURNS[AMouseButton]=
   BEGIN
   RETURN[KeyDefs.Mouse.buttons];
   END;

 GetKeySet: PUBLIC PROCEDURE RETURNS [KeySet]=
   BEGIN OPEN InlineDefs;
   n, keyvalues: KeySet ← 0;
   DO
     n ← BITXOR[KeyDefs.Mouse.keyset, 37B];
     IF n = 0 THEN
       BEGIN
       THROUGH [0..200) DO NULL ENDLOOP;
       n ← BITXOR[KeyDefs.Mouse.keyset, 37B];
       END;
     IF n = 0 THEN RETURN[keyvalues] ELSE keyvalues ← BITOR[keyvalues, n];
     ENDLOOP;
   END;

 FL4Down: PROCEDURE RETURNS [BOOLEAN] =
   BEGIN OPEN KeyDefs;
   RETURN[KeyDefs.Keys.FL4 = down];
   END;

 PutChar: PROCEDURE [c: CHARACTER]=
   BEGIN OPEN JaMWEState;
   defaultks.putback[defaultks, c];
   END;
```

```
  StuffSel: PROCEDURE [w: WindowHandle]=
    BEGIN OPEN JaMWEState;
    s: STRING ← WindowDefs.GetSelection[w];
    i: CARDINAL;
    IF s = NIL THEN RETURN;
    FOR i DECREASING IN [0..s.length) DO
      defaultks.putback[defaultks, s[i]];
      ENDLOOP;
    END;

  CursorToRectangleCoords: PUBLIC PROCEDURE [rectangle: Rptr, x: xCoord, y: yCoord]
    RETURNS[xCoord, yCoord] =
    BEGIN OPEN JaMWEState;
    -- refinements for sensitive points of each cursor
    x ← x + cxa;
    y ← y + cya;
    -- convert cursor coordinates to window coordinates
    [x, y] ← RectangleDefs.CursorToRecCoords[rectangle, x, y];
    RETURN[x, y];
    END;

NullProc: PUBLIC PROCEDURE[w: WindowHandle, x: xCoord, y: yCoord] =
  BEGIN
  RETURN;
  END;

  Cursors: ARRAY CursorType OF CursorArray = [
    -- textpointer
    [100000B, 140000B, 160000B, 170000B, 174000B, 176000B, 177000B, 170000B, 154000B,
      114000B, 6000B, 6000B, 3000B, 3000B, 1400B,1400B],

    -- arrow
    [40B, 60B, 70B, 74B, 177776B, 177777B, 177776B, 74B, 70B, 60B, 40B, 0, 0, 0, 0, 0],

    -- bullseye
    [3700B, 7740B, 14060B, 30030B, 60014B, 140006B, 141606B, 141606B, 141606B,
      140006B, 60014B, 30030B, 14060B, 7740B, 3700B, 0B],

    -- leftbutton
    [177740B, 100040B, 135240B, 135240B, 135240B, 135240B, 135240B, 100040B, 100040B,
      100040B, 100040B, 100040B, 100040B, 100040B, 100040B, 177740B],

    -- uparrow
    [600B, 1700B, 7760B, 37776B, 177777B, 7760B, 7760B, 7760B, 7760B, 7760B, 7760B,7760B,
      7760B, 7760B,7760B, 7760B],

    -- downarrow
    [7760B, 7760B, 7760B,7760B, 7760B, 7760B,7760B, 7760B, 7760B,7760B, 7760B,177777B,
      37776B, 7760B, 1700B, 600B],

    -- botharrow
    [600B, 1700B, 7760B, 37776B  177777B, 7760B, 7760B, 7760B, 7760B, 7760B, 7760B,
      177777B, 37776B, 7760B, 1700B, 600B],

    -- hourglass
    [177777B, 100001B, 40002B, 34034B, 17170B, 7660B, 3740B, 1700B, 1100B, 2440B, 4220B,
      10610B, 21704B, 47762B, 177777B, 177777B],

    -- norm
    [37774B, 37774B, 34034B, 34034B, 34034B, 34034B, 34034B, 34034B, 34034B, 34034B,
      34034B, 34034B, 34034B, 34034B, 37774B, 37774B],

    -- menu
    [1000B, 3001B, 7003B, 36007B, 177776B, 177776B, 36007B, 7003B,
      3001B, 1000B, 0B, 0B, 0B, 0B, 0B, 0B]];

  SetCursor: PUBLIC PROCEDURE [type: CursorType] =
    BEGIN OPEN JaMWEState;
    XAdjust: PACKED ARRAY CursorType OF [0..16) = [0,15,7,0,8,8,0,0,8,0];
    YAdjust: PACKED ARRAY CursorType OF [0..16) = [0,8,7,0,0,15,0,0,8,5];
    Cursor: POINTER TO CursorArray = cursormap;
    currentcursor ← type;
    cya ← YAdjust[type];
    cxa ← XAdjust[type];
    Cursor↑ ← Cursors[type];
    END;
```

```
END. of JaMWEMain
```

```
--File: JaMWEPosition.mesa
--Edited by:
--          Sandman March 29, 1978  2:02 PM
--          Barbara September 21, 1978  4:13 PM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [BytesPerPage],
  RectangleDefs: FROM "rectangledefs" USING [ComputeCharWidth, leftmargin],
  StreamDefs: FROM "streamdefs" USING [
    EqualIndex, GetIndex, GrEqualIndex, SetIndex],
  JaMWindExDefs: FROM "jamwindexdefs" USING [
    CursorToRectangleCoords, GetMouseButton, JumpStrip, LDivMod, LMult,
    maxlines, NullIndex, NullProc, OriginIndex, SetCursor, SetJumpStripe,
    slop, JaMWEDataHandle,xcursorloc, ycursorloc],
  WindowDefs: FROM "windowdefs" USING [
    GetCurrentDisplayWindow, GetLineTable, PaintDisplayWindow,
    ResolveBugToPosition, StreamIndex, WindowHandle, xCoord, yCoord];

DEFINITIONS FROM StreamDefs, RectangleDefs, WindowDefs, JaMWindExDefs;

JaMWEPosition: PROGRAM [JaMWEState:  JaMWEDataHandle]
  IMPORTS StreamDefs, RectangleDefs, WindowDefs, JaMWindExDefs
  EXPORTS JaMWindExDefs
  SHARES StreamDefs, JaMWindExDefs =
  BEGIN
  OPEN JaMWEState;

  CR: CHARACTER = 15C;
  Space: CHARACTER = 40C;

  PositionFile: PUBLIC PROCEDURE[w: WindowHandle, x: xCoord, y: yCoord]=
    BEGIN
    -- Declare Locals
    height: CARDINAL;
    bytepos, eof: LONG INTEGER;
    index: StreamIndex;
    -- compute position in file and set it
    SetCursor[arrow];
    ButtonWait;
    SetCursor[hourglass];
    x ← xcursorloc↑; y ← ycursorloc↑;
    [x, y] ← CursorToRectangleCoords[w.rectangle, x, y];
    -- if out of jump bar then no scrolling
    IF NOT CheckForSlop[w, x, y] THEN
      BEGIN
      SetJumpStripe[w, FALSE];
      RETURN;
      END;
    IF y < defaultlineheight+1 OR w.eofindex.byte = 177777B THEN index ← [0, 0]
    ELSE
      BEGIN OPEN AltoDefs;
      height ← w.rectangle.ch-(defaultlineheight+1);
      y ← MIN[LOOPHOLE[y-(defaultlineheight+1), CARDINAL], height];
      IF y = height THEN index ← w.eofindex
      ELSE
        BEGIN
        eof ← LMult[w.eofindex.page, BytesPerPage] + w.eofindex.byte;
        bytepos ← (eof*y)/height;
        [index.page, index.byte] ← LDivMod[bytepos, BytesPerPage];
        IF index.page > w.eofindex.page OR (index.page = w.eofindex page
          AND index.byte > w.eofindex.byte) THEN index ← w.eofindex,
        END;
      END;
    DoTheScroll[w, index];
    END;

  ScrollUpFile: PUBLIC PROCEDURE[w: WindowHandle, x: xCoord, y: yCoord]=
    BEGIN
    -- Declare Locals
    index: StreamIndex;
    line:  INTEGER;
    -- compute position in file and set it
    SetCursor[uparrow];
    ButtonWait[];
    x ← xcursorloc↑+cxa; y ← ycursorloc↑+cxa;
    [line, , ,index] ← ResolveBugToPosition[w, x, y];
```

```
    [x, y] ← CursorToRectangleCoords[w.rectangle, x-cxa, y-cya];
    SetCursor[hourglass];
    -- if out of jump bar then no scrolling
    IF NOT CheckForSlop[w, x, y] OR line = 1 THEN
      BEGIN
      SetJumpStripe[w, FALSE];
      RETURN;
      END;
    DoTheScroll[w, index];
    END;

  ScrollDownFile: PUBLIC PROCEDURE[w: WindowHandle, x: xCoord, y: yCoord]=
    BEGIN
    -- Declare Locals
    index, posindex: StreamIndex;
    maxbackup, pos: LONG INTEGER;
    line,nlines:  CARDINAL;
    nlines ← (w.rectangle.ch/w.ds.lineheight)-1;
    -- compute position in file and set it
    SetCursor[downarrow];
    ButtonWait[];
    x ← xcursorloc↑; y ← ycursorloc↑;
    [x, y] ← CursorToRectangleCoords[w.rectangle, x, y];
    SetCursor[hourglass];
    line ← MIN[LOOPHOLE[MAX[1, y/w.ds.lineheight],CARDINAL], nlines];
    posindex ← SELECT w.type FROM
      scratch, scriptfile =>
        IF w.tempindex = NullIndex THEN w.fileindex ELSE w.tempindex,
      file => w.fileindex,
      ENDCASE => OriginIndex;
    pos ← LMult[posindex.page, AltoDefs.BytesPerPage] + posindex.byte;
    -- if out of jump bar or first window then nop
    IF NOT CheckForSlop[w, x, y] OR EqualIndex[posindex, OriginIndex] THEN
      BEGIN
      SetJumpStripe[w, FALSE];
      RETURN;
      END;
    maxbackup ← LMult[w.rectangle.cw/ComputeCharWidth[Space,w.ds.pfont], line];
    IF pos > maxbackup THEN
      BEGIN
      maxbackup ← pos-maxbackup;
      [index.page, index.byte] ← LDivMod[maxbackup, AltoDefs.BytesPerPage];
      END
    ELSE index ← OriginIndex;
    index ← GenerateLineTable[w,index,posindex,line,nlines];
    DoTheScroll[w, index];
    END;

  NormalizeSelection: PUBLIC PROCEDURE[w: WindowHandle, x: xCoord, y: yCoord]=
    BEGIN
    linestarts: DESCRIPTOR FOR ARRAY OF StreamIndex;
    maxbackup, pos: LONG INTEGER;
    index: StreamIndex;
    lastindex: StreamIndex ← NullIndex;
    line, nlines, i:  CARDINAL;
    nlines ← (w.rectangle.ch/w.ds.lineheight)-1;
    linestarts ← DESCRIPTOR[GetLineTable[], nlines+1];
    -- compute position in file and set it
    SetCursor[norm];
    ButtonWait[];
    x ← xcursorloc↑; y ← ycursorloc↑;
    [x, y] ← CursorToRectangleCoords[w.rectangle, x, y];
    SetCursor[hourglass];
    line ← MIN[MAX[1, y/w.ds.lineheight], nlines];
    -- if out of jump bar then nop
    IF NOT CheckForSlop[w, x, y] THEN
      BEGIN
      SetJumpStripe[w, FALSE];
      RETURN;
      END;
    FOR i IN [1..nlines) DO
      IF EqualIndex[NullIndex, linestarts[i]] THEN
        BEGIN lastindex ← w.eofindex; EXIT; END;
      REPEAT
        FINISHED => lastindex ← linestarts[nlines];
      ENDLOOP;
```

```
    --if no selection or no scroll, simply move to beginning of file
    IF EqualIndex[w.selection.leftindex, NullIndex] OR
     --selection past current end-of-file
     GrEqualIndex[w.selection.leftindex, lastindex] OR
     (EqualIndex[linestarts[0], OriginIndex]
     AND line > w.selection.leftline)
      THEN index ← OriginIndex
    -- selection visible and below bug
    ELSE IF w.selection.leftline # 0 AND
     (GrEqualIndex[w.selection.leftindex, linestarts[line-1]]
     OR line <= 2 * w.selection.leftline)
      THEN index ← linestarts[IF w.selection.leftline >= line
        THEN w.selection.leftline - line ELSE line - w.selection.leftline]
    -- adjustments necessary
    ELSE BEGIN
      pos ← LMult[w.selection.leftindex.page, AltoDefs.BytesPerPage] +
        w.selection.leftindex.byte;
      maxbackup ← LMult[w.rectangle.cw/ComputeCharWidth[Space,w.ds.pfont], line];
      IF pos > maxbackup THEN
        BEGIN
        maxbackup ← pos - maxbackup;
        [index.page, index.byte] ← LDivMod[maxbackup, AltoDefs.BytesPerPage];
        END
      ELSE index ← OriginIndex;
      -- get within window range
      index ← GenerateLineTable[w,index,w.selection.leftindex,line,nlines];
      END;
    DoTheScroll[w, index];
    END;

  CheckForSlop: PROCEDURE[w:  WindowHandle, x:  xCoord, y:  yCoord]
    RETURNS[BOOLEAN]=
    BEGIN
    flag: BOOLEAN ← FALSE;
    --check if some part of cursor is in jump bar
    IF (x+slop > 0 AND x <= JumpStrip + 15 AND y+slop > 0
      AND y - slop <= w.rectangle.ch)
      THEN flag ← TRUE;
    RETURN[flag];
    END;

  ButtonWait: PROCEDURE=
    BEGIN
    --wait until all mouse buttons are up
    UNTIL GetMouseButton[] = None DO NULL; ENDLOOP;
    RETURN;
    END;

  DoTheScroll: PROCEDURE[w: WindowHandle, index: StreamIndex]=
    BEGIN
    SELECT w.type FROM
      clear => NULL;
      random => NULL;
      scratch,
      scriptfile =>
        BEGIN
        IF index = w.tempindex THEN RETURN;
        w.tempindex ← index;
        w.ds.options.StopBottom ← TRUE;
        IF w = GetCurrentDisplayWindow[] THEN
          BEGIN
          PaintDisplayWindow[w];
          END;
        END;
      file =>
        BEGIN
        IF index = w.fileindex THEN RETURN;
        w.fileindex ← index;
        IF w = GetCurrentDisplayWindow[] THEN
          BEGIN
          PaintDisplayWindow[w];
          END;
        END;
     ENDCASE;
   -- say not in jump mode anymore
   SetJumpStripe[w, FALSE];
```

```
  END;

GenerateLineTable:  PROCEDURE [w: WindowHandle, topindex, find: StreamIndex,
 line, big: CARDINAL] RETURNS [StreamIndex]  =
  BEGIN
  ptr: ARRAY[0..maxlines) OF StreamIndex;
  i, x: CARDINAL;
  char: CHARACTER;
  once: BOOLEAN ← TRUE;
  formatting: BOOLEAN ← FALSE;
  index, savedindex: StreamIndex;
  x ← leftmargin;
  savedindex ← GetIndex[w.file];
  SetIndex[w.file, topindex];
  index ← topindex;
  FOR i IN [0..big) DO
    ptr[i] ← NullIndex;
    ENDLOOP;
  i ← 0;
  -- generate the table
  WHILE NOT EqualIndex[index, find] DO
    index ← GetIndex[w.file];
    char ← w.file.get[w.file];
    IF formatting AND char # CR THEN LOOP;
    IF char = 32C THEN
      BEGIN formatting ← TRUE; LOOP END;
    x ← x + ComputeCharWidth[char,w.ds.pfont];
    IF x >= w.rectangle.cw OR char = CR THEN
      BEGIN
      x ← leftmargin;
      IF char = CR THEN
        BEGIN formatting ← FALSE; index ← GetIndex[w.file]; END;
      ptr[i] ← index;
      i ← (i + 1) MOD big;
      END;
    ENDLOOP;
  index ← ptr[LOOPHOLE[big-line+i, CARDINAL] MOD big];
  IF NOT EqualIndex[index, NullIndex] THEN topindex ← index;
  SetIndex[w.file,savedindex];
  RETURN[topindex];
  END;

-- initialization for position module

InitPosition:  PROCEDURE =
  BEGIN
  ScrollProcArray[RedYellowBlue] ← NullProc;
  ScrollProcArray[RedBlue] ← NullProc;
  ScrollProcArray[RedYellow] ← NullProc;
  ScrollProcArray[Red] ← ScrollUpFile;
  ScrollProcArray[BlueYellow] ← NormalizeSelection;
  ScrollProcArray[Blue] ← ScrollDownFile;
  ScrollProcArray[Yellow] ← PositionFile;
  ScrollProcArray[None] ← NullProc;
  END;

--MAIN BODY CODE

InitPosition[];


END. of weposition
```

```
--File:  JaMWESelection.mesa
--Edited by:
--           Sandman  April 21, 1978  11:55 AM
--           Barbara  September 21, 1978  4:39 PM

DIRECTORY
  AltoFileDefs: FROM "altofiledefs" USING [FA],
  RectangleDefs: FROM "rectangledefs" USING [
    ComputeCharWidth, FAptr, leftmargin, xCoord, yCoord],
  StreamDefs: FROM "streamdefs" USING [GrEqualIndex, EqualIndex,
    GetFA, GrIndex, JumpToFA, ModifyIndex, SetIndex, StreamError],
  JaMWindExDefs: FROM "jamwindexdefs" USING [InvertCursor,
    AMouseButton, GetKeySet, GetMouseButton, MenuSelect, NullProc,
    JaMWEDataHandle, xcursorloc, ycursorloc],
  WindowDefs: FROM "windowdefs" USING [
    GetLineTable, MakeSelection, ResolveBugToPosition, Selection, StreamIndex,
    WindowHandle];

DEFINITIONS FROM StreamDefs, WindowDefs, RectangleDefs, JaMWindExDefs;

JaMWESelection:  PROGRAM [JaMWEState:  JaMWEDataHandle]
  IMPORTS WindowDefs, StreamDefs, RectangleDefs, JaMWindExDefs
  EXPORTS JaMWindExDefs
  SHARES StreamDefs, JaMWindExDefs =
BEGIN

OPEN JaMWEState;

CR: CHARACTER = 15C;
SP: CHARACTER = 40C;
TAB: CHARACTER = 11C;

Finder: TYPE = PROCEDURE [
  w: WindowHandle, sel: POINTER TO Selection, x: xCoord, y: yCoord];

FindTheChar: Finder =
  BEGIN
  width: INTEGER;
  [sel.leftline, sel.leftx, width, sel.leftindex]
    ← ResolveBugToPosition[w, x, y];
  IF GrEqualIndex[sel.leftindex,w.eofindex]
  THEN BEGIN
        sel.leftindex ← w.eofindex;
        sel.rightindex ← ModifyIndex[w.eofindex,-1];
        sel.rightx ← sel.leftx;
        END
  ELSE  BEGIN
        sel.rightindex ← sel.leftindex;
        sel.rightx ← sel.leftx + width;
        END;
  sel.rightline ← sel.leftline;
  END;

TextSelect: PUBLIC PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord] =
  BEGIN
  Select[w, x, y, FindTheChar, Red];
  END;

WordSelect: PUBLIC PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord] =
  BEGIN
  Select[w, x, y, FindTheWord, Yellow];
  END;

Select: PROCEDURE [
  w: WindowHandle, x: xCoord, y: yCoord, find: Finder, button: AMouseButton] =
  BEGIN
  -- Declare Locals
  selection: Selection;
  exselection: Selection;
  sel: POINTER TO Selection ← @selection;
  exsel: POINTER TO Selection ← @exselection;
  fa: AltoFileDefs.FA;

  IF w.file = NIL THEN RETURN;
  GetFA[w.file, @fa];
  IF InvertCursor[] THEN [] ← InvertCursor[];
```

```
  find[w, sel, x, y];
  MakeSelection[w, sel];
  --  check for extensions
  WHILE GetMouseButton[] = button DO
    IF x # xcursorloc↑+cxa OR y # ycursorloc↑+cya THEN
      BEGIN
      x ← xcursorloc↑; y ← ycursorloc↑;
      find[w, exsel, x, y];
      IF GrIndex[sel.leftindex, exsel.leftindex] THEN
        BEGIN
        exsel.rightx ← sel.rightx;
        exsel.rightline ← sel.rightline;
        exsel.rightindex ← sel.rightindex;
        END;
      IF GrIndex[exsel.rightindex, sel.rightindex] THEN
        BEGIN
        exsel.leftx ← sel.leftx;
        exsel.leftline ← sel.leftline;
        exsel.leftindex ← sel.leftindex;
        END;
      IF w.selection # exsel↑ THEN MakeSelection[w, exsel];
      END;
    ENDLOOP;
  JumpToFA[w.file, @fa];
  RETURN
  END;

FindTheWord: Finder =
  BEGIN
  char, newchar:  CHARACTER;
  linestarts:  DESCRIPTOR FOR ARRAY OF StreamIndex;
  index, newindex, charindex:  StreamIndex;
  pos, charpos:  xCoord;
  nlines, charline, line, width, charwidth: CARDINAL;
  class, newclass: Class;

  nlines ← (w.rectangle.ch/w.ds.lineheight)-1;
  linestarts ← DESCRIPTOR[GetLineTable[],nlines];
  [charline, charpos, width, charindex] ← ResolveBugToPosition[w, x, y];
  IF GrEqualIndex[charindex,w.eofindex]
  THEN  BEGIN
        charindex ← ModifyIndex[w.eofindex,-1];
        charpos ← charpos - width;
        END;
  line ← charline - 1;  -- line 0 is first line of text in window
  SetIndex[w.file, charindex];
  char ← w.file.get[w.file];
  class ← CharClass[char];
  IF charindex = linestarts[line] AND line = 0 THEN pos ← leftmargin
  ELSE
    BEGIN
    IF charindex = linestarts[line] THEN line ← line - 1;
    newindex ← charindex;
    DO
      newindex ← ModifyIndex[newindex, -1];
      SetIndex[w.file, newindex];
      newchar ← w.file.get[w.file];
      newclass ← CharClass[newchar];
      IF newclass # class AND newclass # controlZ THEN GOTO foundend;
      IF newindex = linestarts[line] THEN -- linestarts[0] is first line
        IF line # 0 THEN line ← line - 1 ELSE GOTO startwindow;
      REPEAT
        startwindow => pos ← leftmargin;
        foundend =>
        BEGIN
        newindex ← ModifyIndex[newindex, 1];
        IF newindex = linestarts[line+1] THEN line ← line+1;
        SetIndex[w.file, linestarts[line]];
        pos ← leftmargin;
        FOR index ← linestarts[line], ModifyIndex[index, 1]
        UNTIL index = newindex DO
          newchar ← w.file.get[w.file];
          pos ← pos + (IF newchar = 11C THEN ComputeTabWidth[w.ds.pfont, pos]
            ELSE ComputeCharWidth[newchar, w.ds.pfont]);
          ENDLOOP;
        END;
```

```
      ENDLOOP;
    END;
  sel.leftx ← pos; sel.leftline ← line+1; sel.leftindex ← newindex;

--now scan downstream to find end of word
  line ← charline - 1;  -- line 0 is first line of text in window
  newindex ← ModifyIndex[charindex, 1];
  SetIndex[w.file, newindex];
  pos ← charpos + width;
  DO
    IF EqualIndex[newindex,w.eofindex] THEN GOTO foundend;
    newchar ← w.file.get[w.file ! StreamError => EXIT];
    newclass ← CharClass[newchar];
    IF newclass # class OR newclass = controlZ THEN GOTO foundend;
    charwidth ← IF newchar = 11C THEN ComputeTabWidth[w.ds.pfont, pos]
      ELSE ComputeCharWidth[newchar, w.ds.pfont];
    IF line+1 = nlines AND pos + charwidth >= w.rectangle.cw THEN
      GOTO foundend;
    IF newindex = linestarts[line+1] THEN
      BEGIN
      line ← line + 1;
      pos ← leftmargin;
      END;
    pos ← pos + charwidth;
    newindex ← ModifyIndex[newindex, 1];
    REPEAT
      foundend => newindex ← ModifyIndex[newindex, -1];
    ENDLOOP;
  IF GrEqualIndex[newindex,w.eofindex]
  THEN newindex ← ModifyIndex[w.eofindex,-1];
  sel.rightx ← pos; sel.rightline ← line+1; sel.rightindex ← newindex;
  RETURN
  END;

Class: TYPE = {alphanumeric, other, return, controlZ, space};

CharClass: PROCEDURE [char: CHARACTER] RETURNS [Class] =
  BEGIN
  RETURN[SELECT char FROM
    32C => controlZ,
    IN ['a..'z], IN ['A..'Z], IN ['0..'9], '., '/, '$ => alphanumeric,
    SP, TAB => space,
    CR => return,
    ENDCASE => other]
  END;

ComputeTabWidth: PROCEDURE [font: FAptr, x: xCoord] RETURNS [CARDINAL] =
  BEGIN
  tw: CARDINAL = ComputeCharWidth[' ,font] * 8;
  RETURN[tw - LOOPHOLE[x-leftmargin, CARDINAL] MOD tw]
  END;

CommandStuff: PUBLIC PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord]=
  BEGIN
  n: CARDINAL;
  IF ~useKeyset THEN RETURN;
  n ← GetKeySet[];
  IF w.ks # NIL THEN
    SELECT n FROM
      IN [1..26] => w.ks.putback[w.ks, 101B+n-1];
      27 => w.ks.putback[w.ks, '+];
      31 => w.ks.putback[w.ks, 1C]; -- Control A
      ENDCASE;
  END;

-- initialization for selection module

InitSelection:  PROCEDURE =
  BEGIN
  TextProcArray[RedYellowBlue] ← NullProc;
  TextProcArray[RedBlue] ← NullProc;
  TextProcArray[RedYellow] ← CommandStuff;
  TextProcArray[Red] ← TextSelect;
  TextProcArray[BlueYellow] ← NullProc;
  TextProcArray[Blue] ← MenuSelect;
  TextProcArray[Yellow] ← WordSelect;
```

```
  TextProcArray[None] ← NullProc;
  END;

-- MAIN BODY CODE
InitSelection[];

END. of weselection
```

```
--File: JaMWEWindows.mesa

DIRECTORY
  MenuDefs: FROM "menudefs" USING [DestroyMenu, MenuItem],
  OsStaticDefs: FROM "osstaticdefs" USING [OsStatics],
  RectangleDefs: FROM "rectangledefs" USING [
    CreateRectangle, CursorToMapCoords, DestroyRectangle,
    GrowRectangle, MoveRectangle],
  SegmentDefs: FROM "segmentdefs" USING [FileNameError],
  StreamDefs: FROM "streamdefs" USING [
    CloseDiskStream, CreateDisplayStream, CreateKeyStream, EqualIndex,
    GetIndex, ModifyIndex, NormalizeIndex, OpenDiskStream, OpenKeyStream,
    SetIndex, StreamError],
  StringDefs: FROM "stringdefs" USING [InvalidNumber, StringToNumber],
  SystemDefs: FROM "systemdefs" USING [
    AllocateHeapNode, FreeHeapNode, FreeHeapString],
  JaMWindExDefs: FROM "jamwindexdefs" USING [
    AMouseButton, AssignScratchFile, CursorToRectangleCoords, CursorType,
    GetMouseButton, maxscratch, NoteNameError, NullIndex, OriginIndex,
    SetCursor, JaMWEDataHandle, xcursorloc, xmouseloc, ycursorloc, ymouseloc],
  WindowDefs: FROM "windowdefs" USING [
    AlterWindowType, CreateDisplayWindow, DestroyDisplayWindow, DisplayHandle,
    FindDisplayWindow, GetCurrentDisplayWindow, GetSelection, MarkSelection,
    PaintDisplayWindow, Rptr, SetIndexForWindow, StreamHandle, StreamIndex,
    WindowHandle,xCoord, yCoord];

DEFINITIONS FROM StreamDefs, MenuDefs, RectangleDefs, WindowDefs, JaMWindExDefs;

JaMWEWindows: PROGRAM [JaMWEState: JaMWEDataHandle]
  IMPORTS SegmentDefs, StreamDefs, SystemDefs, MenuDefs,
    RectangleDefs, WindowDefs, JaMWindExDefs, StringDefs
  EXPORTS JaMWindExDefs
  SHARES StreamDefs, JaMWindExDefs =

BEGIN

OPEN JaMWEState;

CR: CHARACTER = 15C;

-- some externals

nCommands: CARDINAL = 12;

create: CARDINAL = 0;
destroy: CARDINAL = 1;
move: CARDINAL = 2;
grow: CARDINAL = 3;
load: CARDINAL = 4;
stuff: CARDINAL = 5;
find: CARDINAL = 6;
break: CARDINAL = 7;
clear: CARDINAL = 8;
trace: CARDINAL = 9;
position: CARDINAL = 10;
keys: CARDINAL = 11;


PutSelect: PUBLIC PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord]=
  BEGIN
  -- Declare Locals
  i: CARDINAL;
  str: STRING;
  nw: WindowHandle;
  doit: BOOLEAN;
  [nw, doit] ← WindowFrontEnd[w, leftbutton];
  IF doit THEN
    BEGIN
    -- get current selection and jam it into the keyboard stream
    str ← GetSelection[w];
    IF nw.ks # NIL THEN
      FOR i DECREASING IN [0..str.length) DO
        nw.ks.putback[nw.ks, str[i]];
        ENDLOOP;
    SystemDefs.FreeHeapString[str];
    END;
```

```
  ButtonWait;
  SetCursor[textpointer];
  END;

  CreateWindow: PUBLIC PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord] =
    BEGIN
    i: CARDINAL;
    rectangle: Rptr;
    ds: DisplayHandle;
    ks: StreamHandle;
    name: STRING;
    nw: WindowHandle;
    mb: AMouseButton ← None;
    SetCursor[leftbutton];
    UNTIL (mb ← GetMouseButton[]) = Red DO
      IF mb = Blue THEN
        BEGIN ButtonWait; SetCursor[textpointer]; RETURN; END;
      ENDLOOP;
    SetCursor[hourglass];
    ks ← StreamDefs.CreateKeyStream[];
    [x,y] ← CursorToMapCoords[defaultmapdata, xcursorloc↑+cxa, ycursorloc↑+cya];
    rectangle ← CreateRectangle[defaultmapdata, x, 300, y, 75];
    [name, i] ← AssignScratchFile[];
    ds ← CreateDisplayStream[rectangle];
    nw ← CreateDisplayWindow[scratch, rectangle, ds, ks, name];
    scratchfiles[i] ← nw.file;
    SystemDefs.FreeHeapString[name];
    StreamDefs.OpenKeyStream[ks];
    ButtonWait;
    SetCursor[textpointer];
    END;

  DestroyWindow: PUBLIC PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord]=
    BEGIN
    i: CARDINAL;
    doit: BOOLEAN;
    nw: WindowHandle;
    [nw, doit] ← WindowFrontEnd[w, bullseye];
    IF doit THEN
      BEGIN
      -- check if we can delete this one
      FOR i IN [0..4) DO
        IF windows[i] = nw THEN
          BEGIN ButtonWait[]; SetCursor[textpointer]; RETURN; END;
        ENDLOOP;
      -- check if one of our scratch files is in the window now
      FOR i IN [0..maxscratch) DO
        IF scratchfiles[i] = nw.file THEN
          -- deallocate it (gets automatically deleted!)
          BEGIN scratchfiles[i] ← NIL; EXIT; END;
        ENDLOOP;
      -- get rid of all stuff in the window
      IF nw.ds # NIL THEN nw.ds.destroy[nw.ds];
      IF nw.ks # NIL THEN nw.ks.destroy[nw.ks];
      IF nw.menu # NIL THEN DestroyMenu[nw.menu];
      DestroyRectangle[nw.rectangle];
      DestroyDisplayWindow[nw];
      -- repaint current if not deleted one
      IF nw # w THEN PaintDisplayWindow[w]
      ELSE-- destroy set a new guy current!!
        BEGIN
        nw ← GetCurrentDisplayWindow[];
        StreamDefs.OpenKeyStream[nw.ks];
        MarkSelection[nw];
        END;
    END;
    ButtonWait;
    SetCursor[textpointer];
    END;

  MoveWindow: PUBLIC PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord] =
    BEGIN
    -- define locals
    savex, mapx: xCoord;
    savey, mapy: yCoord;
    mb: AMouseButton;
```

```
    r: Rptr = w.rectangle;
    -- jam cursor
    SetCursor[leftbutton];
    x ← xmouseloc↑ ← xcursorloc↑ ← r.bitmap.x0 + r.x0+cxa;
    y ← ymouseloc↑ ← ycursorloc↑ ← r.bitmap.y0 + r.y0+cya;
    [savex, savey] ← CursorToMapCoords[r.bitmap, x, y];
    -- while no buttons are down move it
    WHILE (mb ← GetMouseButton[]) # Red DO
      IF mb = Blue THEN BEGIN MoveRectangle[r,savex, savey]; EXIT; END;
      [mapx, mapy] ← CursorToMapCoords[r.bitmap, x, y];
      x ← IF INTEGER[mapx] < 0 THEN 0 ELSE mapx;
      y ← IF INTEGER[mapy] < 0 THEN 0 ELSE mapy;
      MoveRectangle[r, x, y];
      x ← xcursorloc↑;
      y ← ycursorloc↑;
      ENDLOOP;
    ButtonWait[];
    -- now paint it for possible cleanup
    PaintDisplayWindow[w];
    SetCursor[textpointer];
    END;

  GrowWindow: PUBLIC PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord]=
    BEGIN
    savewidth, width: xCoord;
    saveheight, height: yCoord;
    mb: AMouseButton;
    r: Rptr = w.rectangle;
    -- first move the cursor to lower right corner
    SetCursor[leftbutton];
    x ← r.bitmap.x0+r.x0+r.cw;
    xmouseloc↑ ← xcursorloc↑ ← x;
    y ← r.bitmap.y0+r.y0+r.ch;
    ymouseloc↑ ← ycursorloc↑ ← y;
    [savewidth, saveheight] ← CursorToRectangleCoords[w.rectangle, x, y];
    -- while no buttons are down grow it
    WHILE (mb ← GetMouseButton[]) # Red DO
      IF GetMouseButton[] = Blue THEN
        BEGIN GrowRectangle[r, savewidth, saveheight]; EXIT; END;
      [width, height] ← CursorToRectangleCoords[w.rectangle, x, y];
      width ← MAX[width, 35];
      height ← MAX[height, 35];
      GrowRectangle[r, width, height];
      x ← xcursorloc↑; y ← ycursorloc↑;
      ENDLOOP;
    ButtonWait[];
    -- now paint it for cleanup
    PaintDisplayWindow[w];
    SetCursor[textpointer];
    END;

  Load: PROCEDURE [w: WindowHandle, str: STRING]=
    BEGIN
    i: CARDINAL;
    -- check if one of our scratch files is in the window now
    FOR i IN [0..maxscratch) DO
      IF scratchfiles[i] = w.file THEN
      -- deallocate it (gets automatically deleted!)
        BEGIN scratchfiles[i] ← NIL; EXIT; END;
      ENDLOOP;
    IF str.length > 38 THEN str.length ← 38; -- max file name length
    AlterWindowType[w, file, str ! SegmentDefs.FileNameError =>
      BEGIN NoteNameError[w, str]; CONTINUE; END];
    w.ks ← defaultks;
    RETURN
    END;

  LoadThisWindow: PUBLIC PROCEDURE [w: WindowHandle]=
    BEGIN
    str: STRING;
    SetCursor[hourglass];
    str ← GetSelection[w];
    Load[w, str];
    SystemDefs.FreeHeapString[str];
    PaintDisplayWindow[w];
    SetCursor[textpointer];
```

```
    RETURN
    END;

  LoadWindow: PUBLIC PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord]=
    BEGIN
    -- define locals
    doit: BOOLEAN;
    nw: WindowHandle;
    [nw, doit] ← WindowFrontEnd[w, leftbutton];
    SetCursor[hourglass];
    IF doit AND windows[0] # nw THEN
      BEGIN
      str: STRING ← GetSelection[w];
      Load[nw, str];
      SystemDefs.FreeHeapString[str];
      IF w = nw THEN PaintDisplayWindow[w];
      END;
    ButtonWait[];
    SetCursor[textpointer];
    END;

SetPosition: PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord] =
  BEGIN
  -- Declare Locals
  str: STRING;
  nw: WindowHandle;
  scrollto: StreamIndex;
  doit: BOOLEAN;
  position: CARDINAL;
  [nw, doit] ← WindowFrontEnd[w, leftbutton];
  SetCursor[hourglass];
  -- get current selection
  IF doit AND NOT EqualIndex[w.selection.rightindex, NullIndex] THEN
    BEGIN
    str ← GetSelection[w];
    w ← GetCurrentDisplayWindow[];
    IF w # nw AND nw.file # NIL THEN
      OpenDiskStream[nw.file
        ! StreamError =>
          BEGIN
          nw.eofindex ← GetIndex[nw.file];
          RESUME
          END];
    position ← StringDefs.StringToNumber[str, 10
      ! StringDefs.InvalidNumber => GOTO badnumber];
    scrollto ← NormalizeIndex[[0, position]];
    SELECT nw.type FROM
      clear => NULL;
      random => NULL;
      scratch,
      scriptfile =>
        BEGIN
        IF scrollto = nw.tempindex THEN RETURN;
        nw.tempindex ← scrollto;
        nw.ds.options.StopBottom ← TRUE;
        END;
      file =>
        BEGIN
        IF scrollto = nw.fileindex THEN RETURN;
        nw.fileindex ← scrollto;
        END;
      ENDCASE;
    IF nw # w AND nw.file # NIL THEN CloseDiskStream[nw.file];
    SystemDefs.FreeHeapString[str];
    EXITS
      badnumber => SystemDefs.FreeHeapString[str];
    END;
  ButtonWait[];
  SetCursor[textpointer];
  END;

FindSelection: PROCEDURE[w: WindowHandle, x: xCoord, y: yCoord]=
  BEGIN
  -- Declare Locals
  str: STRING;
  nw: WindowHandle;
```

```
  scrollto: StreamIndex;
  doit: BOOLEAN;
  noscroll: BOOLEAN;
  [nw, doit] ← WindowFrontEnd[w, leftbutton];
  SetCursor[hourglass];
  -- get current selection
  IF doit AND NOT EqualIndex[w.selection.rightindex, NullIndex] THEN
     BEGIN
     str ← GetSelection[w];
     w ← GetCurrentDisplayWindow[];
     IF w # nw AND nw.file # NIL THEN
       OpenDiskStream[nw.file
         ! StreamError =>
           BEGIN
           nw.eofindex ← GetIndex[nw.file];
           RESUME
           END
           ];
     --do a text string search starting from the end of the current selection
     IF EqualIndex[nw.selection.rightindex, NullIndex] THEN
       nw.selection.rightindex ← OriginIndex;
     BEGIN
     [scrollto, nw.selection.leftindex, nw.selection.rightindex] ←
       TextSearch[nw, str, nw.selection.rightindex
         ! SearchFailed => GOTO fail];
     --scroll file to beginning of line containing text
     noscroll ← EqualIndex[scrollto, w.selection.rightindex];
     IF nw.type = scratch OR nw.type = scriptfile
       THEN nw.tempindex ← scrollto;
     IF NOT noscroll THEN SetIndexForWindow[nw, scrollto]
     ELSE IF nw = w THEN PaintDisplayWindow[nw];
     EXITS fail =>
       IF nw = w THEN PaintDisplayWindow[nw];
     END;
     IF nw # w AND nw.file # NIL THEN CloseDiskStream[nw.file];
     SystemDefs.FreeHeapString[str];
     END;
   ButtonWait[];
   SetCursor[textpointer];
   END;

  SearchFailed: ERROR = CODE;

  TextSearch: PROCEDURE [w: WindowHandle,  s: STRING, pos: StreamIndex]
    RETURNS [StreamIndex, StreamIndex, StreamIndex] =
    -- searches for s using the Knuth, Pratt, Morris algorithm.
    -- returns stream index of the start of the line containing s.
    BEGIN
    i, j: INTEGER;
    char: CHARACTER;
    l: INTEGER = s.length;
    offset: INTEGER ← 1;
    ff: DESCRIPTOR FOR ARRAY OF INTEGER; -- failure function
    IF l = 0 THEN ERROR SearchFailed;  -- empty string
    ff ← DESCRIPTOR[SystemDefs.AllocateHeapNode[l],l];
    -- set up failure function
    j ← 0; i ← ff[0] ← -1;
    WHILE j < l-1 DO
      WHILE i >= 0 AND s[j] # s[i] DO i ← ff[i] ENDLOOP;
      i ← i+1; j ← j+1;
      ff[j] ← IF s[j] = s[i] THEN ff[i] ELSE i;
      ENDLOOP;
    SetIndex[w.file,pos];
    char ← w.file.get[w.file];
    j ← 0;  -- j = pattern index
    DO ENABLE UNWIND => SystemDefs.FreeHeapNode[BASE[ff]];
      IF j >= l THEN EXIT;
      char ← w.file.get[w.file ! StreamError => GOTO fail];
      offset ← offset+1;
      IF char = CR THEN
        BEGIN pos ← ModifyIndex[pos,offset]; offset ← 0 END;
      WHILE j >= 0 AND char # s[j] DO j ← ff[j] ENDLOOP;
      j ← j + 1;
      REPEAT
        fail => ERROR SearchFailed;
      ENDLOOP;
```

```
    SystemDefs.FreeHeapNode[BASE[ff]];
    RETURN[pos, ModifyIndex[pos,offset-1], ModifyIndex[pos,offset-1]]
    END;

WindowFrontEnd: PUBLIC PROCEDURE [w: WindowHandle, cursor: CursorType]
  RETURNS [WindowHandle, BOOLEAN] =
  BEGIN
  -- Declare Locals
  nw: WindowHandle;
  x: xCoord; y: yCoord;
  mb: AMouseButton;
  -- ask for window to put stuff into
  SetCursor[cursor];
  -- wait to select a window or say ignore
  UNTIL (mb ← GetMouseButton[]) = Red DO
    IF mb = Blue THEN  RETURN[w, FALSE];
    ENDLOOP;
  -- now get selected window
  x ← xcursorloc↑+cxa; y ← ycursorloc↑+cya;
  [nw, x, y] ← FindDisplayWindow[x, y];
  IF nw = NIL THEN nw ← w;
  RETURN[nw, TRUE];
  END;

KeysetMessage: ARRAY BOOLEAN OF STRING ← ["Keys On", "Keys Off"];

EnableKeyset: PUBLIC PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord] =
  BEGIN
  IF OsStaticDefs.OsStatics.AltoVersion.engineeringnumber = 4 THEN
    useKeyset ← FALSE
  ELSE menuarray[keys].keyword ← KeysetMessage[useKeyset ← ~useKeyset];
  END;

ButtonWait: PROCEDURE =
  BEGIN
  -- wait until all button are up
  UNTIL GetMouseButton[] = None DO
    NULL;
    ENDLOOP;
  RETURN;
  END;

-- initialization for windows module

InitWindows: PROCEDURE =
  BEGIN OPEN SystemDefs;
  menuarray ←
    DESCRIPTOR[AllocateHeapNode[nCommands*SIZE[MenuItem]], nCommands];
  menuarray[create] ← MenuItem[" Create", CreateWindow];
  menuarray[destroy] ← MenuItem["Destroy", DestroyWindow];
  menuarray[move] ← MenuItem["  Move", MoveWindow];
  menuarray[grow] ← MenuItem["  Grow", GrowWindow];
  menuarray[load] ← MenuItem["  Load", LoadWindow];
  menuarray[stuff] ← MenuItem["Stuff It", PutSelect];
  menuarray[find] ← MenuItem["  Find", FindSelection];
  menuarray[position] ← MenuItem["Set Pos", SetPosition];
  menuarray[keys] ← MenuItem[KeysetMessage[useKeyset ← FALSE], EnableKeyset];
  END;

-- MAIN BODY CODE
InitWindows[];


END. of wmanwindows
```

```
--file JaMIO.mesa
--Written by John Warnock, January, 1979.
--Last Updated: April 11, 1979  2:16 PM by MN

DIRECTORY

EdStreamDefs: FROM "EdStreamDefs",
AltoDefs: FROM "AltoDefs",
RectangleDefs: FROM "RectangleDefs",
WindowDefs: FROM "WindowDefs",
DisplayDefs: FROM "DisplayDefs",
JaMMasterDefs: FROM "JaMMasterDefs",
JaMFnsDefs: FROM "JaMFnsDefs",
JaMStackDefs: FROM "JaMStackDefs",
JaMTypeChkDefs: FROM "JaMTypeChkDefs",
JaMExecDefs: FROM "JaMExecDefs",
JaMLiteralDefs: FROM "JaMLiteralDefs",
JaMVMDefs: FROM "JaMVMDefs",
JaMControlDefs: FROM "JaMControlDefs",
InlineDefs: FROM "InlineDefs",
JaMIODefs: FROM "JaMIODefs",
IODefs: FROM "IODefs",
SegmentDefs: FROM "SegmentDefs",
StreamDefs: FROM "StreamDefs";

JaMIO: PROGRAM
IMPORTS  EdStreamDefs,RectangleDefs,WindowDefs,DisplayDefs,
        JaMFnsDefs,IODefs,JaMTypeChkDefs,
        JaMLiteralDefs,JaMExecDefs,
        JaMVMDefs,JaMControlDefs,JaMStackDefs,
        SegmentDefs,StreamDefs
EXPORTS JaMIODefs
SHARES StreamDefs =
BEGIN
OPEN JaMMasterDefs,IODefs,JaMTypeChkDefs,JaMLiteralDefs,
JaMVMDefs,JaMControlDefs,JaMIODefs,JaMStackDefs,StreamDefs,InlineDefs;

-- RdLine and WrtString get and put StringType Objects on the stack.
ReadLine:PUBLIC PROCEDURE =
BEGIN
c:CHARACTER;
i:CARDINAL←0;
frame:Frame ←GetCurrentFrame[];
stringob:StringType Object←[lit,StringType[,,]];
streamob: StreamType Object ← DescStreamType[Pop[frame.opstk]];
[stringob.Address,stringob.Offset] ←AllocateCharsVM[0];
DO
IF streamob.SHandle.endof[streamob.SHandle] THEN
        BEGIN
        IF i = 0 THEN
                BEGIN
                JaMFnsDefs.PushBoolean[FALSE];
                streamob.SHandle.destroy[streamob.SHandle];
                RETURN;
                END
        ELSE
                BEGIN
                stringob.Length←i;
                [,]←AllocateCharsVM[i];
                Push[stringob,frame.opstk];
                JaMFnsDefs.PushBoolean[TRUE];
                RETURN;
                END;
        END
ELSE
        BEGIN
        c←streamob.SHandle.get[streamob.SHandle];
        IF c = 15C THEN
                BEGIN
                stringob.Length←i;
                [,]←AllocateCharsVM[i];
                Push[stringob,frame.opstk];
                JaMFnsDefs.PushBoolean[TRUE];
                RETURN;
                END;
        PutCharVM[c,stringob.Address,stringob.Offset,i];
```

```
        i←i+1;
END;
ENDLOOP;
END;

WrtString:PUBLIC PROCEDURE [stack: Stack]=
BEGIN
i:CARDINAL;
stringob: StringType Object ← DescStringType[Pop[stack]];
stream: StreamHandle ← DescStreamType[Pop[stack]].SHandle;
FOR i IN [0..stringob.Length)
DO
  stream.put[stream,GetCharVM[stringob.Address,stringob.Offset,i]];
ENDLOOP;
END;


WriteByteStream:PUBLIC PROCEDURE  =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
WrtString[frame.opstk];
END;



Print:PUBLIC PROCEDURE  =
BEGIN
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
StreamLit[GetOutputStream[],frame.opstk];
Exch[frame.opstk];
WrtString[frame.opstk];
END;

Run:PUBLIC PROCEDURE  =
BEGIN
ob: Object;
frame:Frame ← JaMControlDefs.GetCurrentFrame[];
JaMFnsDefs.PushInteger[1];
NByteStream[];
ob ← Pop[frame.opstk];
ob.litflag ← nolit;
Push[ob,frame.execstk];
END;


-- NByteStream, given the file name and options, returns a stream on the
-- stack.

NByteStream: PUBLIC PROCEDURE =
BEGIN
s:STRING ← [256];
frame:Frame ←JaMControlDefs.GetCurrentFrame[];
i:INTEGER ← JaMFnsDefs.PopInteger[];
ob:StringType Object ← DescStringType[Pop[frame.opstk]];
strm: StreamDefs.StreamHandle;
IF ob.Length > 256 THEN JaMExecDefs.JaMError[LongFileName,TRUE];
GetCharsVM[ob.Address,ob.Offset,@s.text,0,ob.Length];
s.length ← ob.Length;
strm ← EdStreamDefs.CreateBufferedStream[NewByteStream[s,i !
        SegmentDefs.FileNameError => JaMExecDefs.JaMError[BadFileName,TRUE]]];
Push[[lit,StreamType[strm]],frame.opstk];
END;


NKeyStream: PUBLIC PROCEDURE =
BEGIN OPEN JaMFnsDefs;
height: RectangleDefs.yCoord ← PopInteger[];
y0: RectangleDefs.yCoord ← PopInteger[];
width: RectangleDefs.xCoord ← PopInteger[];
x0: RectangleDefs.xCoord ← PopInteger[];
name: STRING ← [100];
strm: StreamHandle;
PopString[name ! IODefs.LineOverflow =>
                BEGIN
                name ← "anon";
                CONTINUE;
```

```
                END];
strm ← EdStreamDefs.CreateEditedStream[NIL, IODefs.ESC,
        name, BitmapHandle, x0, width, y0, height];
Push[[lit,StreamType[strm]],JaMControlDefs.GetCurrentFrame[].opstk];
END;


DestroyStream: PUBLIC PROCEDURE =
BEGIN
s:StreamHandle ← DescStreamType[Pop[JaMControlDefs.GetCurrentFrame[].opstk]].SHandle;
s.destroy[s];
END;

SetOutput: PUBLIC PROCEDURE =
BEGIN
IODefs.SetOutputStream[JaMFnsDefs.PopStream[]];
END;

GetOutput: PUBLIC PROCEDURE =
BEGIN
streamob: StreamType Object ← [lit,StreamType[IODefs.GetOutputStream[]]];
JaMFnsDefs.PushObject[streamob];
END;

        BitmapHeightDots: CARDINAL=350;
        BitmapWidthDots:        CARDINAL=512;
        BitmapWidthWords: CARDINAL =
                (BitmapWidthDots+15)/16+(((BitmapWidthDots+15)/16) MOD 2);
        BitmapAreaWords: CARDINAL = BitmapWidthWords*BitmapHeightDots;
        BitmapAreaPages: CARDINAL =
                (BitmapAreaWords+AltoDefs.PageSize-1)/AltoDefs.PageSize;
        Bitmap: RectangleDefs.BMptr;
        BitmapHandle: PUBLIC RectangleDefs.BMHandle ← NIL;

        SetUpScreen: PROCEDURE =        -- Set up centered bitmap display
        BEGIN OPEN RectangleDefs;
                W1: WindowDefs.WindowHandle;
                edstrm: StreamDefs.StreamHandle;
                DisplayDefs.DisplayOff[white];
                START RectanglesB[BitmapAreaPages,BitmapWidthWords];
                BitmapHandle ← GetDefaultBitmap[];
                BitmapHandle.indenting ← (38-BitmapWidthWords)/2;
                [] ← UpdateBitmap[BitmapHandle];
                Bitmap ← BitmapHandle.addr;

                START WindowDefs.WindowsB["JaM.typescript"];
                W1 ← WindowDefs.GetCurrentDisplayWindow[];
                edstrm ←
                        EdStreamDefs.CreateEditedStream[W1,IODefs.ESC,NIL,NIL,0,0,0,0];
                IODefs.SetOutputStream[edstrm];
                IODefs.SetInputStream[edstrm];
                WindowDefs.RepaintDisplayWindows[BitmapHandle];
        END;


--START Code--

LongFileName:StringType Object ← MakeStringObject[".longname"];
BadFileName:StringType Object ← MakeStringObject[".badname"];

SetUpScreen[];

RegisterCommand[".print"L,Print];
RegisterCommand[".readline"L,ReadLine];
RegisterCommand[".writebytes"L,WriteByteStream];
RegisterCommand[".bytestream"L,NByteStream];
RegisterCommand[".keystream"L,NKeyStream];
RegisterCommand[".killstream"L,DestroyStream];
RegisterCommand[".run"L,Run];
RegisterCommand[".setoutput"L,SetOutput];
RegisterCommand[".getoutput"L,GetOutput];

END.
```

```
--file JaMControl.mesa
--Written by John Warnock, Feb., 1979.
--Updated: March 19, 1979  9:07 AM by MN

DIRECTORY

SegmentDefs: FROM "SegmentDefs",
StringDefs: FROM "StringDefs",
IODefs: FROM "IODefs",
SystemDefs: FROM "SystemDefs",
FrameDefs: FROM "FrameDefs",
ControlDefs: FROM "ControlDefs",
ImageDefs: FROM "ImageDefs",
JaMMasterDefs: FROM "JaMMasterDefs",
JaMExecDefs: FROM "JaMExecDefs",
JaMFnsDefs: FROM "JaMFnsDefs",
JaMVMDefs: FROM "JaMVMDefs",
JaMControlDefs: FROM "JaMControlDefs",
JaMLiteralDefs: FROM "JaMLiteralDefs",
JaMDictionaryDefs: FROM "JaMDictionaryDefs",
JaMStackDefs: FROM "JaMStackDefs";

JaMControl: PROGRAM
IMPORTS SegmentDefs,StringDefs,IODefs,FrameDefs,JaMExecDefs,JaMFnsDefs,
JaMVMDefs,JaMLiteralDefs,JaMDictionaryDefs,JaMStackDefs
EXPORTS JaMControlDefs =
BEGIN
OPEN JaMMasterDefs,JaMVMDefs,JaMLiteralDefs,
JaMDictionaryDefs,JaMStackDefs;


-- This is the main control program for JaM. This program registers several
-- critical commands, creates an edited stream, and then transfers execution
-- control.

FrameStack: TYPE = ARRAY FrameStkPtr OF Frame;
FrameStkPtr: TYPE = [0..16);

frame: Frame;

GetCurrentFrame: PUBLIC PROCEDURE RETURNS [frm:Frame] =
BEGIN
RETURN[frame];
END;

RegisterCommand: PUBLIC PROCEDURE [stringname:STRING,procedure:PROCEDURE ]=
BEGIN
frame:Frame;
command:CommandType Object← [nolit,CommandType[procedure]];
frame ←GetCurrentFrame[];
StringLit[stringname,frame.opstk];
Push[command,frame.opstk];
DictDefine[];
END;

RegisterString: PUBLIC PROCEDURE [stringname:STRING,stringval:STRING]=
BEGIN
frame:Frame;
frame ←GetCurrentFrame[];
StringLit[stringname,frame.opstk];
Push[MakeStringObject[stringval],frame.opstk];
DictDefine[];
END;


RegisterBoolean: PUBLIC PROCEDURE [obname:STRING,b:BOOLEAN]=
BEGIN
frame:Frame;
frame ←GetCurrentFrame[];
StringLit[obname,frame.opstk];
BooleanLit[b,frame.opstk];
DictDefine[];
END;

RegisterObject: PUBLIC PROCEDURE [obname:STRING,ob:Object]=
BEGIN
```

```
frame:Frame;
frame ←GetCurrentFrame[];
StringLit[obname,frame.opstk];
Push[ob,frame.opstk];
DictDefine[];
END;

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

LoadBCD: PROCEDURE =
-- Expects opstk: (bcdFileName)
-- Loads and STARTS the configuration in bcdFileName
BEGIN
        s: STRING ← [256];
        frame: FrameDefs.GlobalFrameHandle;
        JaMFnsDefs.PopString[s ! IODefs.LineOverflow =>

**ngFileName,TRUE]];
        frame ← FrameDefs.New[s ! SegmentDefs.FileNameError =>


**ed

**[s,".bcd"];

** !

**eNameError =>

**Error[BadFileName,TRUE]];


        IF frame#ControlDefs.NullGlobalFrame THEN FrameDefs.Start[frame];
END;

LongFileName: StringType Object;  --must be initialized after starting VM
BadFileName: StringType Object;

OldVM: BOOLEAN;
SysDict:Object;
frameplace:INTEGER ←0;

-- Build the initial frame stack. (note: 16 entries).

framestack:FrameStack;

stackLinkArray: PRIVATE ARRAY [0..SIZE[Frame]*LENGTH[framestack]) OF StackLink;

i: CARDINAL;

FOR i IN [0..LENGTH[stackLinkArray]) DO
  stackLinkArray[i] ← NIL
  ENDLOOP;

FOR i IN [0..LENGTH[framestack]) DO
  framestack[i] ←
        [@stackLinkArray[SIZE[Frame]*i],
         @stackLinkArray[SIZE[Frame]*i+1],
         @stackLinkArray[SIZE[Frame]*i+2]]
  ENDLOOP;

frame ← framestack[frameplace];
OldVM ← RestartVM["JaM.VM",1000,20];

IF OldVM THEN
BEGIN
GetWordsVM[LOOPHOLE[LONG[0]],@SysDict,SIZE[Object]];
Push[SysDict,frame.dictstk];
END
ELSE
BEGIN
-- Build the System Dictionary (note: only 100 entries).
vma:LONG POINTER ←AllocateWordsVM[SIZE[Object]]; --This must be zero.
IntegerLit[256,frame.opstk];
DictDict[];
```

```
JaMExecDefs.JaMError[Lo

BEGIN
--maybe extension omitt

StringDefs.AppendString

frame ← FrameDefs.New[s

        SegmentDefs.Fil

        JaMExecDefs.JaM

CONTINUE;
END];
```

```
DictBegin[];
SysDict←Top[frame.dictstk];
PutWordsVM[vma,@SysDict,SIZE[Object]];
END;

LongFileName ← MakeStringObject[".longname"];
BadFileName ← MakeStringObject[".badname"];

-- Register the needed commands that the stack, and dictionary modules
-- could not register.

RegisterCommand[".quit"L,Quit];
--Stack commands
RegisterCommand[".pop"L,PopOpStk];
RegisterCommand[".exch"L,ExchOpStk];
RegisterCommand[".dup"L,DupOpStk];
RegisterCommand[".clrstk"L,ClearOpStk];
RegisterCommand[".copy"L,CopyOpStk];
RegisterCommand[".roll"L,RollOpStk];
RegisterCommand[".cntstk"L,CountOpStk];
RegisterCommand[".cnttomrk"L,CountToMrk];
RegisterCommand[".clrtomrk"L,ClearToMrk];
RegisterCommand[".mark"L,Mark];
--Dictionary commands
RegisterObject[".sysdict"L,SysDict];
RegisterCommand[".dict"L,DictDict];
RegisterCommand[".maxlength"L,DictMaxLength];
RegisterCommand[".known"L,DictKnown];
RegisterCommand[".where"L,DictWhere];
RegisterCommand[".get"L,DictGet];
RegisterCommand[".put"L,DictPut];
RegisterCommand[".def"L,DictDefine];
RegisterCommand[".load"L,DictLoad];
RegisterCommand[".store"L,DictStore];
RegisterCommand[".del"L,DictDelete];
RegisterCommand[".clrdict"L,DictClear];
RegisterCommand[".begin"L,DictBegin];
RegisterCommand[".end"L,DictEnd];
RegisterCommand[".dictforall"L,DictForall];
RegisterString[".undefkey"L," (Undefined key - .undefkey: ).print
               (                    ) .cvis .print"];
--RegisterString[".prompt"L,"(*) .print"];
--Booleans
RegisterBoolean[".true"L,TRUE];
RegisterBoolean[".false"L,FALSE];
--Loading a configuration
RegisterCommand[".loadbcd"L,LoadBCD];

-- All other commands are expected to register their own intrinsics.
END.
```

```
-- WPack.config
-- Last edited by Beau Sheil on April 23, 1979  6:04 PM

WPack: CONFIGURATION
  IMPORTS MiscDefs, SegmentDefs, StreamDefs, StringDefs, SystemDefs
  EXPORTS RectangleDefs, MenuDefs, WindowDefs, WStreamDefs =
    BEGIN
      Fonts;  BitMaps;  Rectangles;     -- RectangleDefs
      Menus;                            -- MenuDefs
      Windows;  Selections;             -- WindowDefs
      WStreams;                         -- WStreamDefs
    END.
```

Mesa 5.0 version
of the Mesa 4 WindowPacks.
by Beau Sheil.
[Ivy]<Sheil>WPack.sources

```
-- BitMaps.Mesa
-- Last edited by Beau Sheil on April 22, 1979  1:30 PM

DIRECTORY
  AltoDefs:       FROM "AltoDefs",
  InlineDefs:     FROM "InlineDefs",
  MiscDefs:       FROM "MiscDefs",
  RectangleDefs: FROM "rectangledefs",
  SystemDefs:     FROM "SystemDefs";

BitMaps: PROGRAM
  IMPORTS InlineDefs, MiscDefs, RectangleDefs, SystemDefs
  EXPORTS RectangleDefs =

  BEGIN OPEN RectangleDefs;

-- Display Declarations

  DCBchainHead: DCBptr = LOOPHOLE[420B];
  DCBnil: DCBptr = LOOPHOLE[0];
  offDCB, savedDCB: DCBptr ← DCBnil;

-- Bitmap Declarations

  bitmaps: PUBLIC BMHandle ← NIL;
  defaultmapdata: PUBLIC BMHandle ← NIL;

-- Display Control Routines

  DisplayOn: PUBLIC PROCEDURE =
    BEGIN
    mapdata: BMHandle;
    FOR mapdata ← bitmaps, mapdata.link UNTIL mapdata = NIL DO
      ReallocateBitmap[mapdata, mapdata.words/256, mapdata.wordsperline];
      -- NOTE: this code relies on the fields "words" and
      -- "wordsperline" in the BitmapObject being valid
      ENDLOOP;
    LoadDefaultFont[];
    DCBchainHead.next ← savedDCB;
    END;

  DisplayOff: PUBLIC PROCEDURE [background: backgtype] =
    BEGIN
    mapdata: BMHandle;
    savedDCB ← DCBchainHead.next;
    offDCB.background ← background;
    offDCB.next ← DCBnil;       -- this must be the end of the line
    DCBchainHead.next ← offDCB;
    -- NOTE: Turn ON code relies on the fields "words" and
    -- "wordsperline" in the BitmapObject being valid
    FOR mapdata ← bitmaps, mapdata.link UNTIL mapdata = NIL
      DO ReallocateBitmap[mapdata, 0, 0];
         ENDLOOP;
    UnLoadDefaultFont[];
    END;

  EVEN: PUBLIC PROCEDURE[v: UNSPECIFIED] RETURNS [UNSPECIFIED] =
    BEGIN
    -- make an even value by rounding v up. Used to align DCBs.
    RETURN[v+InlineDefs.BITAND[v, 1]];
    END;

  MakeDCB: PROCEDURE [h: CARDINAL ← 0] RETURNS [dcb: DCBptr] =
    BEGIN
      dcb ← EVEN[SystemDefs.AllocateHeapNode[SIZE[DCB]+1]];
      MiscDefs.Zero[dcb, SIZE[DCB]];  -- Sets link to DCBnil
      dcb.height ← h;
      dcb.background ← white;
      dcb.resolution ← high;
    END;

-- Bitmap Routines

  BitmapError: PUBLIC SIGNAL [bitmap: BMHandle, error: BitmapErrorCode] = CODE;

  CreateBitmap: PUBLIC PROCEDURE [pagesformap, wordsperline: CARDINAL]
```

```
            RETURNS[BMHandle] =
  BEGIN
  mapdata: BMHandle ← SystemDefs.AllocateHeapNode[SIZE[BitmapObject]];
  mapdata↑ ← BitmapObject[NIL, NIL, MakeDCB[], NIL, 0, 0, 0, 0, 0, 0,
                          3, high, white];
  ReallocateBitmap[mapdata, pagesformap, wordsperline];
  mapdata.link ← bitmaps;
  bitmaps ← mapdata;
  RETURN[mapdata];
  END;

CursorToMapCoords: PUBLIC PROCEDURE [mapdata: BMHandle, x: xCoord, y: yCoord]
                   RETURNS [mapx: xCoord, mapy: yCoord] =
  BEGIN
  -- NOTE!! if bitmap ptr not supplied then use system default...
  IF mapdata = NIL THEN mapdata ← defaultmapdata;
  mapx ← MAX[0, MIN[mapdata.width, INTEGER[x - mapdata.x0]]];
  mapy ← MAX[0, MIN[mapdata.height, INTEGER[y - mapdata.y0]]];
  RETURN[mapx, mapy]
  END;

DestroyBitmap: PUBLIC PROCEDURE [mapdata: BMHandle] RETURNS [POINTER] =
  BEGIN
  IF mapdata.rectangles # NIL THEN
    SIGNAL BitmapError[mapdata, BitmapOperation];
  IF mapdata.addr # NIL THEN SystemDefs.FreePages[mapdata.addr];
  IF mapdata.dcb # NIL THEN SystemDefs.FreeHeapNode[mapdata.dcb];
  IF mapdata = bitmaps
    THEN bitmaps ← mapdata.link
    ELSE
      BEGIN
      prev: BMHandle;
      FOR prev ← bitmaps, prev.link UNTIL mapdata = prev.link DO ENDLOOP;
      prev.link ← mapdata.link;
      END;
  SystemDefs.FreeHeapNode[mapdata];
  RETURN[NIL];  -- At one time, returned the address of the freed pages!
  END;

DisplayBitmap: PUBLIC PROCEDURE [mapdata: BMHandle] =
  BEGIN
  -- links a bitmap into the displaychain using a BitmapObject
  -- Assumes Bitmap Record is correct, eg: map is even word alligned etc...
  -- fills in the x0,y0 fields in the bitmap record too!
  dcb, nextdcb: DCBptr;
  dcb ← UpdateBitmap[mapdata];
  nextdcb ← DCBchainHead.next;
  mapdata.y0 ← 0;
  IF nextdcb # DCBnil THEN
    BEGIN
    WHILE nextdcb.next # DCBnil DO
      mapdata.y0 ← mapdata.y0 + (nextdcb.height)*2;
      nextdcb ← nextdcb.next;
      ENDLOOP;
    mapdata.y0 ← mapdata.y0 + (nextdcb.height)*2;
    END;
  nextdcb.next ← dcb;
  mapdata.x0 ← mapdata.indenting*16;
  END;

GetDefaultBitmap: PUBLIC PROCEDURE RETURNS [BMHandle] =
  BEGIN
  IF defaultmapdata=NIL THEN defaultmapdata ← CreateBitmap[40, 30];
  RETURN[defaultmapdata];
  END;

ReallocateBitmap: PUBLIC PROCEDURE [
  mapdata: BMHandle, pagesformap, wordsperline: CARDINAL] =
  BEGIN
  -- physically alters a display bitmap
  map: BMptr ← mapdata.addr;
  rectangle: Rptr;
  wordsformap: CARDINAL = pagesformap*AltoDefs.PageSize;
  IF map # NIL AND wordsformap # mapdata.words THEN
    BEGIN
    mapdata.dcb.width ← 0; -- ensure no trash on screen
```

```
      SystemDefs.FreePages[mapdata.addr];
      map ← NIL;
      END;
    IF pagesformap # 0 THEN
      BEGIN
      -- NOTE: assumes pages allocated on EVEN word boundary
      IF map = NIL THEN map ← SystemDefs.AllocateResidentPages[pagesformap];
      MiscDefs.Zero[map, wordsformap];
      mapdata.addr ← map;
      mapdata.words ← wordsformap;
      mapdata.wordsperline ← wordsperline;
      mapdata.width ← wordsperline*16;
      mapdata.height ← wordsformap/wordsperline;
      IF InlineDefs.BITAND[mapdata.height, 1] = 1 THEN
        mapdata.height ← mapdata.height-1;
      [] ← UpdateBitmap[mapdata];
      END
    ELSE
      BEGIN
      mapdata.addr ← NIL;
      mapdata.width ← 0;
      mapdata.height ← 0;
      END;
    FOR rectangle ← mapdata.rectangles, rectangle.link UNTIL rectangle = NIL
        DO FixupRectangle[rectangle]; ENDLOOP;
    END;

  UpdateBitmap: PUBLIC PROCEDURE [mapdata: BMHandle] RETURNS [DCBptr] =
    BEGIN
    dcb: DCBptr = EVEN[mapdata.dcb];
    dcb.bitmap ← mapdata.addr;
    dcb.height ← mapdata.height/2;
    dcb.width ← mapdata.wordsperline;
    dcb.indenting  ← mapdata.indenting;
    dcb.resolution  ← mapdata.resolution;
    dcb.background  ← mapdata.background;
    RETURN[dcb];
    END;

  UnDisplayBitmap: PUBLIC PROCEDURE[mapdata: BMHandle] =
    BEGIN
    -- nop for now
    END;


-- MAIN BODY CODE

    -- Define the DisplayOff DCB
    offDCB ← MakeDCB[];

    IF DCBchainHead.next = DCBnil      -- dummy dcb must never be deallocated
       THEN DCBchainHead.next ← MakeDCB[topmargin];

  END. of BitMaps
```

```
-- Fonts.Mesa
-- Last edited by Beau Sheil on April 22, 1979  2:04 PM

DIRECTORY
  IODefs: FROM "iodefs" USING [CR, DEL, SP],
  RectangleDefs: FROM "rectangledefs",
  SegmentDefs: FROM "segmentdefs";

Fonts: PROGRAM
  IMPORTS RectangleDefs, SegmentDefs
  EXPORTS RectangleDefs =

  BEGIN OPEN RectangleDefs;

-- Local Data

  FileSegmentHandle:  TYPE = SegmentDefs.FileSegmentHandle;
  SevenBitCharacter:  TYPE = CHARACTER[0C..177C];

  defaultcharwidths:  PACKED ARRAY SevenBitCharacter OF [0..256);
  defaultfont:        Fptr ← NIL;
  defaultfontsegment: FileSegmentHandle ← NIL;

-- Font routines

  ComputeCharWidth: PUBLIC PROCEDURE [c: CHARACTER, font: FAptr]
                    RETURNS [CARDINAL] =
    BEGIN
    IF c = IODefs.CR THEN c ← IODefs.SP;
    IF c < IODefs.SP THEN
     RETURN[ComputeCharWidth['↑, font] +
            ComputeCharWidth[LOOPHOLE[LOOPHOLE[c,INTEGER]+100B,CHARACTER],
                             font]];
    IF font = @defaultfont.FCDptrs AND c <= IODefs.DEL
       THEN RETURN[defaultcharwidths[c]]
       ELSE -- compute the width of this character directly from font
            BEGIN
            code: CARDINAL ← LOOPHOLE[c];  w: CARDINAL ← 0;
            cw: FCDptr;
            fontdesc: DESCRIPTOR FOR ARRAY OF FCDptr ← DESCRIPTOR[font, 256];
            DO cw ← LOOPHOLE[fontdesc[code]+LOOPHOLE[font]+code, FCDptr];
               IF cw.HasNoExtension THEN RETURN [w + cw.widthORext];
               w ← w+16;
               code ← LOOPHOLE[cw.widthORext];
               ENDLOOP;
            END;
    END;

  ComputeStringWidth: PUBLIC PROCEDURE[s: STRING, font: FAptr]
                      RETURNS[w: CARDINAL] =
    BEGIN
    i: CARDINAL;
    w ← 0;
    FOR i IN [0..s.length) DO w ← w+ComputeCharWidth[s[i],font] ENDLOOP;
    END;

  GetDefaultFont: PUBLIC PROCEDURE RETURNS [FAptr, CARDINAL] =
    BEGIN
    RETURN[@defaultfont.FCDptrs, defaultfont.FHeader.MaxHeight];
    END;

  GetFontSegment: PUBLIC PROCEDURE [filename: STRING] RETURNS [FileSegmentHandle] =
    BEGIN OPEN SegmentDefs;
    RETURN[NewFileSegment[NewFile[filename, Read, OldFileOnly], DefaultBase,
                          DefaultPages, Read]];
    END;

  LoadFont: PUBLIC PROCEDURE [segment: FileSegmentHandle] RETURNS [Fptr] =
    BEGIN OPEN SegmentDefs;
    SwapIn[segment];
    RETURN[FileSegmentAddress[segment]];
    END;

  LoadDefaultFont: PUBLIC PROCEDURE =
    BEGIN
    defaultfont ← LoadFont[defaultfontsegment];
```

```
    END;

  UnLoadDefaultFont: PUBLIC PROCEDURE =
    BEGIN
    SegmentDefs.Unlock[defaultfontsegment];
    END;

  SetUpDefaultFont: PROCEDURE [dfont: Fptr]=
    BEGIN
    c: SevenBitCharacter;
    pfont: FAptr ← @dfont.FCDptrs;
    FOR c IN SevenBitCharacter DO
      defaultcharwidths[c] ← ComputeCharWidth[c, pfont];
      -- ComputeCharWidth computes new widths iff pfont not from defaultfont
      ENDLOOP;
    -- now char widths are set, save font in global variable
    defaultfont ← dfont;
    END;


-- MAIN BODY CODE

    defaultfontsegment ← GetFontSegment["MesaFont.A1."
         ! SegmentDefs.FileNameError, SegmentDefs.FileError =>
                BEGIN
                defaultfontsegment ← GetFontSegment["SysFont.A1."];
                CONTINUE;
                END
                                        ];
    SetUpDefaultFont[LoadFont[defaultfontsegment]];

  END. of Fonts
```

```
-- Menudefs.mesa  Last edited by Beau Sheil on April 16, 1979  2:40 PM

DIRECTORY
  StreamDefs:  FROM "streamdefs",
  SegmentDefs: FROM "segmentdefs",
  RectangleDefs: FROM "rectangledefs";

MenuDefs: DEFINITIONS =

  BEGIN OPEN StreamDefs, RectangleDefs;

  MenuLeftMargin: CARDINAL = 5;

  MenuHandle: TYPE = POINTER TO MenuObject;

  MenuObject: TYPE = RECORD [
    link: MenuHandle,
    index: INTEGER,
    width: xCoord,
    rectangle: Rptr,
    menuseg, dataseg: SegmentDefs.FileSegmentHandle,
    array: MenuArray];

  MenuArray: TYPE = DESCRIPTOR FOR ARRAY OF MenuItem;

  MenuItem: TYPE = RECORD [
    keyword: STRING,
    proc: PROCEDURE [UNSPECIFIED, xCoord, yCoord]];


  -- Procedures Implementing Menus

  CreateMenu: PROCEDURE [array: MenuArray] RETURNS [menu: MenuHandle];
  DestroyMenu: PROCEDURE [menu: MenuHandle];
  DisplayMenu: PROCEDURE
    [menu: MenuHandle, mapdata: BMHandle, x: xCoord, y: yCoord];
  ClearMenu: PROCEDURE [menu: MenuHandle];
  MarkMenuItem: PROCEDURE [menu: MenuHandle, index: INTEGER];

  END. of MenuDefs
```

```
-- Menus.mesa
-- Last edited by Beau Sheil on April 22, 1979  5:36 PM

DIRECTORY
  MenuDefs: FROM "menudefs",
  RectangleDefs: FROM "rectangledefs",
  SystemDefs: FROM "systemdefs" USING [AllocateHeapNode, FreeHeapNode];

Menus: PROGRAM
  IMPORTS RectangleDefs, SystemDefs
  EXPORTS MenuDefs =

  BEGIN OPEN MenuDefs, RectangleDefs;

-- Display Menu Routines

  CreateMenu: PUBLIC PROCEDURE [array: MenuArray] RETURNS [menu: MenuHandle] =
    BEGIN
    width: xCoord ← 0; i: CARDINAL;
    -- compute width of widest command word
    FOR i IN [0..LENGTH[array]) DO
      width ← MAX[width, ComputeStringWidth[array[i].keyword, defaultpfont]];
      ENDLOOP;
    -- now create and initialize menu object
    menu ← SystemDefs.AllocateHeapNode[SIZE[MenuObject]];
    menu↑ ← MenuObject[NIL, -1, width+MenuLeftMargin*2, NIL, NIL, NIL, array];
    END;

  DestroyMenu: PUBLIC PROCEDURE [menu: MenuHandle] =
    BEGIN
    SystemDefs.FreeHeapNode[menu];      -- Should we unlink from menu list?
    END;

  DisplayMenu: PUBLIC PROCEDURE [menu: MenuHandle, mapdata: BMHandle,
                                 x: xCoord, y: yCoord] =
    BEGIN
    i: CARDINAL;
    cx: INTEGER ← MAX[0, INTEGER[x-(menu.width+2)]];
    cy: INTEGER;
    nitems: CARDINAL ← 0;
    length: CARDINAL = LENGTH[menu.array];
    -- save data first then clear it
    IF menu.index # -1 THEN nitems ← menu.index;
    nitems ← MAX[length/2, nitems];
    cy ← y-(nitems*defaultlineheight+(defaultlineheight/2));
    cy ← MAX[0, cy];
    menu.rectangle ← CreateRectangle[mapdata, cx, menu.width, cy,
                                     defaultlineheight*length+2];
    menu.rectangle.options.NoteOverflow ← TRUE;
    --test page size of menu before saving
    IF (((menu.rectangle.cw + 15) /16 +1) * menu.rectangle.ch ) / 256 <= 4
      THEN menu.dataseg ← LOOPHOLE[SaveRectangle[menu.rectangle]];
    menu.index ← -1;
    ClearBoxInRectangle[menu.rectangle, 0, menu.rectangle.cw, 0,
                        menu.rectangle.ch];
    -- now paint it up on screen
    DrawBoxInRectangle[
      menu.rectangle, 0, menu.rectangle.cw, 0, menu.rectangle.ch];
    FOR i IN [0..length) DO
      [cx,cy] ← WriteRectangleString[menu.rectangle, MenuLeftMargin,
  (i*defaultlineheight + 1), menu.array[i].keyword, defaultpfont
        ! RectangleError =>
          SELECT error FROM
            RightOverflow => CONTINUE;
            NotVisible , BottomOverflow => EXIT;
            ENDCASE];
      ENDLOOP;
    END;

  ClearMenu: PUBLIC PROCEDURE [menu: MenuHandle] =
    BEGIN
    IF menu.rectangle # NIL THEN
      BEGIN
      ClearBoxInRectangle[menu.rectangle, 0, menu.rectangle.cw, 0,
                          menu.rectangle.ch];
      IF menu.dataseg # NIL THEN
```

```
        BEGIN
        RestoreRectangle[menu.rectangle, LOOPHOLE[menu.dataseg]];
        menu.dataseg ← NIL;
        END;
      DestroyRectangle[menu.rectangle];
      menu.rectangle ← NIL;
      END;
    END;

  MarkMenuItem: PUBLIC PROCEDURE [menu: MenuHandle, index: INTEGER] =
    BEGIN
    i:  CARDINAL ← 1;
    oldIndex: CARDINAL ← menu.index;
    r: Rptr = menu.rectangle;
    IF menu.index = index THEN RETURN;
    IF menu.index = LENGTH[menu.array]-1 THEN i ← 0;
    IF menu.index # -1 THEN -- turn old guy off
      InvertBoxInRectangle[
  r, 1, r.cw-2, oldIndex*defaultlineheight+1, defaultlineheight-i];
    i ← IF index = LENGTH[menu.array]-1 THEN 0 ELSE 1;
    oldIndex ← index;
    IF index # -1 THEN -- turn new guy on
      InvertBoxInRectangle[
  r, 1, r.cw-2, oldIndex*defaultlineheight+1, defaultlineheight-i];
    menu.index ← index;
    END;

 -- Set up defaultfont for menus oin first entry

 defaultpfont: FAptr;
 defaultlineheight: CARDINAL;
 [defaultpfont, defaultlineheight] ← GetDefaultFont[];

END. of Menus
```

```
-- Rectangledefs.mesa
-- Last edited by Beau Sheil on April 22, 1979  2:07 PM

DIRECTORY
  SegmentDefs: FROM "segmentdefs";

RectangleDefs: DEFINITIONS =
  BEGIN

-- Magic constants

  mousebuttonsup: CARDINAL = 7;

-- Constants for Display stuff

  leftmargin: CARDINAL = 10;
  topmargin: CARDINAL = 64; -- ~5 blank lines at the top
  -- not used: blanklines: CARDINAL = 6; = # of blank lines at the top
  maxwordsperline: CARDINAL = 38;
  maxbitsperline: CARDINAL = maxwordsperline*16;
  minwidth: CARDINAL = 32;
  minheight: CARDINAL = 32;

-- some TYPEs

  restype: TYPE = {high, low};
  backgtype: TYPE = {white, black};
  xCoord: TYPE = [0..606];
  yCoord: TYPE = [0..808];

-- Font Declarations

  FHptr: TYPE = POINTER TO FontHeader;
  Fptr: TYPE = POINTER TO FONT;
  FCDptr: TYPE = POINTER TO FCD;
  FAptr: TYPE = POINTER TO fontarray;
  fontarray: TYPE = ARRAY [0..255] OF FCDptr;

  FONT: TYPE = MACHINE DEPENDENT RECORD [
    FHeader: FontHeader,
    FCDptrs: fontarray,  -- array of self-relative pointers to
                         -- FCD's.  Indexed by char value.
                         -- font pointer points hear!
    ExtFCDptrs: fontarray];  -- array of self-relative pointers to
                             -- FCD's for extentions.  As large an
                             -- array as needed.

  FontHeader: TYPE = MACHINE DEPENDENT RECORD [
    MaxHeight: CARDINAL,         -- height of tallest char in font (scan lines)
    VariableWidth: [0..1], -- IF TRUE, proportionally spaced font
    blank:   [0..177B],          -- not used
    MaxWidth: [0..377B]];  -- width of widest char in font (raster units).

  FCD: TYPE = MACHINE DEPENDENT RECORD [
    widthORext: [0..77777B],  -- width or extention index
    HasNoExtension: BOOLEAN,  -- TRUE=> no ext.;prevfield=width
    height: [0..377B],      -- # scan lines to skip for char
    displacement: [0..377B]]; -- displacement back to char bitmap

-- Display Declarations

  DCBptr: TYPE = POINTER TO DCB;
  DCB: TYPE = MACHINE DEPENDENT RECORD [
    next: DCBptr,
    resolution: restype,
    background: backgtype,
    indenting: [0..77B],   -- in units of 16 bits
    width: [0..377B],   -- likewise; must be even
    bitmap: BMptr,      -- must be even
    height: CARDINAL];  -- in double scan lines

-- Bit Map Declarations

  BitmapError: SIGNAL [bitmap: BMHandle, error: BitmapErrorCode];
  BitmapErrorCode: TYPE = {BitmapOperation};
  BITMAP: TYPE = WORD; -- array [0..15] of word
```

```
  BMptr: TYPE = POINTER TO BITMAP;
  BMHandle: TYPE = POINTER TO BitmapObject;

  BitmapObject: TYPE = RECORD [ -- all about a Bitmap
    link: BMHandle,  -- # NIL iff being displayed
    rectangles: Rptr, -- list of display streams for this map
    dcb: DCBptr,
    addr: BMptr,  -- it's address
    words: CARDINAL, -- size of map (in words)
    wordsperline: [0..maxwordsperline],
    x0: xCoord,       -- x,y of upper left corner
    y0: yCoord,
    width: xCoord,
    height: yCoord,
    indenting: [0..77B],    -- in units of 16 bits
    resolution: restype,
    background: backgtype];

-- Rectangle Declarations

  Rptr: TYPE = POINTER TO Rectangle;
  Rectangle: TYPE = RECORD [
    link: Rptr,
    visible: BOOLEAN,
    options: ROptions,
    bitmap: BMHandle,
    x0, width, cw: xCoord,
    y0, height, ch: yCoord];

  -- Rectangle options field definitions

    ROptions: TYPE = RECORD [ -- Rectangle options
      NoteInvisible: BOOLEAN, -- SIGNAL if rectangle off bitmap
      NoteOverflow: BOOLEAN]; -- SIGNAL if storing outside

  RectangleError: SIGNAL [rectangle:Rptr, error:RectangleErrorCode];
  RectangleErrorCode: TYPE = {RightOverflow, BottomOverflow, NotVisible};

  GrayArray: TYPE = ARRAY [0..3] OF WORD;
  GrayPtr: TYPE = POINTER TO GrayArray;

-- Procedures Implementing RECTANGLEDEFS

  -- Font Routines

  ComputeCharWidth: PROCEDURE [c: CHARACTER, font: FAptr] RETURNS [CARDINAL];
  ComputeStringWidth: PROCEDURE[s: STRING, font: FAptr] RETURNS[CARDINAL];
  GetDefaultFont: PROCEDURE RETURNS [FAptr, CARDINAL];
  GetFontSegment: PROCEDURE [filename: STRING]
                  RETURNS [SegmentDefs.FileSegmentHandle];
  LoadFont: PROCEDURE [SegmentDefs.FileSegmentHandle] RETURNS [Fptr];
  LoadDefaultFont: PROCEDURE;
  UnLoadDefaultFont: PROCEDURE;

  -- Display Routines

  DisplayOff: PROCEDURE [backgtype];
  DisplayOn: PROCEDURE;
  EVEN: PROCEDURE [UNSPECIFIED] RETURNS [UNSPECIFIED];

  -- Bitmap Routines

  CreateBitmap: PROCEDURE [pagesformap, wordsperline: CARDINAL]
                RETURNS [BMHandle];
  CursorToMapCoords: PROCEDURE [BMHandle, xCoord, yCoord]
                     RETURNS [xCoord, yCoord];
  DestroyBitmap: PROCEDURE [BMHandle] RETURNS [POINTER];
  GetDefaultBitmap: PROCEDURE RETURNS [BMHandle];
  DisplayBitmap: PROCEDURE [BMHandle];
  ReallocateBitmap: PROCEDURE
    [mapdata: BMHandle, pagesformap, wordsperline: CARDINAL];
  UnDisplayBitmap: PROCEDURE [BMHandle];
  UpdateBitmap: PROCEDURE [BMHandle] RETURNS [DCBptr];

  -- Rectangle Routines
```

```
CreateRectangle: PROCEDURE [bitmap: BMHandle, x0, width: xCoord,
                            y0, height: yCoord] RETURNS[Rptr];
CursorToRecCoords: PROCEDURE [Rptr, xCoord, yCoord]
                   RETURNS [xCoord, yCoord];
DestroyRectangle: PROCEDURE[Rptr];
ClearBoxInRectangle: PROCEDURE [rectangle: Rptr, x0, width: xCoord,
                                y0, height: yCoord, gp: GrayPtr ← NIL];
DrawBoxInRectangle: PROCEDURE [rectangle: Rptr, x0, width: xCoord,
                               y0, height: yCoord];
FixupRectangle: PROCEDURE[Rptr];
GrowRectangle: PROCEDURE [rectangle: Rptr, width: xCoord, height: yCoord];
InvertBoxInRectangle: PROCEDURE [rectangle: Rptr, x0, width: xCoord,
                                 y0, height: yCoord];
IsRectangleVisible: PROCEDURE [Rptr] RETURNS [BOOLEAN];
MoveRectangle: PROCEDURE [Rptr, xCoord, yCoord];
RectangleToMapCoords: PROCEDURE [Rptr, xCoord, yCoord]
                      RETURNS [xCoord, yCoord];
RestoreRectangle:PROCEDURE [rectangle: Rptr, SegPtr: POINTER];
SaveRectangle:PROCEDURE [Rptr] RETURNS [POINTER];
ScrollBoxInRectangle: PROCEDURE [rectangle: Rptr, x0, width: xCoord,
                                 y0, height: yCoord, incr: INTEGER];
WriteRectangleChar: PROCEDURE [Rptr, xCoord, yCoord, CHARACTER, FAptr]
                    RETURNS[xCoord, yCoord];
WriteRectangleString: PROCEDURE [Rptr, xCoord, yCoord, STRING, FAptr]
                      RETURNS[xCoord, yCoord];

END. of RectangleDefs
```

```
-- Rectangles.Mesa
-- Last edited by Beau Sheil on April 22, 1979  4:18 PM

DIRECTORY
  AltoDefs: FROM "altodefs",
  BitBltDefs: FROM "bitbltdefs" USING [BBptr, BBTable, BITBLT],
  RectangleDefs: FROM "rectangledefs",
  SystemDefs: FROM "systemdefs";

Rectangles: PROGRAM
  IMPORTS BitBltDefs, RectangleDefs, SystemDefs
  EXPORTS RectangleDefs =

  BEGIN OPEN RectangleDefs;

-- Local Symbols

  Black: CARDINAL = 177777B;  -- to give as a gray word value

-- Rectangle Routines

  CreateRectangle: PUBLIC PROCEDURE [
    bitmap: BMHandle, x0, width: xCoord, y0, height: yCoord] RETURNS[Rptr] =
    BEGIN
    rectangle: Rptr;
    rectangle ← SystemDefs.AllocateHeapNode[SIZE[Rectangle]];
    rectangle↑ ← Rectangle[NIL, FALSE,, bitmap, x0, width, 0, y0, height, 0];
    rectangle.options ← ROptions[FALSE,FALSE];
    rectangle.link ← bitmap.rectangles;
    bitmap.rectangles ← rectangle;
    FixupRectangle[rectangle];
    RETURN[rectangle];
    END;

  ClearBoxInRectangle: PUBLIC PROCEDURE [
    rectangle: Rptr, x0, width: xCoord, y0, height: yCoord, gp: GrayPtr ← NIL] =
    BEGIN
    bbtable: ARRAY [0..SIZE[BitBltDefs.BBTable]] OF WORD;
    bbptr: BitBltDefs.BBptr = EVEN[BASE[bbtable]];
    mapaddr: BMptr ← rectangle.bitmap.addr;
    wordsperline: CARDINAL = rectangle.bitmap.wordsperline;
    dlx: xCoord ← rectangle.x0+x0;
    dty: yCoord ← rectangle.y0+y0;
    dw: xCoord ← MIN[rectangle.cw, width];
    dh: yCoord ← MIN[rectangle.ch, height];
    gray: GrayArray ← IF gp=NIL THEN GrayArray[0, 0, 0, 0] ELSE gp↑;
    bbptr↑ ← [0, FALSE, FALSE, gray, replace, 0, mapaddr, wordsperline,
      dlx, dty, dw, dh, mapaddr, wordsperline, dlx, dty, gray[0],
      gray[1], gray[2], gray[3]];
    BitBltDefs.BITBLT[bbptr];
    END;

  CursorToRecCoords: PUBLIC PROCEDURE [rectangle: Rptr, x: xCoord, y: yCoord]
    RETURNS[rx: xCoord, ry: yCoord] =
    BEGIN
    rx ← x - (rectangle.x0 + rectangle.bitmap.x0);
    ry ← y - (rectangle.y0 + rectangle.bitmap.y0);
    END;

  DestroyRectangle: PUBLIC PROCEDURE [rectangle: Rptr] =
    BEGIN
    bitmap: BMHandle ← rectangle.bitmap;
    IF bitmap.rectangles = rectangle THEN bitmap.rectangles ← rectangle.link
                                     ELSE
        BEGIN
        prev: Rptr;
        FOR prev ← bitmap.rectangles, prev.link UNTIL rectangle = prev.link
            DO IF prev = NIL THEN ERROR; ENDLOOP;
        prev.link ← rectangle.link;
        END;
    SystemDefs.FreeHeapNode[rectangle];
    END;

  DrawBoxInRectangle: PUBLIC PROCEDURE [
    rectangle: Rptr, x0, width: xCoord, y0, height: yCoord] =
    BEGIN
```

```
    bbtable: ARRAY [0..SIZE[BitBltDefs.BBTable]] OF WORD;
    bbptr: BitBltDefs.BBptr = EVEN[BASE[bbtable]];
    mapaddr: BMptr ← rectangle.bitmap.addr;
    wordsperline: INTEGER = rectangle.bitmap.wordsperline;
    dlx: xCoord ← rectangle.x0+x0;
    dty: yCoord ← rectangle.y0+y0;
    dw: xCoord ← MIN[rectangle.cw, width];
    dh: yCoord ← MIN[rectangle.ch, height];
    bbptr↑ ← [0, FALSE, FALSE, gray, replace, 0, mapaddr, wordsperline, dlx,
      dty, dw, 1, mapaddr, wordsperline, dlx, dty, Black, Black, Black, Black];
    BitBltDefs.BITBLT[bbptr];
    bbptr↑ ← [,,, gray,,,,,,, 1, dh,,,,,,,,];
    BitBltDefs.BITBLT[bbptr];
    bbptr↑ ← [,,,,,,,, dlx+dw-1, dty, 1, dh,,,,,,,,];
    BitBltDefs.BITBLT[bbptr];
    bbptr↑ ← [,,,,,,,, dlx, dty+dh-1, dw, 1,,,,,,,,];
    BitBltDefs.BITBLT[bbptr];
    END;

  FixupRectangle: PUBLIC PROCEDURE [r: Rptr] =
    BEGIN                              -- clips rectangle for visibility
    bm: BMHandle = r.bitmap;
    IF bm = NIL OR bm.addr = NIL
       THEN r.visible ← FALSE
       ELSE BEGIN
            r.cw ← MIN[IF bm.width > r.x0 THEN bm.width-r.x0 ELSE 0,
                       r.width];
            r.ch ← MIN[IF bm.height > r.y0 THEN bm.height-r.y0 ELSE 0,
                       r.height];
            r.visible ← (r.x0+minwidth <= bm.width AND
                         r.y0+minheight <= bm.height);
            END;
    END;

  GrowRectangle: PUBLIC PROCEDURE [rectangle: Rptr, width: xCoord, height: yCoord] =
    BEGIN
    mapaddr: BMptr = rectangle.bitmap.addr;
    wordsperline: CARDINAL = rectangle.bitmap.wordsperline;
    graywords: GrayArray ← [125252B, 52525B, 125252B, 52525B];
    gray: GrayPtr = @graywords;
    IF width = rectangle.width AND height = rectangle.height THEN RETURN;
    ClearBoxInRectangle[rectangle, 0, rectangle.cw, 0, rectangle.ch];
    rectangle.width ← width;
    rectangle.height ← height;
    FixupRectangle[rectangle];
    ClearBoxInRectangle[rectangle, 0, rectangle.cw, 0, rectangle.ch, gray];
    END;

  InvertBoxInRectangle: PUBLIC PROCEDURE [
    rectangle: Rptr, x0, width: xCoord, y0, height: yCoord] =
    BEGIN
    bbtable: ARRAY [0..SIZE[BitBltDefs.BBTable]] OF WORD;
    bbptr: BitBltDefs.BBptr = EVEN[BASE[bbtable]];
    mapaddr: BMptr ← rectangle.bitmap.addr;
    wordsperline: CARDINAL = rectangle.bitmap.wordsperline;
    dlx: xCoord ← rectangle.x0+x0;
    dty: yCoord ← rectangle.y0+y0;
    dw: xCoord ← MIN[rectangle.cw, width];
    dh: yCoord ← MIN[rectangle.ch, height];
    bbptr↑ ← [0, FALSE, FALSE, complement, replace, 0, mapaddr, wordsperline,
      dlx, dty, dw, dh, mapaddr, wordsperline, dlx, dty,,,,];
    BitBltDefs.BITBLT[bbptr];
    END;

  MoveRectangle: PUBLIC PROCEDURE [rectangle: Rptr, x: xCoord, y: yCoord] =
    BEGIN
    bbtable: ARRAY [0..SIZE[BitBltDefs.BBTable]] OF WORD;
    bbptr: BitBltDefs.BBptr = EVEN[BASE[bbtable]];
    oldx: xCoord = rectangle.x0;
    oldw: xCoord = rectangle.cw;
    oldy: yCoord = rectangle.y0;
    oldh: yCoord = rectangle.ch;
    mapaddr: BMptr = rectangle.bitmap.addr;
    wordsperline: CARDINAL = rectangle.bitmap.wordsperline;
    dlx, dw: xCoord;
    dty, dh: yCoord;
```

```
    IF x = oldx AND y = oldy THEN RETURN;
    rectangle.x0 ← x;
    rectangle.y0 ← y;
    FixupRectangle[rectangle];
    IF rectangle.visible = FALSE THEN RETURN;
    dw ← MIN[rectangle.cw, oldw];
    dh ← MIN[rectangle.ch, oldh];
    bbptr↑ ← [0, FALSE, FALSE, block, replace, 0, mapaddr, wordsperline, x,
      y, dw, dh, mapaddr, wordsperline, oldx, oldy, 0, 0, 0, 0];
    BitBltDefs.BITBLT[bbptr];
    IF x # oldx THEN -- first check if moved in x
      BEGIN
      IF x > oldx THEN BEGIN dlx ← oldx; dw ← MIN[x-oldx, oldw]; END
                  ELSE BEGIN
                       dlx ← MAX[oldx, x+MIN[rectangle.cw, oldw]];
                       dw ← (oldx+oldw) - dlx;
                       END;
      dty ← oldy;
      dh ← oldh;
      bbptr↑ ← [,,, gray, replace,,,, dlx, dty, dw, dh,,,,,,,,];
      BitBltDefs.BITBLT[bbptr];
      END;
    IF y # oldy THEN -- now see if moved in y
      BEGIN
      IF y > oldy
         THEN BEGIN dty ← oldy; dh ← MIN[y-oldy, oldh]; END
         ELSE BEGIN
              dty ← MAX[oldy, y+MIN[ rectangle.ch, oldh]];
              dh ← (oldy+oldh)-dty;
              END;
      dlx ← oldx;
      dw ← oldw;
      bbptr↑ ← [,,, gray, replace,,,, dlx, dty, dw, dh,,,,,,,,];
      BitBltDefs.BITBLT[bbptr];
      END;
    END;

  RectangleError: PUBLIC SIGNAL [rectangle: Rptr, error: RectangleErrorCode] = CODE;

  RectangleToMapCoords: PUBLIC PROCEDURE [rectangle: Rptr, x: xCoord, y: yCoord]
    RETURNS[mapx: xCoord, mapy: yCoord] =
    BEGIN
    mapx ← rectangle.x0 + MAX[0, INTEGER[MIN[rectangle.cw, x]]];
    mapy ← rectangle.y0 + MAX[0, INTEGER[MIN[rectangle.ch, y]]];
    END;

  RestoreRectangle: PUBLIC PROCEDURE [rectangle: Rptr, SegPtr: POINTER] =
    BEGIN OPEN BitBltDefs;
    bbTable:  ARRAY[0..SIZE[BBTable]] OF WORD;
    bbptr:  BBptr = EVEN[BASE[bbTable]];
    wordsperline:  INTEGER = rectangle.bitmap.wordsperline;
    mapaddr:  BMptr = rectangle.bitmap.addr;
    dw:  CARDINAL ← rectangle.cw;
    dwWords:  CARDINAL ← (dw + 15)/16 + 1;
    bbptr↑ ← BBTable[0,FALSE,FALSE,block,replace,0,mapaddr,wordsperline,
      rectangle.x0,rectangle.y0,dw,rectangle.ch,SegPtr,
      dwWords,0,0,0,0,0,0];
    BITBLT[bbptr];
    SystemDefs.FreeSegment[SegPtr];
    END;

  SaveRectangle: PUBLIC PROCEDURE [rectangle: Rptr] RETURNS [sptr: POINTER] =
    BEGIN OPEN BitBltDefs;
    bbTable:  ARRAY[0..SIZE[BBTable]] OF WORD;
    bbptr:  BBptr = EVEN[BASE[bbTable]];
    wordsperline:  CARDINAL = rectangle.bitmap.wordsperline;
    mapaddr:  BMptr = rectangle.bitmap.addr;
    dw:  CARDINAL ← rectangle.cw;
    dh:  CARDINAL ← rectangle.ch;
    dwWords:  CARDINAL ← (dw + 15)/16 + 1;
    totalWords:  CARDINAL ← dwWords * dh;
    sptr ← SystemDefs.AllocateSegment[totalWords];
    bbptr↑ ← BBTable[0,FALSE,FALSE,block,replace,0,sptr,dwWords,
      0,0,dw,dh,mapaddr,wordsperline,rectangle.x0,
      rectangle.y0,0,0,0,0];
    BITBLT[bbptr];
```

```
    END;

  ScrollBoxInRectangle: PUBLIC PROCEDURE [
    rectangle: Rptr, x0, width: xCoord, y0, height: yCoord, incr: INTEGER] =
    BEGIN
    bbtable: ARRAY [0..SIZE[BitBltDefs.BBTable]] OF WORD;
    bbptr: BitBltDefs.BBptr = EVEN[BASE[bbtable]];
    mapaddr: BMptr ← rectangle.bitmap.addr;
    wordsperline: CARDINAL = rectangle.bitmap.wordsperline;
    dlx: xCoord ← rectangle.x0+x0;
    dw: xCoord ← MIN[rectangle.cw, width];
    dh: yCoord ← MIN[rectangle.ch, (height-incr)];
    dty: yCoord;
    sty: yCoord;
    IF incr > 0 THEN BEGIN
                     dty ← rectangle.y0+y0;
                     sty ← dty+incr;
                     END
                ELSE BEGIN
                     sty ← rectangle.y0+y0;
                     dty ← sty+incr;
                     END;
    bbptr↑ ← [0, FALSE, FALSE, block, replace, 0, mapaddr, wordsperline,
      dlx, dty, dw, dh, mapaddr, wordsperline, dlx, sty,,,,];
    BitBltDefs.BITBLT[bbptr];
    END;

  WriteRectangleChar: PUBLIC PROCEDURE [
    rectangle: Rptr, x: xCoord, y: yCoord, char: CHARACTER, pfont: FAptr]
    RETURNS[xCoord, yCoord] =
    BEGIN
    bbtable: ARRAY [0..SIZE[BitBltDefs.BBTable]] OF WORD;
    bbptr: BitBltDefs.BBptr = EVEN[BASE[bbtable]];
    cw: FCDptr;
    code: CARDINAL ← LOOPHOLE[char];
    cwidth:  xCoord;
    lineheight: CARDINAL;
    IF pfont=NIL THEN [pfont, lineheight] ← GetDefaultFont[]
                 ELSE lineheight←LOOPHOLE[(pfont-SIZE[FontHeader])↑[0]];
    cwidth ← ComputeCharWidth[char, pfont];
    IF rectangle.visible = FALSE
       THEN IF rectangle.options.NoteInvisible THEN
                SIGNAL RectangleError[rectangle, NotVisible]
       ELSE RETURN[x, y];
    IF y+lineheight >= rectangle.ch
       THEN IF rectangle.options.NoteOverflow THEN
                SIGNAL RectangleError[rectangle,BottomOverflow]
       ELSE RETURN[x, y];
    IF x+cwidth > rectangle.cw
       THEN IF rectangle.options.NoteOverflow THEN
            SIGNAL RectangleError[rectangle, RightOverflow]
       ELSE RETURN[x, y];
    bbptr↑ ← [
      pad:, 0,
      sourcealt: FALSE,
      destalt: FALSE,
      sourcetype: block,
      function: paint,
      unused:,
      dbca: rectangle.bitmap.addr,
      dbmr: rectangle.bitmap.wordsperline,
      dlx: x+rectangle.x0,
      dty:,
      dw: 16,
      dh:,
      sbca:,
      sbmr: 1,
      slx: 0,
      sty: 0,
      gray0:, gray1:, gray2:, gray3:];
    DO cw ← LOOPHOLE[pfont[code]+LOOPHOLE[pfont,CARDINAL]+code];
       bbptr.dty ← y + rectangle.y0 + cw.height;
       bbptr.sbca ← cw - (bbptr.dh ← cw.displacement);
       IF cw.HasNoExtension
          THEN BEGIN
               bbptr.dw ← cw.widthORext;
```

```
            BitBltDefs.BITBLT[bbptr];
            RETURN[x+cwidth, y];
            END
        ELSE BEGIN
            BitBltDefs.BITBLT[bbptr];
            bbptr.dlx ← bbptr.dlx+16;
            END;
      code ← cw.widthORext;
      ENDLOOP;
  END;

WriteRectangleString: PUBLIC PROCEDURE [
  rectangle: Rptr, x: xCoord, y: yCoord, str: STRING, pfont: FAptr]
  RETURNS[xCoord, yCoord] =
  BEGIN
  i: CARDINAL;
  FOR i IN [0..str.length) DO
    [x, y] ← WriteRectangleChar[rectangle, x, y, str[i], pfont];
    ENDLOOP;
  RETURN[x,y]
  END;


END. of Rectangles
```

```
-- Selections.Mesa
-- Last edited by Beau Sheil on April 23, 1979  5:46 PM

--  This stuff is in a separate module to keep down swapping traffic
--  when not using file windows

DIRECTORY
  AltoFileDefs: FROM "altofiledefs",
  IODefs: FROM "iodefs",
  RectangleDefs: FROM "rectangledefs",
  StreamDefs: FROM "streamdefs" USING [
    EqualIndex, GetFA, GrEqualIndex, GrIndex, JumpToFA, ModifyIndex,
    SetIndex, StreamHandle, StreamError, StreamIndex],
  StringDefs: FROM "stringDefs",
  SystemDefs: FROM "systemdefs" USING [AllocateHeapString],
  WindowDefs: FROM "windowdefs";

Selections: PROGRAM
  IMPORTS RectangleDefs, StreamDefs, StringDefs,
          SystemDefs, WindowDefs
  EXPORTS WindowDefs =

  BEGIN OPEN StreamDefs, RectangleDefs, WindowDefs;

-- Selection routines

  ResolveBugToPosition: PUBLIC PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord]
    RETURNS [line: CARDINAL, xpos: xCoord, width: CARDINAL, index: StreamIndex] =
    BEGIN
    char: CHARACTER;
    lastwidth, nlines, newpos: CARDINAL;
    -- NOTE: linestarts array is ONE origin
    linestarts: DESCRIPTOR FOR ARRAY OF StreamIndex;
    IF w.file = NIL THEN RETURN;
    nlines ← (w.rectangle.ch/w.dinfo.lineheight)-1;
    linestarts ← DESCRIPTOR[GetLineTable[], nlines+1];
    IF EqualIndex[linestarts[0], NullIndex] THEN -- empty window
      RETURN[1, leftmargin, 0, OriginIndex];
    [x, y] ← CursorToRecCoords[w.rectangle, x, y];
    -- NOTE:  real line number = line + 1
    line ← MIN[MAX[y/w.dinfo.lineheight, 1], nlines];
    -- back up until real text in window
    WHILE EqualIndex[linestarts[line-1],NullIndex] DO line←line-1 ENDLOOP;
    index ← linestarts[line-1];
    SetIndex[w.file, index];
    xpos ← leftmargin;
    width ← 0;
    IF x <= xpos
       THEN BEGIN
            char ← w.file.get[w.file ! StreamError => GOTO Exit];
            width ← IF char = 11C THEN ComputeTabWidth[w.dinfo.pfont, xpos]
                                  ELSE ComputeCharWidth[char, w.dinfo.pfont];
            EXITS Exit => NULL;
            END
      ELSE
      DO
        char ← w.file.get[w.file ! StreamError => EXIT];
        lastwidth ← width;
        width ← IF char=IODefs.TAB THEN ComputeTabWidth[w.dinfo.pfont,xpos]
                                   ELSE ComputeCharWidth[char, w.dinfo.pfont];
        IF char = IODefs.CR OR char = IODefs.ControlZ THEN
          BEGIN index ← ModifyIndex[index, -1]; width ← lastwidth; EXIT; END;
        IF EqualIndex[index, w.eofindex] OR
           EqualIndex[ModifyIndex[index, 1],linestarts[line]] OR
           x < (newpos ← xpos + width) THEN EXIT;
        xpos ← newpos;
        index ← ModifyIndex[index, 1];
        ENDLOOP;
    RETURN[line, xpos, width, index];
    END;

  GetSelection: PUBLIC PROCEDURE [w: WindowHandle] RETURNS [str: STRING] =
    BEGIN
    i, count: CARDINAL ← 0;
    fa: AltoFileDefs.FA;
    char: CHARACTER;
```

```
    -- put the selection into a string
    IF EqualIndex[w.selection.rightindex, NullIndex] THEN RETURN[NIL];
    GetFA[w.file, @fa];
    count ← (w.selection.rightindex.page-w.selection.leftindex.page)*512 +
      (w.selection.rightindex.byte+1)-w.selection.leftindex.byte;
    str ← SystemDefs.AllocateHeapString[count];
    SetIndex[w.file, w.selection.leftindex];
    DO
      char ← w.file.get[w.file];
      IF char = IODefs.ControlZ THEN --throw out ControlZ formatting
        UNTIL char = IODefs.CR DO
          i ← i + 1;
          char ← w.file.get[w.file];
          ENDLOOP;
      StringDefs.AppendChar[str, char];
      IF (i ← i + 1) >= count THEN EXIT;
      ENDLOOP;
    JumpToFA[w.file, @fa];
    END;

  MakeSelection: PUBLIC PROCEDURE [w: WindowHandle, sel: POINTER TO Selection] =
    BEGIN
    -- unmark the old one if one exists and is visible
    IF NOT (EqualIndex[w.selection.leftindex, NullIndex]
      OR w.selection.leftline = 0) THEN MarkSelection[w];
    -- mark the new one
    w.selection ← sel↑;
    MarkSelection[w];
    END;

  MarkSelection: PUBLIC PROCEDURE [w: WindowHandle] =
    BEGIN
    i: CARDINAL;
    IF EqualIndex[w.selection.leftindex,NullIndex]
      OR w.selection.leftline = 0 THEN RETURN;
    IF w.selection.leftline = w.selection.rightline THEN
      InvertBoxInRectangle[w.rectangle, w.selection.leftx,
        w.selection.rightx-w.selection.leftx,
        w.selection.leftline*w.dinfo.lineheight, w.dinfo.lineheight]
    ELSE
      BEGIN
      InvertBoxInRectangle[w.rectangle, w.selection.leftx,
        (w.rectangle.cw-1)-w.selection.leftx,
        w.selection.leftline*w.dinfo.lineheight, w.dinfo.lineheight];
      i ← w.selection.rightline-w.selection.leftline;
      IF i > 1 THEN
        InvertBoxInRectangle[
    w.rectangle, leftmargin, w.rectangle.cw-(leftmargin+1),
    (w.selection.leftline+1)*w.dinfo.lineheight, (i-1)*w.dinfo.lineheight];
      InvertBoxInRectangle[
        w.rectangle, leftmargin, w.selection.rightx-leftmargin,
        w.selection.rightline*w.dinfo.lineheight, w.dinfo.lineheight];
      END;
    END;

  UpdateSelection:  PUBLIC PROCEDURE[w:  WindowHandle] =
    BEGIN
    --local data
    lastindex:  StreamIndex ← NullIndex;
    fa: AltoFileDefs.FA;
    line:  CARDINAL;
    nlines:  CARDINAL ← (w.rectangle.ch/w.dinfo.lineheight) - 1;
    i:  CARDINAL ← 0;
    linestarts : DESCRIPTOR FOR ARRAY OF StreamIndex;
    linestarts ← DESCRIPTOR[GetLineTable[], nlines +1];
    --figure out the real last line
    FOR line IN [1..nlines) DO
      IF EqualIndex[NullIndex,linestarts[line]] THEN
        BEGIN lastindex ← w.eofindex; nlines ← line-1; EXIT; END;
      REPEAT
        FINISHED => lastindex ← linestarts[nlines];
      ENDLOOP;
    -- for no selection or out of bounds
    IF EqualIndex[w.selection.leftindex, NullIndex] OR
      GrEqualIndex[w.selection.leftindex, lastindex] OR
      GrIndex[linestarts[0], w.selection.rightindex] THEN
```

```
    BEGIN w.selection.leftline ← 0; RETURN; END;
  GetFA[w.file, @fa];
  -- find line numbers
  IF GrEqualIndex[w.selection.leftindex, linestarts[0]] THEN
    FOR i ← 0, i+1 UNTIL i = nlines DO
      IF (GrEqualIndex[w.selection.leftindex, linestarts[i]]
      AND GrIndex[linestarts[i+1], w.selection.leftindex]) THEN EXIT;
      ENDLOOP;
  w.selection.leftline ← MAX[1, i+1];
  IF GrIndex[lastindex, w.selection.rightindex] THEN
    FOR i ← i, i+1 UNTIL i = nlines DO
      IF (GrEqualIndex[w.selection.rightindex, linestarts[i]]
      AND GrIndex[linestarts[i+1], w.selection.rightindex]) THEN EXIT;
      ENDLOOP;
  w.selection.rightline ← MIN[nlines+1, i+1];
  --find xcoords
  w.selection.leftx ←
    IF GrEqualIndex[w.selection.leftindex, linestarts[0]]
       THEN GetPos[w, linestarts[w.selection.leftline-1],
                   w.selection.leftindex, TRUE]
       ELSE leftmargin;
  w.selection.rightx ←
    IF GrIndex[lastindex, w.selection.rightindex]
       THEN GetPos[w, linestarts[w.selection.rightline-1],
                   w.selection.rightindex, FALSE]
       ELSE w.rectangle.cw;
  JumpToFA[w.file, @fa];
  MarkSelection[w];
  END;

ComputeTabWidth: PROCEDURE [font: FAptr, x: xCoord] RETURNS [CARDINAL] =
  BEGIN
  tw: CARDINAL = ComputeCharWidth[' ,font] * 8;
  RETURN[tw - LOOPHOLE[x-leftmargin,CARDINAL] MOD tw]
  END;

GetPos: PROCEDURE [w: WindowHandle, linestart: StreamIndex,
                   index2: StreamIndex, left: BOOLEAN]
        RETURNS [xCoord] =
  BEGIN
  char:  CHARACTER;
  xpos:  xCoord ← leftmargin;
  width: CARDINAL;
  IF GrIndex[linestart, index2] THEN RETURN[xpos];
  IF EqualIndex[linestart, index2] AND left THEN RETURN[xpos];
  SetIndex[w.file, linestart];
  DO
    char ← w.file.get[w.file];
    width ← IF char = 11C THEN ComputeTabWidth[w.dinfo.pfont,xpos]
      ELSE ComputeCharWidth[char, w.dinfo.pfont];
    xpos ← xpos + width;
    IF EqualIndex[linestart, index2] THEN EXIT;
    linestart ← ModifyIndex[linestart, 1];
    ENDLOOP;
  IF left THEN xpos ← xpos - width;
  RETURN[xpos];
  END;

END.  of Selections
```

```
-- Windowdefs.Mesa
-- Last edited by Beau Sheil on April 20, 1979  11:20 AM

DIRECTORY
 AltoDefs: FROM "AltoDefs",
 MenuDefs: FROM "menudefs",
 RectangleDefs: FROM "rectangledefs",
 SegmentDefs: FROM "segmentdefs",
 StreamDefs: FROM "streamdefs"
             USING [DisplayHandle, StreamHandle, DiskHandle, StreamIndex];

WindowDefs: DEFINITIONS =

  BEGIN OPEN RectangleDefs, StreamDefs;

  -- Window Types

  WindowHandle: TYPE = POINTER TO DisplayWindow;

  DisplayWindow: TYPE = RECORD [
    link: WindowHandle ← NIL,
    type: WindowType ← ,
    name: STRING ← NIL,
    menu: MenuDefs.MenuHandle ← NIL,
    displayproc: PROCEDURE [WindowHandle] ← NullProc,
    dinfo: WindowDisplayInfo ← DefaultWindowDisplayInfo,
    rectangle: Rptr ← ,
    ds: DisplayHandle ← NIL,
    ks: StreamHandle ← NIL,
    file: DiskHandle ← NIL,
    fileindex, tempindex, eofindex: StreamIndex ← NullIndex,
    selection: Selection ← NULL];

  WindowDisplayInfo: TYPE = RECORD [
      options: RECORD [StopRight, StopBottom, NoteLineBreak, NoteScrolling:
                       BOOLEAN],
      charx: xCoord ← 1,
      chary: yCoord ← 1,
      pfont: FAptr ← NIL,
      line, lineheight: CARDINAL ← 0];

  DefaultWindowDisplayInfo: WindowDisplayInfo
      = WindowDisplayInfo[options: [FALSE, FALSE, FALSE, FALSE]];

  Selection: TYPE = RECORD [
    leftx, rightx: xCoord,
    leftline, rightline: CARDINAL,
    leftindex, rightindex: StreamIndex];

  WindowType: TYPE = {clear, random, scratch, file, scriptfile};

  OriginIndex: StreamIndex = StreamIndex[0, 0];
  NullIndex: StreamIndex = StreamIndex[0, AltoDefs.BytesPerPage+1];

  -- procedures implementing Windows

  CreateDisplayWindow: PROCEDURE [WindowType, Rptr, DisplayHandle,
                                  StreamHandle, STRING]
                       RETURNS [WindowHandle];
  AlterWindowType: PROCEDURE [WindowHandle, WindowType, STRING];
  DestroyDisplayWindow: PROCEDURE [WindowHandle];
  UnlinkDisplayWindow: PROCEDURE [WindowHandle];
  PaintDisplayWindow: PROCEDURE [WindowHandle];
  DrawDisplayWindow: PROCEDURE [WindowHandle];
  FindDisplayWindow: PROCEDURE [xCoord, yCoord]
                     RETURNS [WindowHandle, xCoord, yCoord];
  NullProc: PROCEDURE [WindowHandle];
  SetCurrentDisplayWindow: PROCEDURE [WindowHandle];
  SetFileForWindow: PROCEDURE [WindowHandle, STRING];
  SetFileHandleForWindow: PROCEDURE [WindowHandle, SegmentDefs.FileHandle, STRING];
  SetIndexForWindow: PROCEDURE [WindowHandle, StreamIndex];
  SetPositionForWindow: PROCEDURE [WindowHandle, CARDINAL];
  GetCurrentDisplayWindow: PROCEDURE RETURNS [WindowHandle];
  RepaintDisplayWindows: PROCEDURE [mapdata: BMHandle];
  BlinkCursor: PROCEDURE RETURNS [BOOLEAN];
  GetLineTable: PROCEDURE RETURNS [POINTER];
```

```
-- Procedures Implementing Selections

  ResolveBugToPosition: PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord]
    RETURNS [line: CARDINAL, xpos: xCoord, width: CARDINAL,index: StreamIndex];
  MakeSelection: PROCEDURE [w: WindowHandle, sel: POINTER TO Selection];
  MarkSelection: PROCEDURE [w: WindowHandle];
  GetSelection: PROCEDURE [w: WindowHandle] RETURNS [STRING];
  UpdateSelection: PROCEDURE [w: WindowHandle];

  END. of windowdefs
```

```
-- Windows.Mesa
-- Last edited by Beau Sheil on April 23, 1979  8:15 PM

DIRECTORY
  BitBltDefs: FROM "bitbltdefs" USING [BBptr, BBTable, BITBLT],
  IODefs: FROM "iodefs",
  RectangleDefs: FROM "rectangledefs",
  SegmentDefs: FROM "segmentdefs" USING [
    Append, DefaultVersion, FileHandle, LockFile, NewFile, Read, UnlockFile,
    Write],
  StreamDefs: FROM "streamdefs" USING [
    AccessOptions, CleanupDiskStream, CloseDiskStream, CreateByteStream,
    DisplayHandle, FileLength, GetIndex, GrIndex, ModifyIndex, NormalizeIndex, OpenDiskStream, SetIndex
**,
    StreamHandle, StreamError, StreamIndex, StreamErrorCode, OpenKeyStream],
  StringDefs: FROM "stringDefs",
  SystemDefs: FROM "systemdefs" USING [
    AllocateHeapNode, AllocateHeapString, FreeHeapNode, FreeHeapString],
  WindowDefs: FROM "windowdefs";

Windows: PROGRAM
  IMPORTS BitBltDefs, RectangleDefs, SegmentDefs, StreamDefs, StringDefs,
          SystemDefs, WindowDefs
  EXPORTS WindowDefs =

  BEGIN OPEN RectangleDefs, StreamDefs, WindowDefs;

-- Local Symbols

  blinkOn: BOOLEAN ← FALSE;
  maxlines: CARDINAL = 80;
  maxwindows: CARDINAL = 15;

  xmloc: POINTER = LOOPHOLE[424B];          -- mouse locations
  ymloc: POINTER = LOOPHOLE[425B];
  xcloc: POINTER = LOOPHOLE[426B];
  ycloc: POINTER = LOOPHOLE[427B];

-- Local bindings

  currentwindow: WindowHandle ← NIL;
  linestarts: ARRAY [1..maxlines] OF StreamIndex;

-- Window Routines

  CreateDisplayWindow: PUBLIC PROCEDURE [type: WindowType,
    rectangle: Rptr, ds: DisplayHandle, ks: StreamHandle, name: STRING]
    RETURNS [w: WindowHandle] =
    BEGIN
    w ← SystemDefs.AllocateHeapNode[SIZE[DisplayWindow]];
    w↑ ← DisplayWindow[type: type, rectangle: rectangle,
                       displayproc: NullProc, ds: ds, ks: ks];
    [w.dinfo.pfont, w.dinfo.lineheight] ← GetDefaultFont[];
    AlterWindowType[w, type, name];
    SetCurrentDisplayWindow[w];
    END;

  AlterWindowType: PUBLIC PROCEDURE [ w: WindowHandle, type: WindowType, name: STRING] =
    BEGIN
    -- first undo all stuff for current type
    SELECT w.type FROM
      clear, random => NULL;          -- ignore
      scratch, scriptfile, file =>  -- data is a window on file
        BEGIN
        IF w.file # NIL THEN
          BEGIN
          w.file.destroy[w.file];
          w.file ← NIL;
          END;
        END;
      ENDCASE;
    -- now fixup all stuff for new type
    w.type ← type;
    SELECT type FROM
      clear, random => NULL;          -- ignore
      scratch, scriptfile, file =>  -- data is a window on file
```

```
      IF name # NIL THEN SetFileForWindow[w, name];
    ENDCASE;
  -- and set name (if not done already)
  IF w.name # name THEN
    BEGIN
    IF w.name # NIL THEN SystemDefs.FreeHeapString[w.name];
    w.name ← SystemDefs.AllocateHeapString[name.length];
    StringDefs.AppendString[w.name, name];
    END;
  -- set current selection null
  w.selection ← Selection[leftmargin,leftmargin,1,1,NullIndex, NullIndex];
  -- setup Stream options based upon stream existance
  IF w.ds # NIL THEN
    BEGIN
    w.ds.put ← WriteWindowChar;
    w.dinfo.options.NoteLineBreak ← TRUE;
    w.dinfo.options.NoteScrolling ← TRUE;
    w.dinfo.options.StopBottom ← type = file;
    END;
  END;

DestroyDisplayWindow: PUBLIC PROCEDURE [w: WindowHandle] =
  BEGIN                    -- clear, unlink, deallocate and return
  rectangle: Rptr = w.rectangle;
  ClearBoxInRectangle[rectangle, 0, rectangle.cw, 0, rectangle.ch];
  UnlinkDisplayWindow[w];
  IF w.file # NIL THEN w.file.destroy[w.file];
  SystemDefs.FreeHeapNode[w];
  -- later!! must undo anything done to StreamObject
  END;

GetCurrentDisplayWindow: PUBLIC PROCEDURE RETURNS [WindowHandle] =
  BEGIN
  RETURN[currentwindow];
  END;

RepaintDisplayWindows: PUBLIC PROCEDURE [mapdata: BMHandle] =
  BEGIN
    PaintProc: PROCEDURE [w: WindowHandle] =
      BEGIN
      IF w#NIL THEN BEGIN
                    IF w.link # currentwindow THEN PaintProc[w.link];
                    SetCurrentDisplayWindow[w];
                    END;
      END;
    PaintProc[currentwindow];
  END;

PaintDisplayWindow: PUBLIC PROCEDURE [w: WindowHandle] =
  BEGIN
  rectangle: Rptr = w.rectangle;
  IF NOT w.rectangle.visible THEN RETURN;      -- it isn't visible
  -- clear it and draw a box around it
  IF w = currentwindow THEN blinkOn ← FALSE;
  ClearBoxInRectangle[rectangle, 0, rectangle.cw, 0, rectangle.ch];
  DrawDisplayWindow[w];
  -- do type dependent stuff
  SELECT w.type FROM
    clear,       -- window is simply cleared on activation
    random => w.displayproc[w];  -- dispatch to repaint procedure
    scratch, -- data in scratch file
    scriptfile, -- data in typescript file
    file =>  -- window on a file
      IF w.file # NIL THEN w.displayproc[w];  -- dispatch to procedure
    ENDCASE;
  UpdateSelection[w];
  END;

DrawDisplayWindow: PUBLIC PROCEDURE [w: WindowHandle] =
  BEGIN
  pfont: FAptr;
  rectangle: Rptr = w.rectangle;
  x,y,lineheight: INTEGER;
  [pfont, lineheight] ← GetDefaultFont[];
  IF w.name # NIL THEN                    -- write box name
    [x,y] ← WriteRectangleString[rectangle, 2, 1, w.name, pfont
```

```
      ! RectangleError =>
        SELECT error FROM
          NotVisible,
          RightOverflow,
          BottomOverflow => CONTINUE;
          ENDCASE];
  -- invert the top stripe;
  InvertBoxInRectangle[rectangle, 0, rectangle.cw, 0, lineheight + 1];
  -- draw box arround the edge;
  DrawBoxInRectangle[rectangle, 0, rectangle.cw, 0, rectangle.ch];
  END;

FindDisplayWindow: PUBLIC PROCEDURE [x: xCoord, y: yCoord]
  RETURNS [wptr: WindowHandle, wx: xCoord, wy: yCoord] =
  -- Takes Cursor coordinates and tries to find the "top most"
  -- window for them. Returns the x,y in window coords
  BEGIN
  rectangle: Rptr;
  slop:  INTEGER ← 5;
  -- now check windows
  FOR wptr ← currentwindow, wptr.link UNTIL wptr=NIL
    DO rectangle ← wptr.rectangle;
       wx ← x - (rectangle.x0 + rectangle.bitmap.x0);
       wy ← y - (rectangle.y0 + rectangle.bitmap.y0);
       IF (wx >= -slop) AND (wx <= rectangle.width + slop) AND
          (wy >= -slop) AND (wy <= rectangle.height + slop)
          THEN RETURN;
       IF wptr=currentwindow THEN
          IF slop=0 THEN EXIT ELSE slop ← 0;
       ENDLOOP;
  RETURN[NIL, 0, 0];  -- couldnt find it
  END;

SetCurrentDisplayWindow: PUBLIC PROCEDURE [w: WindowHandle] =
  BEGIN
  SELECT currentwindow FROM
    NIL     => w.link ← w;    -- knot his tail; ring is currently empty
    w       => BEGIN PaintDisplayWindow[w]; RETURN; END; -- refresh only
    ENDCASE => BEGIN
               next: WindowHandle;
               IF blinkOn THEN [] ← BlinkCursor[];
               -- unlink if linked, then relink
               IF w.link # NIL THEN UnlinkDisplayWindow[w];
               FOR next ← currentwindow, next.link
                   UNTIL next.link = currentwindow
                   DO IF next=NIL THEN ERROR; ENDLOOP;
               w.link ← currentwindow;  next.link ← w;
               END;
  NewCurrentDisplayWindow[w];
  END;

UnlinkDisplayWindow: PUBLIC PROCEDURE [w: WindowHandle] =
  BEGIN                -- windows are linked in a ring!!
  prev: WindowHandle;
  FOR prev ← w, prev.link UNTIL prev.link=w
      DO IF prev=NIL THEN ERROR; ENDLOOP;
  IF w=currentwindow THEN
     NewCurrentDisplayWindow[IF prev=w THEN NIL ELSE w.link];
  prev.link ← w.link;  -- detach from ring
  w.link ← NIL;        -- mark as unlinked
  END;

NewCurrentDisplayWindow: PROCEDURE [new: WindowHandle] =
  BEGIN
  IF currentwindow#NIL THEN UndoDataSetup[currentwindow];
  IF (currentwindow ← new)#NIL THEN BEGIN
                                    DoDataSetup[new];
                                    PaintDisplayWindow[new];
                                    END;
  END;

DoDataSetup: PROCEDURE [w: WindowHandle] =
  BEGIN
  -- do everything to make this guy's data backup active
  SELECT w.type FROM
    clear,             -- window is simply cleared on activation
```

```
    random,              -- USERS responsibility to repaint screen
    scriptfile => NULL; -- data is maintained in typescript file
    scratch,             -- data is maintained in scratch file
    file =>              -- window on a file
      IF w.file # NIL THEN
        OpenDiskStream[w.file ! StreamError =>
          BEGIN  w.eofindex ← GetIndex[w.file]; RESUME END];
      ENDCASE;
  END;

UndoDataSetup: PROCEDURE [w: WindowHandle] =
  BEGIN
  -- do everything to make this guy's data backup inactive
  SELECT w.type FROM
    clear,           -- window is simply cleared on activation
    random => NULL; -- USERS responsibility to repaint screen
    scratch => -- data is maintained in scratch file
      IF w.file # NIL THEN
        BEGIN
        IF w.tempindex = NullIndex THEN w.eofindex ← GetIndex[w.file];
        CloseDiskStream[w.file];
        END;
    scriptfile =>  -- data is maintained in typescript file
      IF w.file # NIL THEN
        BEGIN
        IF w.tempindex = NullIndex THEN w.eofindex ← GetIndex[w.file];
        StreamDefs.CleanupDiskStream[w.file];
        END;
    file =>      -- window on a file
      IF w.file # NIL THEN StreamDefs.CloseDiskStream[w.file];
    ENDCASE;
  END;

DisplayFileData: PROCEDURE [w: WindowHandle] =
  BEGIN
  -- NOTE: this routine wants to be super efficient!!
  -- should use port to write characters
  bbtable: ARRAY [0..SIZE[BitBltDefs.BBTable]] OF WORD;
  bbptr: BitBltDefs.BBptr ←
    LOOPHOLE[BASE[bbtable]+LOOPHOLE[BASE[bbtable],CARDINAL] MOD 2];
  nchars, count: INTEGER;
  i, width: CARDINAL;
  cw: FCDptr;
  fullwin:  BOOLEAN ← FALSE;
  index: StreamIndex;
  char: UNSPECIFIED;
  pfont: FAptr = w.dinfo.pfont;
  x: CARDINAL ← w.dinfo.charx;
  x0: CARDINAL = w.rectangle.x0;
  y: CARDINAL ← w.dinfo.chary + w.rectangle.y0;
  -- check if really a file there
  IF w.file = NIL THEN RETURN;
  -- check if temporary positioning
  IF w.tempindex # NullIndex THEN
    BEGIN
    linestarts[w.dinfo.line] ← w.tempindex;
    SetIndex[w.file, w.tempindex];
    END
  ELSE
    BEGIN
    linestarts[w.dinfo.line] ← w.fileindex;
    SetIndex[w.file, w.fileindex];
    END;
  -- setup to do this fast
  bbptr↑ ← [
    pad: 0,
    sourcealt: FALSE,
    destalt: FALSE,
    sourcetype: block,
    function: paint,
    dbca: w.rectangle.bitmap.addr,
    dbmr: w.rectangle.bitmap.wordsperline,
    dlx:,
    dty:,
    dw:,
    dh:,
```

```
    sbca:,
    sbmr: 1,
    slx: 0,
    sty: 0,
    unused:,
    gray0:, gray1:, gray2:, gray3:];
  index ← GetIndex[w.file];
  nchars ← 0;
  count ←(IF w.type = file THEN 10000
                           ELSE
         (w.eofindex.page-index.page)*512 + (w.eofindex.byte-index.byte));
  -- fill the window with text
  WHILE count > 0 DO
    char ← w.file.get[w.file
      ! StreamError =>
        BEGIN
        w.eofindex ← GetIndex[w.file];
        EXIT;
        END];
    nchars ← nchars + 1;
    x ← (width ← ComputeCharWidth[char, pfont]) + x;
    IF char < 40C THEN
      BEGIN
      IF char = 32C THEN  --ControlZ formatting
        WHILE char # IODefs.CR DO
          char ← w.file.get[w.file
            ! StreamError =>
              BEGIN
              w.eofindex ← GetIndex[w.file];
              EXIT;
              END];
           ENDLOOP;
      w.dinfo.charx ← x - width;
      ScrollDisplay[w.ds, char
        ! StreamError =>
          IF stream = w.ds
             THEN BEGIN
                  IF error = StreamEnd AND w.dinfo.options.StopBottom
                     THEN BEGIN
                          linestarts[w.dinfo.line+1] ← ModifyIndex[index, nchars];
                          fullwin ← TRUE;
                          EXIT;
                          END
                     ELSE FixupOnOverflow[w, error, okay];
                  RESUME;
                  END];
      y ← w.dinfo.chary + w.rectangle.y0;
      x ← w.dinfo.charx;
      END
    ELSE IF x >= w.rectangle.cw THEN
      BEGIN
      w.dinfo.charx ← x - width;
      ScrollDisplay[w.ds, char
        ! StreamError =>
          IF stream = w.ds THEN
            BEGIN
            IF error = StreamEnd
            AND w.dinfo.options.StopBottom THEN
              BEGIN
              linestarts[w.dinfo.line+1] ← ModifyIndex[index, nchars-1];
              fullwin ← TRUE;
              EXIT;
              END
            ELSE FixupOnOverflow[w, error, backup];
            RESUME;
            END];
      y ← w.dinfo.chary + w.rectangle.y0;
      x ← w.dinfo.charx;
      END
    ELSE
      BEGIN
      bbptr.dlx ← x - width + x0;
      DO
        cw ← LOOPHOLE[pfont[char]+LOOPHOLE[pfont,CARDINAL]+char];
        bbptr.dty ← y + cw.height;
        bbptr.sbca ← cw - (bbptr.dh ← cw.displacement);
```

```
        IF cw.HasNoExtension THEN
          BEGIN
          bbptr.dw ← cw.widthORext;
          BitBltDefs.BITBLT[bbptr];
          EXIT
          END
        ELSE
          BEGIN
          BitBltDefs.BITBLT[bbptr];
          bbptr.dlx ← bbptr.dlx+16;
          END;
        char ← cw.widthORext;
        ENDLOOP;
      END;
    count ← count-1;
    ENDLOOP;
  w.dinfo.charx ← x;
  -- set remainder of line table null
  IF NOT fullwin THEN linestarts[w.dinfo.line +1] ← NullIndex;
  FOR i IN [w.dinfo.line+2..maxlines) DO
    linestarts[i] ← NullIndex;
    ENDLOOP;
  IF w.type = file AND GrIndex[w.fileindex, w.eofindex] THEN
    BEGIN
    w.eofindex ← FileLength[w.file];
    SetIndex[w.file, w.fileindex];
    END;
  END;

FixupOnOverflow: PROCEDURE [
  w: WindowHandle, error: StreamErrorCode, index: {okay, backup}] =
  -- NOTE: this routine is specific to windows using streams
  BEGIN
  -- define locals
  i: CARDINAL;
  -- fix up the line table based upon error code
  SELECT error FROM
    StreamEnd => -- scroll it all up one line
      BEGIN
      FOR i IN  [1..maxlines) DO linestarts[i] ←linestarts[i+1]; ENDLOOP;
      w.fileindex ← linestarts[1];
      END;
    StreamPosition => NULL;
    ENDCASE;
  -- set current position in correspondence table
  linestarts[w.dinfo.line] ← ModifyIndex[GetIndex[w.file],
                                         IF index=backup THEN -1 ELSE 0];
  END;

FindWindowWithStream: PROCEDURE [ds: DisplayHandle] RETURNS [w: WindowHandle] =
  BEGIN  -- runs around window ring until it finds it
  FOR w ← currentwindow, w.link UNTIL w=NIL OR w.ds = ds DO ENDLOOP;
  END;

BlinkCursor: PUBLIC PROCEDURE RETURNS [BOOLEAN] =
  BEGIN OPEN w: currentwindow;
  IF currentwindow = NIL OR (w.type # scratch AND w.type # scriptfile) OR
    w.file # NIL AND GetIndex[w.file] # w.eofindex THEN RETURN[blinkOn];
  RectangleDefs.InvertBoxInRectangle[
    w.rectangle, w.dinfo.charx+1, 2, w.dinfo.chary+1, w.dinfo.lineheight-2];
  RETURN[blinkOn ← ~blinkOn]
  END;

ScrollDisplay: PROCEDURE [ds: DisplayHandle, c: CHARACTER] =
  BEGIN  -- dummy procedure added in conversion to 5.0
  END;

ScrollWindow: PROCEDURE [W: WindowHandle, c: CHARACTER] =
  BEGIN  -- dummy procedure added in conversion to 5.0
  END;

WriteDisplayChar: PROCEDURE [w: WindowHandle, char: CHARACTER] =
  BEGIN  -- paste a character on the screen
    SELECT LOOPHOLE[char, CARDINAL] FROM
      1B, 10B, > 377B => RETURN;
      < 40B => ScrollWindow[w, char];
```

```
      ENDCASE =>
        [w.dinfo.charx, w.dinfo.chary] ← WriteRectangleChar[
          w.rectangle, w.dinfo.charx, w.dinfo.chary, char, w.dinfo.pfont
            ! RectangleError =>
              SELECT error FROM
                BottomOverflow, RightOverflow =>
                  BEGIN
                  ScrollWindow[w, char];
                  CONTINUE;
                  END;
                ENDCASE];
  END;

  WriteWindowChar: PROCEDURE [stream: StreamHandle, char: UNSPECIFIED] =
    BEGIN
    WITH ds:stream SELECT FROM
      Display =>                    -- make sure its a Display Stream
        BEGIN
        w: WindowHandle ← FindWindowWithStream[@ds];
        r: Rptr;
        index: StreamIndex;
        update:  BOOLEAN ← FALSE;
        IF blinkOn THEN [] ← BlinkCursor[];
        SELECT w.type FROM
          clear, -- window is simply cleared on activation
          random => WriteDisplayChar[w, char];
          scratch, -- data is maintained in scratch file
          scriptfile => -- data is maintained in typescript file
            BEGIN
            IF currentwindow.ds # @ds THEN
              BEGIN
              w.tempindex ← NullIndex;
              w.dinfo.options.StopBottom ← FALSE;
              SetCurrentDisplayWindow[w];
              IF w.ks # NIL THEN OpenKeyStream[w.ks];
              -- move the mouse into the window!!
              r ← w.rectangle;
              xmloc↑ ← r.bitmap.x0+r.x0+(r.width/2);
              ymloc↑ ← r.bitmap.y0+r.y0+(r.height/2);
              xcloc↑ ← xmloc↑;
              ycloc↑ ← ymloc↑;
              END
            ELSE
              IF w.tempindex # NullIndex THEN
                BEGIN
                w.dinfo.options.StopBottom ← FALSE;
                IF GetIndex[w.file] = w.eofindex THEN
                  BEGIN
                  w.fileindex ← w.tempindex;
                  w.tempindex ← NullIndex;
                  END
                ELSE
                  BEGIN  -- reposition file to the end
                  w.tempindex ← NullIndex;
                  PaintDisplayWindow[w];
                  END;
                END;
            SELECT char FROM
              IODefs.ControlA, IODefs.BS => -- back space character
                BEGIN
                index ← ModifyIndex[GetIndex[w.file], -1];
                w.eofindex ← index;
                SetIndex[w.file, index];
                END;
              ENDCASE =>
                BEGIN
                index ← GetIndex[w.file];
                w.file.put[w.file, char];
                w.eofindex ← ModifyIndex[index, 1];
                WriteDisplayChar[w, char
                  ! StreamError =>
                    IF stream = w.ds THEN
                      BEGIN
                      IF error = StreamEnd THEN
                        BEGIN
                        -- update the selection
```

```
                    w.selection.leftline ←
                      MAX[1, w.selection.leftline] - 1;
                    w.selection.rightline ←
                      MAX[1, w.selection.rightline] - 1;
                    END;
                  FixupOnOverflow[
    w, error, IF char # 15B THEN backup ELSE okay];
                  RESUME;
                  END
                ];
              END;
          END;
        file => NULL; -- window on a file
        ENDCASE;
      END;
    ENDCASE => ERROR StreamError[stream, StreamType];
  END;

GetLineTable: PUBLIC PROCEDURE RETURNS[POINTER] =  -- used by menus
  BEGIN RETURN[@linestarts] END;

SetFileForWindow: PUBLIC PROCEDURE [w: WindowHandle, filename: STRING]  =
  BEGIN
  SetFileHandleForWindow[w, NIL, filename];
  END;

SetFileHandleForWindow: PUBLIC PROCEDURE [
  w: WindowHandle, fileh: SegmentDefs.FileHandle, filename: STRING]  =
  BEGIN OPEN SegmentDefs;
  access: AccessOptions;
  -- do file type specific stuff
  SELECT w.type FROM
    scratch, scriptfile => access ← Read+Write+Append;
    file => access ← Read;
    ENDCASE =>  ERROR;
  -- verify file access is ok
  IF fileh = NIL THEN fileh ← NewFile[filename, access, DefaultVersion];
  -- if already one shit can it (and name too)
  IF w.file # NIL THEN
    BEGIN   --same file
    IF w.file.file = fileh THEN SegmentDefs.LockFile[w.file.file];
    w.file.destroy[w.file];
    IF w.file.file = fileh THEN SegmentDefs.UnlockFile[w.file.file];
    END;
  -- now create a stream associated with the file
  w.file ← CreateByteStream[fileh,access];
  -- set length based on type
  w.fileindex ← w.eofindex ← OriginIndex;
  SELECT w.type FROM
    scriptfile => NULL;  -- data is maintained in typescript file
    scratch =>  -- data is maintained in scratch file
      IF w # currentwindow THEN CloseDiskStream[w.file];
    file =>     -- data is a window on file
      BEGIN
      w.eofindex ← FileLength[w.file];
      IF w # currentwindow THEN CloseDiskStream[w.file];
      END;
    ENDCASE => ERROR;
  -- assign name and display procedure
  IF w.name # filename THEN
    BEGIN
    IF w.name # NIL THEN SystemDefs.FreeHeapString[w.name];
    w.name ← SystemDefs.AllocateHeapString[filename.length];
    StringDefs.AppendString[w.name, filename];
    END;
  w.tempindex ← NullIndex;
  w.displayproc ← DisplayFileData;
  -- set current selection null
  w.selection ← Selection[leftmargin,leftmargin,1,1,NullIndex, NullIndex];
  END;

SetIndexForWindow: PUBLIC PROCEDURE [w: WindowHandle, index: StreamIndex]  =
  BEGIN
  -- set fileposition
  SELECT w.type FROM
    scratch, scriptfile, file => w.fileindex ← index;
```

```
      ENDCASE;
    -- and paint it if it is the current one
    IF w = currentwindow THEN PaintDisplayWindow[w];
    END;

  SetPositionForWindow: PUBLIC PROCEDURE [w: WindowHandle, pos: CARDINAL]  =
    BEGIN
    -- define locals
    fileindex: StreamIndex;
    -- set fileposition
    SELECT w.type FROM
      scratch, scriptfile, file =>
   BEGIN
   fileindex ← NormalizeIndex[StreamIndex[0, pos]];
   SetIndexForWindow[w, fileindex];
   END;
      ENDCASE;
    END;

  NullProc: PUBLIC PROCEDURE [w: WindowHandle] =
    BEGIN
    -- Dummy Display procedure
    END;

  END.  of Windows
```

```
-- WStreamDefs.mesa
-- Last edited by Beau Sheil on April 20, 1979  2:58 PM

DIRECTORY
  RectangleDefs: FROM "rectangledefs" USING [Rptr],
  StreamDefs: FROM "streamdefs" USING [DisplayHandle, StreamHandle];

WStreamDefs: DEFINITIONS =

  BEGIN OPEN StreamDefs, RectangleDefs;

-- Window Display Stream Routines

  CreateDisplayStream: PUBLIC PROCEDURE [r:Rptr ← NULL] RETURNS [DisplayHandle];
  DestroyDisplayStream: PUBLIC PROCEDURE [StreamHandle];

-- Text Output and Support Routines

  WriteDisplayChar: PUBLIC PROCEDURE [StreamHandle, UNSPECIFIED];
  ClearCurrentLine: PUBLIC PROCEDURE [StreamHandle];
  ClearDisplayLine: PUBLIC PROCEDURE [StreamHandle, CARDINAL];
  ClearDisplayChar: PUBLIC PROCEDURE  [StreamHandle, CHARACTER];
  SetDisplayLine: PUBLIC PROCEDURE [DisplayHandle, CARDINAL, CARDINAL];
  ScrollDisplay: PUBLIC PROCEDURE [DisplayHandle, UNSPECIFIED];

  END. of WStreamDefs
```

```
-- WStreams.mesa
-- Last edited by Beau Sheil on April 22, 1979  5:43 PM

DIRECTORY
  IODefs: FROM "iodefs",
  RectangleDefs: FROM "rectangledefs" USING [
    BMHandle, BMptr, ClearBoxInRectangle, ComputeCharWidth, leftmargin,
    RectangleError, Rptr, ScrollBoxInRectangle, WriteRectangleChar],
  StreamDefs: FROM "streamdefs" USING [
    DisplayHandle, StreamError, StreamHandle, StreamObject],
  SystemDefs: FROM "systemdefs" USING [AllocateHeapNode, FreeHeapNode],
  WindowDefs: FROM "windowdefs",
  WStreamDefs: FROM "WStreamDefs";

WStreams: PROGRAM
  IMPORTS RectangleDefs, StreamDefs, SystemDefs, WindowDefs
  EXPORTS WStreamDefs =

  BEGIN OPEN StreamDefs, RectangleDefs, WindowDefs;

-- CHARACTER constants
  BackSpace: CHARACTER = IODefs.BS;
  Space: CHARACTER = IODefs.SP;
  MaxCharCode: CHARACTER = 377C;

-- Mesa Display Stream Routines

  CreateDisplayStream: PUBLIC PROCEDURE [rectangle: Rptr ← NULL]
                       RETURNS[ds: DisplayHandle] =
    BEGIN
    ds ← SystemDefs.AllocateHeapNode[SIZE[Display StreamObject]];
    ds↑ ← StreamObject[ResetDS, GetDS, PutbackDS, WriteDisplayChar,
                       EndofDS, DestroyDisplayStream, NIL,
          Display[ClearCurrentLine, ClearDisplayLine, ClearDisplayChar, , NIL]];
    END;

  DestroyDisplayStream: PUBLIC PROCEDURE [stream: StreamHandle] =
    BEGIN
    WITH ds:stream SELECT FROM
      Display => IF ThereIsWindowForDS[@ds] THEN SystemDefs.FreeHeapNode[@ds];
      ENDCASE => ERROR StreamError[stream, StreamType];
    END;

-- Text Output and Support Routines

  WriteDisplayChar: PUBLIC PROCEDURE [stream: StreamHandle, char: UNSPECIFIED] =
    BEGIN
    WITH ds:stream SELECT FROM
      Display =>
        BEGIN  -- paste a character on the screen
          SELECT char FROM
          1B, 10B, > 377B => RETURN;
          < 40B => ScrollDisplay[@ds, char];
          ENDCASE =>
            BEGIN
            w: WindowHandle = WindowForDS[@ds];
            [w.dinfo.charx, w.dinfo.chary] ← WriteRectangleChar[
              w.rectangle, w.dinfo.charx, w.dinfo.chary, char, w.dinfo.pfont
              ! RectangleError =>
                SELECT error FROM
                  BottomOverflow, RightOverflow => BEGIN
                                                   ScrollDisplay[@ds, char];
                                                   CONTINUE;
                                                   END;
                  ENDCASE];
            END;
        END;
      ENDCASE => ERROR StreamError[stream, StreamType];
    END;

  ClearCurrentLine: PUBLIC PROCEDURE [stream: StreamHandle]=
    BEGIN
    WITH ds:stream SELECT FROM
      Display => ClearDisplayLine[@ds, WindowForDS[@ds].dinfo.line];
      ENDCASE => ERROR StreamError[stream,StreamType];
    END;
```

```
ClearDisplayLine: PUBLIC PROCEDURE [stream: StreamHandle, line: CARDINAL] =
  BEGIN
  ds: DisplayHandle = WITH s:stream SELECT FROM
                        Display => @s,
                        ENDCASE => ERROR StreamError[stream,StreamType];
  w: WindowHandle = WindowForDS[ds];
  rectangle: Rptr = w.rectangle;
  lineheight: CARDINAL = w.dinfo.lineheight;
  IF line > rectangle.ch/lineheight THEN RETURN;
  ClearBoxInRectangle[rectangle, 1, rectangle.cw-2, lineheight*line, lineheight];
  -- reset line position to left margin
  SetDisplayLine[ds, line, leftmargin];
  END;

ClearDisplayChar: PUBLIC PROCEDURE  [stream: StreamHandle, char: CHARACTER] =
  -- erases last character written.  Assumptions: position not changed
  BEGIN OPEN IODefs;
  cwidth: CARDINAL;
  ds: DisplayHandle = WITH s:stream SELECT FROM
                        Display => @s,
                        ENDCASE => ERROR StreamError[stream,StreamType];
  w: WindowHandle = WindowForDS[ds];
  rectangle: Rptr = w.rectangle;
  SELECT char FROM
    NUL, CR, >MaxCharCode => RETURN;
    ControlA, BackSpace, TAB => NULL;
    < Space =>
      BEGIN
      ClearDisplayChar[ds,LOOPHOLE[LOOPHOLE[char,INTEGER]+100B, CHARACTER]];
      char ← '↑;
      END;
    ENDCASE => NULL;
  -- now backup convert stuff and compute left word
  cwidth ← ComputeCharWidth[char, w.dinfo.pfont];
  ds.put[ds, BackSpace];
  w.dinfo.charx ← MAX[leftmargin, LOOPHOLE[w.dinfo.charx-cwidth, INTEGER]];
  -- now clear it
  ClearBoxInRectangle[
    rectangle, w.dinfo.charx, cwidth, w.dinfo.chary, w.dinfo.lineheight];
  END;

SetDisplayLine: PUBLIC PROCEDURE [ds: DisplayHandle, line, pos: CARDINAL] =
  BEGIN
  w: WindowHandle = WindowForDS[ds];
  lineheight: CARDINAL = w.dinfo.lineheight;
  -- make sure line no. is in range and set character x and y
  line ← MIN[line, LOOPHOLE[(w.rectangle.ch/lineheight)-1,CARDINAL]];
  w.dinfo.charx ← pos;  w.dinfo.chary ← line*lineheight;
  w.dinfo.line ← line;
  END;

ScrollDisplay: PUBLIC PROCEDURE [ds: DisplayHandle, char: UNSPECIFIED] =
  BEGIN
  newx, tw: CARDINAL;  -- for TABs
  lastline: CARDINAL;
  lineheight: CARDINAL = w.dinfo.lineheight;
  blockheight: CARDINAL;
  w: WindowHandle = WindowForDS[ds];
  rectangle: Rptr = w.rectangle;
  mapaddr: BMptr = rectangle.bitmap.addr;
  wordsperline: INTEGER = rectangle.bitmap.wordsperline;
  SELECT char FROM
    15B => NULL; -- <Carriage Return>
    0, 12B => RETURN;  -- null character
    11B =>  -- TAB
      BEGIN
      tw ← ComputeCharWidth[' ,w.dinfo.pfont] * 8;
      newx ← (LOOPHOLE[w.dinfo.charx-leftmargin, CARDINAL]/tw+1)*tw+leftmargin;
      IF newx < rectangle.cw THEN BEGIN w.dinfo.charx ← newx; RETURN END;
      IF tw >= rectangle.cw THEN char ← ' ;
      END;
    ENDCASE =>
      BEGIN
      IF w.dinfo.options.StopRight THEN RETURN;
      IF char < 40B THEN
```

```
          BEGIN
          WriteDisplayChar[ds, '↑];
          WriteDisplayChar[ds, char+100B];
          RETURN;
          END;
        END;
    -- check if at bottom of window
    lastline ← ((rectangle.ch-1)/lineheight)-1;
    IF w.dinfo.line = lastline THEN
      BEGIN
      -- if he wants to be notified at the bottom, do it
      IF w.dinfo.options.NoteScrolling THEN SIGNAL StreamError[ds, StreamEnd];
      IF w.dinfo.options.StopBottom THEN RETURN;
      -- scroll the box containing the text
      blockheight ← rectangle.ch-lineheight-1;
      ScrollBoxInRectangle[rectangle, 0, rectangle.cw, lineheight,
  blockheight, lineheight];
      -- now clear the last line
      ClearDisplayLine[ds, w.dinfo.line];
      END
    ELSE
      BEGIN
      -- assumes next window line is cleared
      w.dinfo.line ← w.dinfo.line+1;
      SetDisplayLine[ds, w.dinfo.line, leftmargin];
      IF w.dinfo.options.NoteLineBreak THEN SIGNAL StreamError[ds, StreamPosition];
      END;
    -- now write the char
    IF char NOT= 15B THEN WriteDisplayChar[ds, char];
    END;

  ThereIsWindowForDS: PROCEDURE [ds: DisplayHandle] RETURNS [BOOLEAN] =
    BEGIN
      RETURN[WindowForDS[ds]#NIL];
    END;

  WindowForDS: PROCEDURE [ds: DisplayHandle] RETURNS [w: WindowHandle] =
    BEGIN                    -- finds the window containing a given ds
      FOR w ← GetCurrentDisplayWindow[], w.link UNTIL w=NIL
        DO IF w.ds=ds THEN RETURN;
           IF w=GetCurrentDisplayWindow[] THEN EXIT;
        ENDLOOP;
      ERROR StreamError[ds, StreamType];
    END;

-- NOP routines

  ResetDS: PROCEDURE [stream: StreamHandle]=
    BEGIN
    END;

  GetDS: PROCEDURE [stream: StreamHandle] RETURNS [UNSPECIFIED] =
    BEGIN
    ERROR StreamError[stream, StreamType];
    END;

  PutbackDS: PROCEDURE [stream: StreamHandle, char: UNSPECIFIED] =
    BEGIN
    ERROR StreamError[stream, StreamType];
    END;

  EndofDS: PROCEDURE [stream: StreamHandle] RETURNS [BOOLEAN] =
    BEGIN
    RETURN[FALSE];
    END;

  END. of WStreams
```