```
-- Windows.Mesa  Edited by Sandman on September 27, 1977  11:28 AM

DIRECTORY
  ControlDefs: FROM "controldefs",
  ImageDefs: FROM "imagedefs",
  InlineDefs: FROM "inlinedefs",
  MenuDefs:  FROM "menudefs",
  NovaOps: FROM "novaops",
  RectangleDefs: FROM "rectangledefs",
  SegmentDefs: FROM "segmentdefs",
  StreamDefs: FROM "streamdefs",
  StringDefs: FROM "stringdefs",
  SystemDefs: FROM "systemdefs",
  WindowDefs: FROM "windowdefs";

DEFINITIONS FROM MenuDefs,SystemDefs, SegmentDefs, StreamDefs, RectangleDefs, WindowDefs;

Windows: PROGRAM [dfn: STRING]
  IMPORTS ImageDefs, RectangleDefs, SegmentDefs, StreamDefs, StringDefs, SystemDefs,
    WindowDefs
  EXPORTS WindowDefs SHARES WindowDefs, StreamDefs =
  BEGIN


-- GLOBAL Data

  --dfn: STRING ← "foo.script";
  currentwindow: WindowHandle ← NIL;
  defaultwindow: WindowHandle ← NIL;
  maxwindows: CARDINAL = 15;
  maxlines: CARDINAL = 50;
  linestarts: ARRAY [1..maxlines] OF StreamIndex;
  originindex: StreamIndex = StreamIndex[0, 0];
  nullindex: StreamIndex = StreamIndex[0, -1];

  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

  CreateDisplayWindow: PUBLIC PROCEDURE
    [type: WindowType, rectangle: Rptr, ds: DisplayHandle, ks: StreamHandle, name: STRING]
    RETURNS[WindowHandle] =
    BEGIN
    -- declare locals
      w: WindowHandle;
    -- create window structure and init it
      w ← SystemDefs.AllocateHeapNode[SIZE[DisplayWindow]];
      w↑ ← DisplayWindow[NIL, type, NIL, NIL, NILProc, rectangle, ds, ks, NIL,,,,];
    -- initialize data refresh mechanism based upon window type
      AlterWindowType[w, type, name];
      SetCurrentDisplayWindow[w];
    RETURN[w];
    END;

  AlterWindowType: PUBLIC PROCEDURE [w: WindowHandle, type: WindowType, name: STRING]  =
    BEGIN
    -- first undo all stuff for current type
      SELECT w.type FROM
        clear => NULL; -- window is simply cleared on activation
        random => NULL; -- USERS responsibility to repaint screen
        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 => NULL; -- window is simply cleared on activation
        random => NULL; -- USERS responsibility to repaint screen
        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 = NIL AND name # NIL) OR w.name # name THEN
        BEGIN
        IF w.name # NIL THEN 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.options.NoteLineBreak ← TRUE;
        w.ds.options.NoteScrolling ← TRUE;
        w.ds.put ← WriteWindowChar;
        SELECT type FROM
          clear => NULL;
          random => NULL;
          scratch,
          scriptfile => w.ds.options.StopBottom ← FALSE;
          file => w.ds.options.StopBottom ← TRUE;
          ENDCASE;
        END;
    END;

  DestroyDisplayWindow: PUBLIC PROCEDURE [w: WindowHandle] =
    BEGIN -- clear it, unlink it deallocate record space and return
    -- define locals
      rectangle: Rptr = w.rectangle;
      clearwords: GrayArray ← [0, 0, 0, 0];
      clear: GrayPtr = @clearwords;
    -- clear it and free any storage
      ClearBoxInRectangle[rectangle, 0, rectangle.cw, 0, rectangle.ch, clear];
      IF w = currentwindow THEN
        BEGIN
        IF w = w.link THEN
          BEGIN
          currentwindow ← NIL;
          UndoDataSetup[w];
          END
        ELSE
          BEGIN
          SetCurrentDisplayWindow[w.link];
          END;
        END;
      UnlinkDisplayWindow[w];
      IF w.file # NIL THEN w.file.destroy[w.file];
      SystemDefs.FreeHeapNode[w];
      -- later!! must undo anything done to StreamObject
    END;

  UnlinkDisplayWindow: PUBLIC PROCEDURE [w: WindowHandle] =
    BEGIN
    -- define locals
      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;
    w ← currentwindow;
    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
    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
      BEGIN
      [x,y] ← WriteRectangleString[rectangle, 2, 1, w.name, pfont
        ! RectangleError =>
          SELECT error FROM
            NotVisible,
            RightOverflow,
            BottomOverflow => CONTINUE;
```

```
            ENDCASE
          ];
      END;
  -- 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, y: INTEGER]
  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
    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;

GetLineTable: PUBLIC PROCEDURE RETURNS[POINTER] =
  BEGIN
  RETURN[@linestarts];
  END;

GetCurrentDisplayWindow: PUBLIC PROCEDURE RETURNS [WindowHandle] =
  BEGIN
  RETURN[currentwindow];
  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
      -- 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

  OpenDisplayWindows: PUBLIC PROCEDURE =
    BEGIN OPEN StreamDefs;
    -- This guy should set up anything to do with display windows
    -- currently used: switching to/from the extenal debugger
      IF defaultwindow.type = scriptfile THEN
        OpenDiskStream[defaultwindow.file
          ! StreamError =>
            BEGIN  defaultwindow.eofindex ← GetIndex[defaultwindow.file];
            RESUME END
          ];
    -- ensure at end
      SetIndex[defaultwindow.file, defaultwindow.eofindex];
    END;

  CloseDisplayWindows: PUBLIC PROCEDURE =
    BEGIN
    -- define locals
      file: StreamHandle = defaultwindow.file;
    -- This guy cleans up anything to do with display windows
    -- currently used: switching to/from the extenal debugger
      IF file = NIL THEN RETURN;
      IF defaultwindow=currentwindow AND defaultwindow.tempindex=nullindex THEN
        defaultwindow.eofindex ← GetIndex[file]
      ELSE SetIndex[file, defaultwindow.eofindex];
      file.put[file,CR];
      THROUGH [0..9] DO file.put[file,'~] ENDLOOP;
      CloseDiskStream[file];
    END;

  SetFileForWindow: PUBLIC PROCEDURE [w: WindowHandle, filename: STRING]  =
    BEGIN
    SetFileHandleForWindow[w, NIL, filename];
    END;


  SetFileHandleForWindow: PUBLIC PROCEDURE [
    w: WindowHandle, fileh: FileHandle, filename: STRING]  =
    BEGIN
    -- define locals
      access: AccessOptions;
    -- do file type specific stuff
      SELECT w.type FROM
        scratch,          -- data is maintained in scratch file
        scriptfile =>     -- data is maintained in typescript file
          access ← Read+Write+Append;
        file =>           -- data is a window on 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
        w.file.destroy[w.file];
    -- 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 FreeHeapString[w.name];
        w.name ← 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: INTEGER]  =
  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;

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 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 ← GetIndex[w.file];
                    w.eofindex ← index;
                    IF  index.byte = 0 THEN
                      BEGIN
                      IF index.page # 0 THEN
                        BEGIN
                        index.page ← index.page -1;
                        index.byte ← 254;
                        END;
                      END
                    ELSE index.byte ← index.byte-1;
                    SetIndex[w.file, index];
                    END;
                  ENDCASE =>
                    BEGIN
                    index ← GetIndex[w.file];
                    w.file.put[w.file, char];
                    w.eofindex ← GetIndex[w.file];
```

```
                    WriteDisplayChar[@ds, char
                      ! StreamError =>
                        IF stream = w.ds THEN
                          BEGIN
                          IF error = StreamEnd THEN
                            BEGIN
                            -- update the selection
                            w.selection.leftline ← MAX[0, w.selection.leftline - 1];
                            w.selection.rightline ←
MAX[0, w.selection.rightline - 1];
                            END;
                          IF char # 15B THEN SetIndex[w.file, index];
                          FixupOnOverflow[w, error];
                          IF char # 15B THEN char ← w.file.get[w.file];
                          RESUME;
                          END
                      ];
                    END;
                END;
              file => NULL;       -- window on a file
              ENDCASE;
            END;
        ENDCASE => ERROR StreamError[stream, StreamType];
    END;

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

  NILProc: PROCEDURE [w: WindowHandle] =
    BEGIN
    -- Dummy Display procedure
    END;

  DisplayFileData: PROCEDURE [w: WindowHandle] =
    BEGIN
    -- NOTE: this routine wants to be super efficient!!
    -- should use port to write characters
    -- define locals
      i, count, width: INTEGER;
      fullwin:  BOOLEAN ← FALSE;
      index: StreamIndex;
      char: CHARACTER;
      pfont: FAptr = w.ds.pfont;
      x: INTEGER ← w.ds.charx;
      dba: INTEGER;
      wad: BMptr;
      wordsperline: INTEGER = w.rectangle.bitmap.wordsperline;
    -- check if really a file there
      IF w.file = NIL THEN RETURN;
    -- check if temporary positioning
      IF NOT EqualIndex[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
      index ← GetIndex[w.file];
      IF w.type = file THEN count ← 10000
      ELSE count ← (w.eofindex.page-index.page)*512 + (w.eofindex.byte-index.byte);
      [dba, wad] ← SetupforConvert[w.rectangle, w.ds.charx, w.ds.chary];
    -- fill the window with text
```

```
      WHILE count > 0 DO
        index ← GetIndex[w.file];
        char ← w.file.get[w.file
          ! StreamError =>
            BEGIN
            w.eofindex ← GetIndex[w.file];
            EXIT;
            END
          ];
        width ← ComputeCharWidth[char, pfont];
        x ← x + width;
        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] ← index;
                  fullwin ← TRUE;
                  EXIT;
                  END
                ELSE FixupOnOverflow[w, error];
                RESUME;
                END
            ];
          x ← w.ds.charx;
          [dba, wad] ← SetupForConvert[w.rectangle, x, w.ds.chary];
          END
        ELSE IF x >= w.rectangle.cw THEN
          BEGIN
          w.ds.charx ← x - width;
          SetIndex[w.file, index];
          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] ← index;
                  fullwin ← TRUE;
                  EXIT;
                  END
                ELSE FixupOnOverflow[w, error];
                RESUME;
                END
              ];
          char ← w.file.get[w.file];
          x ← w.ds.charx;
          [dba, wad] ← SetupForConvert[w.rectangle, x, w.ds.chary];
          END
        ELSE
          [width, dba, wad] ← CONVERT[char, pfont, wad,
                wordsperline,dba];
        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 [LOOPHOLE[w.ds.line+2, INTEGER]..LOOPHOLE[maxlines, INTEGER]) DO
        linestarts[i] ← nullindex;
        ENDLOOP;
    END;

  SetupForConvert: PROCEDURE [rectangle: Rptr, x: xCoord, y: yCoord]
    RETURNS[INTEGER, BMptr] =
    BEGIN
    -- define locals
      ywordoffset: INTEGER;
      wad: BMptr;
      dba: INTEGER;
      xoffset: xCoord;
```

```
      wordsperline: INTEGER = rectangle.bitmap.wordsperline;
    -- compute DBA and WAD
      xoffset ← rectangle.x0 + x;
      ywordoffset ← (rectangle.y0 + y - 1) * wordsperline;
      dba ← InlineDefs.BITAND[InlineDefs.BITNOT[xoffset], 17B];
      wad ← rectangle.bitmap.addr+(xoffset/16)+ywordoffset;
    RETURN[dba, wad]
    END;

  FixupOnOverflow: PROCEDURE [w: WindowHandle, error: StreamErrorCode] =
    -- 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.ds.line] ← GetIndex[w.file];
    END;


-- Mesa Display Window Initialization Routine

  setupdefaultwindow: PROCEDURE =
    BEGIN
    -- Smokey asked me to say this is awful and ugly (JDW)
    i: CARDINAL;
    defaultwindow.file ← CreateByteStream[preopen.file,Read+Write+Append];
    defaultwindow.type ← scriptfile;
    defaultwindow.ds.options.StopBottom ← FALSE;
    defaultwindow.tempindex ← nullindex;
    defaultwindow.displayproc ← DisplayFileData;
    defaultwindow.fileindex ← defaultwindow.eofindex ← originindex;
    FOR i IN [1..maxlines] DO
      linestarts[i] ← nullindex;
      ENDLOOP;
    END;

  initwindows: PROCEDURE =
    BEGIN
      ds: DisplayHandle;
    -- create the default window
      ds ← GetDefaultDisplayStream[];
      defaultwindow ← CreateDisplayWindow [
        clear, ds.rectangle, ds, GetDefaultKey[], dfn];
      setupdefaultwindow[];
    END;

  CleanupItem: ImageDefs.CleanupItem ← ImageDefs.CleanupItem[, CleanupWindows];

  CleanupWindows: ImageDefs.CleanupProcedure =
    BEGIN
    SELECT why FROM
      Finish, Abort, Save =>
        BEGIN
        IF defaultwindow.file = NIL THEN RETURN;
        IF defaultwindow.tempindex # nullindex THEN
          SetIndex[defaultwindow.file, defaultwindow.eofindex];
        StreamDefs.TruncateDiskStream[defaultwindow.file];
        defaultwindow.file ← NIL;
        IF why = Save THEN
          BEGIN
          preopen.file ← NIL;
          preopen.name ← defaultwindow.name;
          ImageDefs.AddFileRequest[@preopen];
          END;
        END;
```

```
      OutLd => CloseDisplayWindows[];
      InLd => OpenDisplayWindows[];
      Restore =>
        BEGIN
        IF preopen.file = NIL THEN
          preopen.file ← NewFile[defaultwindow.name,Read+Write+Append,DefaultVersion];
        setupdefaultwindow[];
        SetCurrentDisplayWindow[defaultwindow];
        END;
      ENDCASE => ERROR;
    END;

-- MAIN BODY CODE

  preopen: short ImageDefs.FileRequest ← ImageDefs.FileRequest [
    file: NIL, access: Read+Write+Append, link:, body: short[fill:, name: dfn]];

  IF (REGISTER[ControlDefs.SDreg]+ControlDefs.sAddFileRequest)↑ # 0 THEN
    BEGIN
    ImageDefs.AddFileRequest[@preopen];
    STOP;
    END;

  IF preopen.file = NIL THEN
    preopen.file ← NewFile[dfn,Read+Write+Append,DefaultVersion];
  initwindows[];
  ImageDefs.AddCleanupProcedure[@CleanupItem];

  END. of Window
```