

```
-- File: display.mesa Edited by: Johnsson, September 27, 1977 11:23 AM
```

```
DIRECTORY
```

```
SystemDefs: FROM "systemdefs",
StreamDefs: FROM "streamdefs",
RectangleDefs: FROM "rectangledefs";
```

```
DEFINITIONS FROM StreamDefs, RectangleDefs;
```

```
Display: PROGRAM
```

```
IMPORTS RectangleDefs, StreamDefs, SystemDefs EXPORTS StreamDefs SHARES StreamDefs =
BEGIN
```

```
-- CHARACTER constants
```

```
NUL: CHARACTER = 0C;
TAB: CHARACTER = 11C;
CR: CHARACTER = 15C;
BackSpace: CHARACTER = 10C;
ControlA: CHARACTER = 1C;
Space: CHARACTER = 40C;
MaxCharCode: CHARACTER = 377C;
```

```
-- GLOBAL PUBLIC Data
```

```
defaultdisplaystream: PUBLIC DisplayHandle ← NIL;
displaystreams: DisplayHandle ← NIL;
```

```
-- Mesa Display Stream Routines
```

```
CreateDisplayStream: PUBLIC PROCEDURE [rectangle: Rptr] RETURNS[DisplayHandle] =
BEGIN
  -- declare locals
  ds: DisplayHandle;
  -- now create stream structure and init it
  ds ← SystemDefs.AllocateHeapNode[SIZE[Display StreamObject]];
  ds↑ ← StreamObject[ResetDisplayStream, GetNOP, PutbackNOP, WriteDisplayChar,
  EndofNOP, DestroyDisplayStream, Display[.....]];
  ds.rectangle ← rectangle;
  [ds.pfont, ds.lineheight] ← GetDefaultFont[];
  ds.options ← DSOptions[FALSE, FALSE, FALSE, FALSE];
  -- link into list of display streams
  ds.link ← displaystreams;
  displaystreams ← ds;
  -- set options in rectangle to tell about overflow(s)
  rectangle.options.NoteOverflow ← TRUE;
  -- and set line to Zero
  SetDisplayLine[ds, 0, leftmargin];
  RETURN[ds]
END;
```

```
DestroyDisplayStream: PROCEDURE [stream: StreamHandle] =
```

```
BEGIN
  -- define locals
  prev: DisplayHandle;
  WITH ds:stream SELECT FROM
  Display =>
  BEGIN
    -- delink from list of display streams
    IF @ds = displaystreams THEN displaystreams ← ds.link
    ELSE
      BEGIN
        prev ← displaystreams;
        UNTIL @ds = prev.link DO
          IF prev = NIL THEN ERROR;
          prev ← prev.link;
        FNDLOOP;
        prev.link ← ds.link
      FND;
    -- now free all storage
    SystemDefs.FreeHeapNode[@ds];
  FND;
  FNDCASE => ERROR StreamError[stream, StreamType];
FND;
```

```
GetDefaultDisplayStream: PUBLIC PROCEDURE RETURNS[DisplayHandle]=
BEGIN
```

```
RETURN[defaultdisplaystream];
END;
```

```
GetDisplayStreamList: PUBLIC PROCEDURE RETURNS[DisplayHandle]=
BEGIN
RETURN[displaystreams];
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 =>
[ds.charx, ds.chary] ← WriteRectangleChar[ds.rectangle, ds.charx, ds.chary, char, ds.pfont
! RectangleError =>
SELECT error FROM
BottomOverflow,
RightOverflow =>
BEGIN
ScrollDisplay[@ds, char];
CONTINUE;
END;
ENDCASE
];
END;
ENDCASE => ERROR StreamError[stream, StreamType];
END;
```

```
ClearCurrentLine: PUBLIC PROCEDURE [stream: StreamHandle]=
BEGIN
WITH ds:stream SELECT FROM
Display => ClearDisplayLine[@ds, ds.line];
ENDCASE => ERROR StreamError[stream, StreamType];
END;
```

```
ClearDisplayLine: PUBLIC PROCEDURE [stream: StreamHandle, line: CARDINAL] =
BEGIN
-- define locals
clearwords: GrayArray ← [0, 0, 0, 0];
clear: GrayPtr = @clearwords;
ds: DisplayHandle;
rectangle: Rptr;
lineheight: CARDINAL;
WITH s:stream SELECT FROM
Display => ds ← @s;
ENDCASE => ERROR StreamError[stream, StreamType];
rectangle ← ds.rectangle;
lineheight ← ds.lineheight;
-- now clear it
IF line > LOOPHOLE[rectangle.ch/LOOPHOLE[lineheight, INTEGER], CARDINAL] THEN RETURN;
ClearBoxInRectangle[rectangle, 1, rectangle.cw-2, lineheight*line, lineheight, clear];
-- reset line position to left margin
SetDisplayline[ds, line, leftmargin];
END;
```

```
ClearDisplayChar: PUBLIC PROCEDURE [stream: StreamHandle, char: CHARACTER] =
-- Assumptions:
-- erases last character written, iff position not changed
BEGIN
-- define and setup locals
cwidth: CARDINAL;
ds: DisplayHandle;
rectangle: Rptr;
clearwords: GrayArray ← [0, 0, 0, 0];
clear: GrayPtr = @clearwords;
-- check for control chars
WITH s:stream SELECT FROM
Display => ds ← @s;
```

```

    ENDCASE => ERROR StreamError[stream,StreamType];
rectangle ← ds.rectangle;
SELECT char FROM
NUL, CR, >MaxCharCode => RETURN;
ControlA, BackSpace => NULL;
TAB =>
    BEGIN
        IF ds.TABindex > 0 THEN
            BEGIN ds.TABindex ← ds.TABindex-1; ds.charx ← ds.TABs[ds.TABindex] END;
            ds.put[ds, 1B];
            RETURN
        END;
    < 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, ds.pfont];
    ds.charx ← MAX[leftmargin, LOOPHOLE[ds.charx-cwidth, INTEGER]];
    -- now clear it
    ClearBoxInRectangle[rectangle, ds.charx, cwidth, ds.chary, ds.lineheight, clear];
    ds.put[ds, 1B];
END;

SetDisplayLine: PUBLIC PROCEDURE [ds: DisplayHandle, line, pos: CARDINAL] =
BEGIN
    -- define locals
    lineheight: INTEGER = ds.lineheight;
    -- make sure line no. is in range and set character x and y
    line ← MIN[line, LOOPHOLE[(ds.rectangle.ch/lineheight)-1,CARDINAL]];
    ds.charx ← pos; ds.TABindex ← 0;
    ds.chary ← line*lineheight;
    ds.line ← line;
END;

ScrollDisplay: PUBLIC PROCEDURE [ds: DisplayHandle, char: UNSPECIFIED] =
BEGIN
    -- define locals
    newx, tw: CARDINAL; -- for TABs
    lastline: INTEGER;
    lineheight: INTEGER = ds.lineheight;
    blockheight: INTEGER;
    rectangle: Rptr = ds.rectangle;
    mapaddr: BPtr = rectangle.bitmap.addr;
    wordsperline: INTEGER = rectangle.bitmap.wordsperline;
    -- deal with character
    SELECT char FROM
        15B => NULL; -- <Carriage Return>
        0, 12B => RETURN; -- null character
        11B => -- TAB
            BEGIN
                ds.TABs[ds.TABindex] ← ds.charx;
                ds.TABindex ← ds.TABindex + 1;
                tw ← ComputeCharWidth[' ',ds.pfont] * 8;
                newx ← (ds.charx/tw+1)*tw;
                IF newx < rectangle.cw THEN
                    BEGIN ds.charx ← newx; RETURN END;
                IF tw >= rectangle.cw THEN char ← ' ';
                END;
            ENDCASE =>
            BEGIN
                IF ds.options.StopRight THEN RETURN;
                IF char < 40B THEN
                    BEGIN
                        WriteDisplayChar[ds, '↑'];
                        WriteDisplayChar[ds, char+100B];
                        RETURN;
                    END;
                END;
            END;
    -- check if at bottom of window
    lastline ← ((rectangle.ch-1)/lineheight)-1;
    IF ds.line = lastline THEN
        BEGIN

```

```
-- if he wants to be notified at the bottom, do it
  IF ds.options.NoteScrolling THEN
    SIGNAL StreamError[ds, StreamEnd];
  IF ds.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, ds.line];
END
ELSE
BEGIN
-- assumes next window line is cleared
  ds.line ← ds.line+1;
  SetDisplayLine[ds, ds.line, leftmargin];
  IF ds.options.NoteLineBreak THEN
    SIGNAL StreamError[ds, StreamPosition];
  END;
-- now write the char
  IF char NOT= 15B THEN WriteDisplayChar[ds, char];
END;

-- NOP routines

ResetDisplayStream: PROCEDURE [stream: StreamHandle]=
BEGIN
RETURN;
END;

GetNOP: PROCEDURE [stream: StreamHandle] RETURNS [UNSPECIFIED] =
BEGIN
RETURN[0];
END;

PutbackNOP: PROCEDURE [stream: StreamHandle, char: UNSPECIFIED] =
BEGIN
RETURN;
END;

EndofNOP: PROCEDURE [stream: StreamHandle] RETURNS [BOOLEAN] =
BEGIN
RETURN[FALSE];
END;

-- Alto Display Window Initialization Routine

initdisplaystreams: PROCEDURE =
BEGIN
mapdata: BMHandle = GetDefaultBitmap[];
defaultdisplaystream ←
  CreateDisplayStream[CreateRectangle[mapdata, 0, mapdata.width, 0, mapdata.height]];
END;

-- MAIN BODY CODE

initdisplaystreams[];

END. of Display
```