```
--File:  WManControl.mesa
--Edited by Sandman       October 7, 1977  9:14 AM

DIRECTORY
  AltoDefs:  FROM "altodefs",
  ControlDefs: FROM "controldefs",
  DoubleDefs: FROM "doubledefs",
  SegmentDefs:  FROM "segmentdefs",
  SystemDefs:  FROM "systemdefs",
  StringDefs:  FROM "stringdefs",
  StreamDefs:  FROM "streamdefs",
  InlineDefs:  FROM "inlinedefs",
  IODefs:  FROM "iodefs",
  KeyDefs: FROM "keydefs",
  MenuDefs:  FROM "menudefs",
  RectangleDefs:  FROM "rectangledefs",
  WindowDefs:  FROM "windowdefs",
  WManagerDefs:  FROM "wmanagerdefs";

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

WManControl:  PROGRAM
  IMPORTS DoubleDefs, SegmentDefs, SystemDefs, StringDefs, StreamDefs, RectangleDefs,
    WindowDefs, WManagerDefs
  EXPORTS WManagerDefs
  SHARES WManagerDefs, StreamDefs, MenuDefs =
  BEGIN

  WMState: WMDataHandle;

  --external proc
  --global data
  CursorA, CursorB, CursorC, CursorD, CursorE,
    CursorF, CursorG, CursorH, CursorI:  ARRAY[0..15] OF INTEGER;

  -- Window Manager Main Control Routine

  WindowManager: PUBLIC PROCEDURE =
    BEGIN OPEN WMState;
    -- Declare Locals
      x, y,i: INTEGER;
      k: KeySet;
      char: UNSPECIFIED;
      mousewindow: WindowHandle;
      buttons: AMouseButton;
      cw: WindowHandle ← GetCurrentDisplayWindow[];
      ds: DisplayHandle;
    -- check if need to service KeyStream
      IF cw.ks # defaultks AND NOT cw.ks.endof[cw.ks] THEN
        FOR i IN [0..maxscratch) DO ENABLE StreamDefs.StreamError => EXIT;
          IF scratchfiles[i] = cw.file THEN
            BEGIN
            ReadEditChar[cw.ks.get[cw.ks], cw];
            EXIT;
            END;
          ENDLOOP;
      cw ← GetCurrentDisplayWindow[];
    -- check if some part of cursor is in jump bar
      [x, y] ← CursorToRectangleCoords[cw.rectangle, xcursorloc↑, ycursorloc↑];
      IF x+slop > 0 AND x <= JumpStrip + CursorXAdjust[]
        AND y+slop > 0 AND y-slop <= cw.rectangle.ch THEN SetJumpStripe[cw, TRUE]
      ELSE SetJumpStripe[cw, FALSE];
      -- check mouse buttons
      buttons ← GetMouseButton[];
    -- look at the mouse and flip from one to the other
      cw ← GetCurrentDisplayWindow[ ! StreamDefs.StreamError => CONTINUE];
      [mousewindow, x, y] ← FindDisplayWindow[xcursorloc↑, ycursorloc↑
        ! StreamDefs.StreamError => CONTINUE];
      IF cw # mousewindow AND mousewindow # NIL AND buttons # None THEN
        BEGIN
        SetCurrentDisplayWindow[mousewindow
          ! SegmentDefs.InvalidFP =>
            BEGIN
            SetFileForWindow[mousewindow, mousewindow.name
              !  SegmentDefs.FileNameError =>
                 BEGIN
```

```
                AlterWindowType[mousewindow, mousewindow.type, NIL];
                CONTINUE;
                END
              ];
            RETRY;
            END
          ];
        OpenKeyStream[mousewindow.ks ! StreamDefs.StreamError => CONTINUE];
        END
      ELSE IF buttons = None THEN
        BEGIN
        IF useKeyset 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 IF FL4Down[] THEN
          BEGIN StuffSel[cw]; WHILE FL4Down[] DO NULL ENDLOOP; END;
        END
      ELSE
        BEGIN
        THROUGH [0..700) DO NULL ENDLOOP;  -- Debounce Mouse
        ButtonProcArray[GetMouseButton[]][cw, xcursorloc↑,ycursorloc↑ !
          StreamDefs.StreamError => CONTINUE];
        END;
    END;

  ReadEditChar: PROCEDURE
      [char: CHARACTER, w: WindowHandle]=
    BEGIN
    --declare locals
    index: StreamIndex;
    fixup: BOOLEAN ← FALSE;
    firstchar: BOOLEAN ← TRUE;
    ch: CHARACTER;
    controlA: CHARACTER = 1C;
    controlH: CHARACTER = 10C;
    controlW: CHARACTER = 27C;
    controlQ: CHARACTER = 21C;
    Space: CHARACTER = 40C;
    --do editing like ReadEditString
    SELECT char FROM
       controlA, controlH =>
         BEGIN
         IF w.ds.charx # leftmargin THEN
           BEGIN
           index ← GetIndex[w.file];
           w.eofindex ← index;
           index ← ModifyIndex[index, -1];
           IF EqualIndex[w.selection.rightindex, index] THEN
             BEGIN
             MarkSelection[w];
             fixup ← TRUE;
             END;
           SetIndex[w.file, index];
           ch ← w.file.get[w.file];
           StreamDefs.ClearDisplayChar[w.ds, ch];
           IF fixup THEN
             BEGIN
             w.selection.rightx ← w.ds.charx;
             index ← ModifyIndex[index, -1];
             w.selection.rightindex ← index;
             MarkSelection[w];
             fixup ← FALSE;
             END;
           END;
         END;
       controlW, controlQ =>
         BEGIN
         DO
         IF w.ds.charx = leftmargin THEN EXIT;
         index ← GetIndex[w.file];
```

```
        index ← ModifyIndex[index, -1];
        IF EqualIndex[w.selection.rightindex, index] THEN
          BEGIN
          MarkSelection[w];
          fixup ← TRUE;
          END;
        SetIndex[w.file, index];
        w.eofindex ← index;
        ch ← w.file.get[w.file
           ! StreamError => EXIT];
        IF ch = Space AND NOT firstchar THEN EXIT
          ELSE IF ch # Space THEN firstchar ← FALSE;
        StreamDefs.ClearDisplayChar[w.ds, ch];
        ENDLOOP;
        IF fixup THEN
          BEGIN
          index ← ModifyIndex[index, -1];
          w.selection.rightindex ← index;
          w.selection.rightx ← w.ds.charx;
          MarkSelection[w];
          fixup ← FALSE;
          END;
        END;
      ENDCASE =>
        MakeOrExtendSelection[w, char];
    END;

MakeOrExtendSelection: PROCEDURE[w: WindowHandle, char: CHARACTER] =
  BEGIN OPEN WMState;
  -- declare locals
  ds: DisplayHandle ← w.ds;
  sel: Selection;
  index: StreamIndex ← GetIndex[w.file];
  -- now make/extend the current selection
  IF NOT ds.charx = w.selection.rightx OR EqualIndex[originindex, index] THEN
    BEGIN  --make this char the current selection
    w.ds.put[w.ds, char];
    sel ← Selection[
      leftx: ds.charx - ComputeCharWidth[char, ds.pfont],
      leftline: ds.line,
      leftindex: index,
      rightx: ds.charx ,
      rightline: ds.line,
      rightindex: index
      ];
    END
  ELSE
    BEGIN -- extend it to include this char
    w.ds.put[w.ds, char];
    sel ← Selection[
      leftx: w.selection.leftx,
      leftline: w.selection.leftline,
      leftindex: w.selection.leftindex,
      rightx: ds.charx ,
      rightline: ds.line,
      rightindex: index
      ];
  END;
  MakeSelection[w, @sel];
  END;

SetJumpStripe: PUBLIC PROCEDURE[w: WindowHandle, flag: BOOLEAN] =
    BEGIN OPEN WMState;
    -- Declare Locals
      r: Rptr = w.rectangle;
      y: yCoord;
      bytepos, eof: InlineDefs.LongCARDINAL;
      barheight: INTEGER = r.ch-defaultlineheight-2;
      barwidth: INTEGER = leftmargin-1;
      currentindex: 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 OPEN DoubleDefs;
        IF currentcursor = botharrow THEN RETURN;
        SetCursor[botharrow];
        ButtonProcArray ← ScrollProcArray;
        ClearBoxInRectangle[r, 1, barwidth, defaultlineheight +1, barheight, grayarray];
        IF w.tempindex = nullindex
          THEN currentindex ← w.fileindex
          ELSE currentindex ← w.tempindex;
        -- compute position in file and paint(reset) position
        bytepos ← IF currentindex.byte=177777B THEN [0, 0]
          ELSE DAdd[InlineDefs.LongMult[currentindex.page, AltoDefs.BytesPerPage],
            LongCARDINAL[currentindex.byte, 0]];
        eof ← IF w.eofindex.byte=177777B THEN [1, 0]
          ELSE DAdd[InlineDefs.LongMult[w.eofindex.page, AltoDefs.BytesPerPage],
            LongCARDINAL[w.eofindex.byte, 0]];
        y ← DDivide[DMultiply[bytepos, LongCARDINAL[barheight, 0]], eof].quotient.lowbits;
        ClearBoxInRectangle[r, 3, barwidth-6, defaultlineheight +1, y, 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 WMState;
    defaultks.putback[defaultks, c];
    END;

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

  SetCursor: PUBLIC PROCEDURE[type: CursorType] =
    BEGIN OPEN WMState;
    cursorptr: POINTER;
    currentcursor ← type;
    cursorptr ← SELECT type FROM
      textpointer => BASE[CursorA],
      arrow => BASE[CursorB],
      bullseye => BASE[CursorC],
      leftbutton => BASE[CursorD],
      uparrow => BASE[CursorE],
      downarrow => BASE[CursorF],
```

```
      botharrow => BASE[CursorG],
      norm => BASE[CursorH],
      hourglass => BASE[CursorI],
      ENDCASE => cursormap;
    InlineDefs.COPY[from: cursorptr, to: cursormap, nwords: 16];
    END;

  CursorXAdjust:  PUBLIC PROCEDURE RETURNS[INTEGER] =
    BEGIN OPEN WMState;
    adjust:  INTEGER ← 0;
    SELECT currentcursor FROM
      arrow => adjust ← 15;
      bullseye, uparrow, downarrow => adjust ← 8;
      botharrow, textpointer, leftbutton, hourglass, norm => NULL;
      ENDCASE =>  SetCursor[norm];
    RETURN[adjust];
    END;

  CursorToRectangleCoords: PUBLIC PROCEDURE [rectangle: Rptr, x: xCoord, y: yCoord]
    RETURNS[xCoord, yCoord] =
    BEGIN OPEN WMState;
    -- refinements for sensitive points of each cursor
      x ← x + CursorXAdjust[];
      SELECT currentcursor FROM
        arrow, bullseye => y ← y + 8;
        downarrow => y ← y + 16;
        ENDCASE => NULL;
    -- convert cursor coordinates to window coordinates
      [x, y] ← CursorToRecCoords[rectangle, x, y];
      RETURN[x, y];
    END;

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

NoteNameError: PUBLIC PROCEDURE [w:WindowHandle, str:  STRING] =
    BEGIN OPEN WMState;
    -- Declare Locals
      i: INTEGER;
      scratchstr: STRING;
    -- convert window into scratch and tell bad name
      IF w.type # scratch THEN
        BEGIN
        [scratchstr, i] ← AssignScratchFile[];
        AlterWindowType[w, scratch, scratchstr];
        scratchfiles[i] ← w.file;
        SystemDefs.FreeHeapString[scratchstr];
        END;
    WriteMessageString[w, str];
    WriteMessageString[w, "FileNameError!"];
    END;

WriteMessageString: PUBLIC PROCEDURE [w:WindowHandle, str: STRING] =
    BEGIN
    -- Declare Locals
      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 WMState;
    -- Declare Locals
      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"];
          StringDefs.AppendChar[str, LOOPHOLE[i+zero,CHARACTER]];
          RETURN[str, i];
          END;
        ENDLOOP;
    END;

CursorInit:  PROCEDURE =
  BEGIN
  CursorA ←
    [100000B, 140000B, 160000B, 170000B, 174000B, 176000B, 177000B, 170000B, 154000B,
      114000B, 6000B, 6000B, 3000B, 3000B, 1400B,1400B];
  CursorB ←
    [40B, 60B, 70B, 74B, 177776B, 177777B, 177776B, 74B, 70B, 60B, 40B, 0, 0, 0, 0, 0];
  CursorC ←
    [3740B, 17770B, 60006B, 140003B, 140003B, 141703B, 141703B, 140603B, 60006B, 34034B,
      17770B, 3740B, 0, 0, 0, 0];
  CursorD ←
    [177740B, 100040B, 135240B, 135240B, 135240B, 135240B, 135240B, 100040B, 100040B,
      100040B, 100040B, 100040B, 100040B, 100040B, 100040B, 177740B];
  CursorE ←
    [600B, 1700B, 7760B, 37776B, 177777B, 7760B, 7760B, 7760B, 7760B, 7760B, 7760B,7760B,
      7760B, 7760B,7760B, 7760B];
  CursorF ←
    [7760B, 7760B, 7760B,7760B, 7760B, 7760B,7760B, 7760B, 7760B,7760B, 7760B,177777B,
      37776B, 7760B, 1700B, 600B];
   CursorG ←
    [600B, 1700B, 7760B, 37776B, 177777B, 7760B, 7760B, 7760B, 7760B, 7760B, 7760B,
      177777B, 37776B, 7760B, 1700B, 600B];
  CursorH ←
    [37774B, 37774B, 34034B, 34034B, 34034B, 34034B, 34034B, 34034B, 34034B, 34034B,
      34034B, 34034B, 34034B, 34034B, 37774B, 37774B];
  CursorI ←
    [177777B, 100001B, 40002B, 34034B, 17170B, 7660B, 3740B, 1700B, 1100B, 2440B, 4220B,
      10610B, 21704B, 47762B, 177777B, 177777B];
  RETURN;
  END;

-- initialization for wmanager

  InitConfiguration: PROCEDURE =
    BEGIN OPEN WManagerDefs;
    WMState ← SystemDefs.AllocateHeapNode[SIZE[WMDataObject]];
    START WManSelection[WMState];
    START WManWindows[WMState];
    START WManPosition[WMState];
    -- Double is started by use of the START Trap
    END;

  InitManager: PROCEDURE =
    BEGIN OPEN WMState;
    -- Declare Locals
    i: INTEGER;
    w: WindowHandle ← 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
    CursorInit[];
    defaultmapdata ← GetDefaultBitmap[];
    defaultks ← GetDefaultKey[];
    [defaultfont, defaultlineheight] ← GetDefaultFont[];
    nullindex ← StreamIndex[0,-1];
    originindex ← StreamIndex[0, 0];
    currentcursor ← textpointer;
    -- setup External Button Procedures
    ButtonProcArray ← TextProcArray;
    StreamDefs.SetIdleProc[WindowManager];
    END;
```

```
-- MAIN BODY CODE

  InitConfiguration[];
  InitManager[];

END. of wmancontrol
```