```
-- WindowsA.Mesa  Edited by Sandman on May 12, 1978  4:48 PM

DIRECTORY
  BitBltDefs: FROM "bitbltdefs" USING [BBptr, BBTable, BITBLT],
  RectangleDefs: FROM "rectangledefs" USING [
    ClearBoxInRectangle, ComputeCharWidth, DrawBoxInRectangle, FAptr,
    FCDptr, GetDefaultFont, GrayArray, GrayPtr, InvertBoxInRectangle,
    leftmargin, RectangleError, WriteRectangleString],
  StreamDefs: FROM "streamdefs" USING [
    CleanupDiskStream, CloseDiskStream, DisplayHandle, FileLength, GetIndex,
    GrIndex, ModifyIndex, OpenDiskStream, OpenKeyStream, ScrollDisplay,
    SetDisplayLine, SetIndex, StreamError, StreamErrorCode, WriteDisplayChar],
  WindowDefs: FROM "windowdefs" USING [
    BMHandle, DisplayHandle, NullIndex, Rptr, StreamHandle, StreamIndex,
    UpdateSelection, WindowHandle, xCoord, yCoord];

DEFINITIONS FROM StreamDefs, RectangleDefs, WindowDefs;

WindowsA: PROGRAM
  IMPORTS RectangleDefs, StreamDefs, WindowDefs
  EXPORTS WindowDefs SHARES WindowDefs, StreamDefs =
  BEGIN

  currentwindow: WindowHandle ← NIL;
  defaultwindow: WindowHandle ← NIL;
  maxwindows: CARDINAL = 15;
  maxlines: CARDINAL = 80;
  linestarts: ARRAY [1..maxlines] OF StreamIndex;

  ControlA: CHARACTER = 1C;
  BS: CHARACTER = 10C;
  CR: CHARACTER = 15C;

-- mouse locations

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

-- Mesa Display Window Routines

  UnlinkDisplayWindow: PUBLIC PROCEDURE [w: WindowHandle] =
    BEGIN
    next: WindowHandle;
    -- check if only window
    IF w.link = w THEN currentwindow ← NIL
    ELSE
      BEGIN
      IF w = currentwindow THEN currentwindow ← w.link;
      next ← w;
      WHILE next.link # w DO next ← next.link; ENDLOOP;
      next.link ← w.link;
      END;
    w.link ← NIL;
    END;

  RepaintDisplayWindows: PUBLIC PROCEDURE [mapdata: BMHandle] =
    BEGIN
    -- declare locals
    i, j: INTEGER;
    w: WindowHandle;
    wa: ARRAY[0..maxwindows) OF WindowHandle;
    -- Build array of window handles
    i ← 0;
    IF (w ← currentwindow) = NIL THEN RETURN;
    DO
      wa[i] ← w;
      w ← w.link;
      i ← i + 1;
      IF w = currentwindow THEN EXIT
      ENDLOOP;
    -- now paint them in reverse order
    FOR j DECREASING IN [0..i) DO
      SetCurrentDisplayWindow[wa[j]];
      ENDLOOP;
```

```
    END;

  PaintDisplayWindow: PUBLIC PROCEDURE [w: WindowHandle] =
    BEGIN
    -- declare locals
    rectangle: Rptr = w.rectangle;
    clearwords: GrayArray ← [0, 0, 0, 0];
    clear: GrayPtr = @clearwords;
    -- first see if it's visible
    IF w.rectangle.visible = FALSE THEN RETURN;
    -- clear it and draw a box around it
    IF w = currentwindow THEN blinkOn ← FALSE;
    ClearBoxInRectangle[rectangle, 0, rectangle.cw, 0, rectangle.ch, clear];
    DrawDisplayWindow[w];
    -- do type dependent stuff
    IF w.ds # NIL THEN
      BEGIN
      SetDisplayLine[w.ds, 1, leftmargin];
      w.ds.chary ← w.ds.chary + 1;
      END;
    SELECT w.type FROM
      clear, -- window is simply cleared on activation
      random => -- USERS responsibility to repaint screen
        w.displayproc[w];  -- dispatch to procedure
      scratch,  -- data is maintained in scratch file
      scriptfile,       -- data is maintained 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
    -- declare locals
    pfont: FAptr;
    rectangle: Rptr = w.rectangle;
    x,y,lineheight: INTEGER;
    -- write box name
    [pfont, lineheight] ← GetDefaultFont[];
    IF w.name # NIL THEN
      [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 [WindowHandle, xCoord, yCoord] =
    -- This guy takes Cursor coordinates and tries to find
    -- the "top most" window for them and
    -- returns the x,y in window coords
    BEGIN
    -- define locals
    wptr: WindowHandle ← currentwindow;
    rectangle: Rptr;
    wx: xCoord;
    wy: yCoord;
    slop:  INTEGER ← 5;
    -- now check windows
    IF wptr = NIL THEN RETURN[NIL, 0, 0];
    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[wptr, wx, wy];
      wptr ← wptr.link;
```

```
      IF wptr = currentwindow THEN RETURN[NIL, 0, 0];
      slop ← 0;
      ENDLOOP;
    END;

  SetCurrentDisplayWindow: PUBLIC PROCEDURE [w: WindowHandle] =
    BEGIN
    -- define locals
    next: WindowHandle;
    -- check if window ring is empty
    IF currentwindow = NIL THEN
      BEGIN w.link ← w; DoDataSetup[w]; END
    ELSE
      IF w # currentwindow THEN
        BEGIN
        IF blinkOn THEN [] ← BlinkCursor[];
  -- unlink him if he is currently linked
        IF w.link # NIL THEN UnlinkDisplayWindow[w];
  -- push current guys data
        UndoDataSetup[currentwindow];
  -- now link him into window ring
        w.link ← currentwindow;
        next ← currentwindow;
        WHILE next.link # currentwindow DO
          next ← next.link;
          ENDLOOP;
        next.link ← w;
        DoDataSetup[w];
        END;
    -- make it current and repaint the data
    currentwindow ← w;
    PaintDisplayWindow[w];
    END;


-- Routines for maintaining Window Data

  DoDataSetup: PROCEDURE [w: WindowHandle] =
    BEGIN
    -- do everything to make this guy's data backup active
    SELECT w.type FROM
      clear => NULL; -- window is simply cleared on activation
      random => NULL; -- 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 => NULL; -- 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;
```

```
WriteWindowChar: PROCEDURE [stream: StreamHandle, char: UNSPECIFIED] =
  BEGIN
  -- define locals
  w: WindowHandle ← currentwindow;
  r: Rptr;
  index: StreamIndex;
  update:  BOOLEAN ← FALSE;
  -- make sure its a Display Stream
  WITH ds:stream SELECT FROM
    Display =>
      BEGIN
      IF blinkOn THEN [] ← BlinkCursor[];
      IF w.ds # @ds THEN w ← FindWindowWithStream[@ds];
      SELECT w.type FROM
        clear, -- window is simply cleared on activation
        random => -- USERS responsibility to repaint
          WriteDisplayChar[@ds, 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.ds.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.ds.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
            ControlA, 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[@ds, 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;

DisplayFileData: PROCEDURE [w: WindowHandle] =
  BEGIN
  -- NOTE: this routine wants to be super efficient!!
  -- should use port to write characters
  -- define locals
  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.ds.pfont;
  x: CARDINAL ← w.ds.charx;
  x0: CARDINAL = w.rectangle.x0;
  y: CARDINAL ← w.ds.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.ds.line] ← w.tempindex;
    SetIndex[w.file, w.tempindex];
    END
  ELSE
    BEGIN
    linestarts[w.ds.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;
  IF w.type = file THEN count ← 10000
  ELSE count ←
    (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
```

```
      w.ds.charx ← x - width;
      StreamDefs.ScrollDisplay[w.ds, char
        ! StreamError =>
          IF stream = w.ds THEN
            BEGIN
            IF error = StreamEnd
            AND w.ds.options.StopBottom THEN
              BEGIN
              linestarts[w.ds.line+1] ← ModifyIndex[index, nchars];
              fullwin ← TRUE;
              EXIT;
              END
            ELSE FixupOnOverflow[w, error, okay];
            RESUME;
            END];
      y ← w.ds.chary + w.rectangle.y0;
      x ← w.ds.charx;
      END
    ELSE IF x >= w.rectangle.cw THEN
      BEGIN
      w.ds.charx ← x - width;
      StreamDefs.ScrollDisplay[w.ds, char
        ! StreamError =>
          IF stream = w.ds THEN
            BEGIN
            IF error = StreamEnd
            AND w.ds.options.StopBottom THEN
              BEGIN
              linestarts[w.ds.line+1] ← ModifyIndex[index, nchars-1];
              fullwin ← TRUE;
              EXIT;
              END
            ELSE FixupOnOverflow[w, error, backup];
            RESUME;
            END];
      y ← w.ds.chary + w.rectangle.y0;
      x ← w.ds.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.ds.charx ← x;
  -- set remainder of line table null
  IF NOT fullwin THEN linestarts[w.ds.line +1] ← NullIndex;
  FOR i IN [w.ds.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
    IF index = backup THEN
      linestarts[w.ds.line] ← ModifyIndex[GetIndex[w.file], -1]
    ELSE linestarts[w.ds.line] ← GetIndex[w.file];
    END;

  FindWindowWithStream: PROCEDURE [ds: DisplayHandle] RETURNS [WindowHandle] =
    BEGIN
    -- define locals
    w: WindowHandle ← currentwindow;
    -- run around window ring and find it
    IF w = NIL THEN RETURN[NIL];
    DO
      IF  w.ds = ds THEN RETURN[w];
      w ← w.link;
      IF w = currentwindow THEN EXIT;
      ENDLOOP;
    ERROR;  -- good enough for now
    END;

  blinkOn: BOOLEAN ← FALSE;

  BlinkCursor: PUBLIC PROCEDURE RETURNS [BOOLEAN] =
    BEGIN OPEN w: currentwindow;
    ds: StreamDefs.DisplayHandle;
    IF currentwindow = NIL OR (w.type # scratch AND w.type # scriptfile) OR
      w.file # NIL AND GetIndex[w.file] # w.eofindex THEN RETURN[blinkOn];
    ds ← w.ds;
    RectangleDefs.InvertBoxInRectangle[
      ds.rectangle, ds.charx+1, 2, ds.chary+1, ds.lineheight-2];
    RETURN[blinkOn ← ~blinkOn]
    END;

  END...  of Window
```