```
-- SystemDisplay.mesa; edited by Sandman; Aug 30, 1978 9:30 AM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [PageSize, wordlength],
  BitBltDefs: FROM "bitbltdefs" USING [BBptr, BBTable, BITBLT],
  DisplayDefs: FROM "displaydefs" USING [Background],
  FontDefs: FROM "fontdefs" USING [BitmapState, FontHandle],
  ImageDefs: FROM "imagedefs" USING [
    AddCleanupProcedure, CleanupItem, CleanupMask, CleanupProcedure],
  InlineDefs: FROM "inlinedefs" USING [BITSHIFT, BITXOR, COPY],
  SegmentDefs: FROM "segmentdefs" USING [
    DataSegmentAddress, DataSegmentHandle, DefaultBase, DeleteDataSegment,
    NewDataSegment],
  StreamDefs: FROM "streamdefs" USING [
    DiskHandle, DisplayHandle, GetIndex, ModifyIndex, SetIndex, StreamError,
    StreamHandle, StreamIndex, StreamObject];

SystemDisplay: PROGRAM
  IMPORTS ImageDefs, SegmentDefs, StreamDefs
  EXPORTS DisplayDefs, StreamDefs
  SHARES StreamDefs =

  BEGIN

  BitsPerWord: CARDINAL = AltoDefs.wordlength;
  StreamHandle: TYPE = StreamDefs.StreamHandle;
  OrderedPOINTER: TYPE = ORDERED POINTER;
  OrderedNIL: OrderedPOINTER = LOOPHOLE[NIL];

  TAB: CHARACTER = 11C;
  CR: CHARACTER = 15C;
  NUL: CHARACTER = 0C;
  SP: CHARACTER = ' ;

-- Display Hardware

  DCBchainHead: DCBptr = LOOPHOLE[420B];
  DCBnil: DCBptr = LOOPHOLE[0];
  DCBptr: TYPE = POINTER TO DCB;
  DCB: TYPE = RECORD [
    next: DCBptr,
    resolution: {high,low},
    background: DisplayDefs.Background,
    indenting: [0..77B],                -- in units of 16 bits
    width: [0..377B],           -- in units of 16 bits, must be even
    bitmap: OrderedPOINTER,             -- must be even
    height: CARDINAL];          -- in double scan lines

  LeftIndent: CARDINAL;
  LeftMargin: CARDINAL;
  RightMargin: CARDINAL;
  WordsPerLine: CARDINAL;

  DSP: Display StreamDefs.StreamObject ← StreamDefs.StreamObject [
    reset: ResetDS,
    get: GetNop,
    put: DPutChar,
    putback: PutbackNop,
    endof: EndofNop,
    destroy: DestroyNop,
    body: Display[,,,,,,,,]];

  systemDS: StreamDefs.DisplayHandle = @DSP;

  displayOn: BOOLEAN ← FALSE;
  bmSegment: SegmentDefs.DataSegmentHandle;
  bmFirst, bmTail, bmNext, bmLastLine: OrderedPOINTER;
  lastLineSize: CARDINAL;
  dummyDCB: DCBptr;
  firstDCB, lastDCB, currentDCB: DCBptr ← DCBnil;
  -- layout of bitmap is:
  --    DCBs: ARRAY [firstDCB..lastDCB] OF DCB,
  --    bmFirst: ARRAY OF UNSPECIFIED,
  --    bmLastLine: ARRAY [0..DSP.lineheight*WordsPerLine) OF UNSPECIFIED,
  -- bmNext points to next word to allocate,
  -- bmTail points to oldest allocated bitmap.
```

```
  bmState: FontDefs.BitmapState;
  tabWidth: CARDINAL;

  font: PUBLIC FontDefs.FontHandle ← NIL;

  -- Typescript data
  typescript: PUBLIC StreamDefs.DiskHandle ← NIL;
  startLine: StreamDefs.StreamIndex ← [0, 0];

  GetDefaultDisplayStream: PUBLIC PROCEDURE RETURNS[StreamDefs.DisplayHandle] =
    BEGIN
    RETURN[systemDS];
    END;

  NotEnoughSpaceForDisplay: PUBLIC SIGNAL = CODE;

  SetupBitmap: PROCEDURE [bitmap: POINTER, nLines, nWords: CARDINAL] =
    BEGIN
    dcb: DCBptr;
    lastLineSize ← DSP.lineheight*WordsPerLine;
    WHILE nLines*SIZE[DCB] + lastLineSize > nWords DO
      IF nLines > 1 THEN nLines ← nLines - 1
      ELSE
        BEGIN
        IF WordsPerLine = 2 THEN ERROR NotEnoughSpaceForDisplay;
        WordsPerLine ← WordsPerLine - 2;
        RightMargin ← WordsPerLine * BitsPerWord;
        lastLineSize ← DSP.lineheight*WordsPerLine;
        END;
      ENDLOOP;
    currentLines ← nLines;
    firstDCB ← dcb ← bitmap;
    THROUGH [0..nLines) DO
      dcb.next ← dcb + SIZE[DCB];
      dcb.height ← DSP.lineheight/2;
      dcb ← dcb.next;
      ENDLOOP;
    lastDCB ← dcb-SIZE[DCB];
    lastDCB.next ← DCBnil;
    bmFirst ← LOOPHOLE[dcb, OrderedPOINTER];
    bmLastLine ← LOOPHOLE[bitmap+nWords-lastLineSize, OrderedPOINTER];
    bmState ← [origin: bmLastLine, wordsPerLine: WordsPerLine, x:, y:0];
    END;

  currentPages, currentLines, currentDummySize: CARDINAL;

  DisplayOff: PUBLIC PROCEDURE [color: DisplayDefs.Background] =
    BEGIN
    IF ~displayOn THEN RETURN;
    SetSystemDisplaySize[0,0];
    font.close[font];
    dummyDCB.background ← color;
    dummyDCB.height ← 1;
    END;

  DisplayOn: PUBLIC PROCEDURE =
    BEGIN
    IF displayOn THEN RETURN;
    dummyDCB.background ← white;
    SetDummyDisplaySize[currentDummySize];
    SetSystemDisplaySize[currentLines, currentPages];
    END;

  SetSystemDisplaySize: PUBLIC PROCEDURE [nTextLines, nPages: CARDINAL] =
    BEGIN OPEN SegmentDefs;
    IF displayOn THEN
      BEGIN
      firstDCB.next ← lastDCB.next;
      RemoveDCB[firstDCB];
      firstDCB ← lastDCB ← currentDCB ← DCBnil;
      DeleteDataSegment[bmSegment];
      blinkOn ← displayOn ← FALSE;
      END;
    IF nPages = 0 THEN RETURN;
    currentPages ← nPages;  -- for Display Off/On
```

```
    currentLines ← nTextLines;  -- for Display Off/On
    bmSegment ← NewDataSegment[DefaultBase, nPages];
    SetupBitmap[DataSegmentAddress[bmSegment], nTextLines, nPages*AltoDefs.PageSize];
    displayOn ← TRUE;
    ClearDS[systemDS];
    InsertDCB[new: firstDCB, before: dummyDCB.next];
    RETURN
    END;

  SetSystemDisplayWidth: PUBLIC PROCEDURE [indent, width: CARDINAL] =
    BEGIN
    LeftMargin ← indent MOD BitsPerWord;
    LeftIndent ← indent / BitsPerWord;
    WordsPerLine ← Even[width/BitsPerWord];
    RightMargin ← WordsPerLine * BitsPerWord;
    IF displayOn THEN SetSystemDisplaySize[currentLines, currentPages];
    RETURN
    END;

  SetDummyDisplaySize: PUBLIC PROCEDURE [nScanLines: CARDINAL] =
    BEGIN
    currentDummySize ← nScanLines;  -- for Display Off/On
    IF nScanLines/2 = dummyDCB.height THEN RETURN;
    IF dummyDCB.height # 0 THEN RemoveDCB[dummyDCB];
    dummyDCB.height ← nScanLines/2;
    IF dummyDCB.height # 0 THEN InsertDCB[new: dummyDCB, before: firstDCB];
    RETURN
    END;

  ResetDS: PROCEDURE [stream: StreamHandle] =
    BEGIN
    ClearDS[stream];
    IF typescript # NIL THEN typescript.reset[typescript];
    RETURN
    END;

  ClearDS: PROCEDURE [stream: StreamHandle] =
    BEGIN
    dcb: DCBptr;
    IF stream # systemDS THEN
      SIGNAL StreamDefs.StreamError[stream,StreamType];
    IF ~displayOn THEN RETURN;
    FOR dcb ← firstDCB, dcb.next DO
      dcb.resolution ← high;
      dcb.background ← white;
      dcb.indenting ← dcb.width ← 0;
      dcb.bitmap ← bmLastLine;
      IF dcb = lastDCB THEN EXIT;
      ENDLOOP;
    bmNext ← bmFirst;
    bmTail ← bmLastLine;
    currentDCB ← firstDCB;
    ClearCurrentLine[stream];
    RETURN
    END;


  ClearCurrentLine: PUBLIC PROCEDURE [stream: StreamHandle] =
    BEGIN
    IF stream # systemDS THEN
      SIGNAL StreamDefs.StreamError[stream,StreamType];
    IF typescript # NIL THEN
      StreamDefs.SetIndex[typescript, startLine];
    IF ~displayOn THEN RETURN;
    bmLastLine↑ ← 0;
    InlineDefs.COPY[from: bmLastLine, to: bmLastLine+1, nwords: lastLineSize-1];
    currentDCB.indenting ← LeftIndent;
    currentDCB.bitmap ← bmLastLine;
    currentDCB.width ← WordsPerLine;
    bmState.x ← LeftMargin;
    blinkOn ← FALSE;
    DSP.TABindex ← 0;
    RETURN
    END;
```

```
Scroll: PROCEDURE [char: CHARACTER] =
  BEGIN OPEN BitBltDefs;
  bbt: ARRAY [0..SIZE[BBTable]] OF WORD;
  bbp: BBptr ← Even[BASE[bbt]];
  pos: CARDINAL;
  SELECT char FROM
    CR => NULL;
    TAB =>
      BEGIN
      DSP.TABs[DSP.TABindex] ← bmState.x;
      DSP.TABindex ← DSP.TABindex + 1;
      pos ← (LOOPHOLE[bmState.x-LeftMargin,CARDINAL]/tabWidth+1)*tabWidth+LeftMargin;
      IF pos < RightMargin THEN bmState.x ← pos
      ELSE DPutChar[systemDS, SP];
      RETURN
      END;
    NUL => RETURN;
    ENDCASE =>
      IF char < 40C THEN
        BEGIN
        DPutChar[systemDS, '↑];
        DPutChar[systemDS,
          LOOPHOLE[LOOPHOLE[char,CARDINAL]+100B,CHARACTER]];
        RETURN
        END;
  -- Do the scroll, assuming last (current) line is in bmLastLine.
  -- scroll all others by BLTing their DCBs.  move old last line to
  -- new bitmap and free bmLastLine for reuse.
  UNTIL Compact[currentDCB,bmState.x] DO
    IF ~DeleteTopLine[] THEN RETURN;-- not enough space
    ENDLOOP;
  IF currentDCB # lastDCB THEN currentDCB ← currentDCB.next
  ELSE
    BEGIN
    IF firstDCB.width # 0 THEN
      [] ← DeleteTopLine[];
    bbp↑ ← [
      pad: 0,
      sourcealt: FALSE, destalt: FALSE,
      sourcetype: block, function: replace,
      dbca: firstDCB, dbmr: SIZE[DCB],
      dlx: 16, dty: 0,
      dw: 16*(SIZE[DCB]-1), dh: currentLines-1,
      sbca: firstDCB, sbmr: SIZE[DCB],
      slx: 16, sty: 1,
      unused:, gray0:, gray1:, gray2:, gray3:];
    IF firstDCB # lastDCB THEN BITBLT[bbp];
    END;
  IF typescript # NIL THEN startLine ← StreamDefs.GetIndex[typescript];
  ClearCurrentLine[systemDS];
  IF char # CR THEN DPutChar[systemDS, char];
  END;


DeleteTopLine: PROCEDURE RETURNS [BOOLEAN] =
  BEGIN
  dcb: DCBptr;
  -- find first line with bitmap allocated
  FOR dcb ← firstDCB, dcb.next DO
    IF dcb.width # 0 THEN EXIT;
    IF dcb = lastDCB THEN RETURN[FALSE]; -- found no top line to delete
    ENDLOOP;
  dcb.width ← dcb.indenting ← 0;
  -- find next line with bitmap allocated
  UNTIL dcb = lastDCB DO
    dcb ← dcb.next;
    IF dcb.width # 0 THEN
      BEGIN bmTail ← dcb.bitmap; EXIT END;
    REPEAT FINISHED => -- all linex deleted
      BEGIN bmTail ← bmLastLine; bmNext ← bmFirst END;
    ENDLOOP;
  RETURN[TRUE];
  END;


Compact: PROCEDURE [dcb: DCBptr, x: CARDINAL] RETURNS [BOOLEAN] =
```

```
    BEGIN
    newWidth: CARDINAL ← (x+15)/16;
    oldWidth: CARDINAL ← dcb.width;
    old: OrderedPOINTER ← dcb.bitmap;
    lineHeight: CARDINAL ← dcb.height*2;
    d: CARDINAL;
    IF x <= LeftMargin THEN d ← 0
    ELSE
      FOR d IN [0..newWidth) DO
        BEGIN
        p: OrderedPOINTER;
        p ← old + d;
        THROUGH [0..lineHeight) DO
          IF p↑ # 0 THEN GO TO foundit;
          p ← p + oldWidth;
          ENDLOOP;
        END;
        REPEAT foundit => NULL;
        ENDLOOP;
    newWidth ← Even[newWidth-d];
    IF newWidth > 0 THEN
      BEGIN OPEN BitBltDefs;
      bbt: ARRAY [0..SIZE[BBTable]] OF WORD;
      bbp: BBptr ← Even[BASE[bbt]];
      new: OrderedPOINTER;
      IF (new ← GetMapSpace[newWidth*lineHeight]) = OrderedNIL THEN
        RETURN[FALSE];
      dcb.width ← 0;
      bbp↑ ← [
        pad: 0,
        sourcealt: FALSE, destalt: FALSE,
        sourcetype: block, function: replace,
        dbca: new, dbmr: newWidth,
        dlx: 0, dty: 0,
        dw: newWidth*16, dh: lineHeight,
        sbca: old, sbmr: oldWidth,
        slx: d*16, sty: 0,
        unused:, gray0:, gray1:, gray2:, gray3:];
      BITBLT[bbp];
      dcb.indenting ← LeftIndent + d;
      dcb.bitmap ← new;
      dcb.width ← newWidth;
      END
    ELSE dcb.indenting ← dcb.width ← 0;
    RETURN [ TRUE ];
    END;

  GetMapSpace: PROCEDURE [nwords: [0..77777B]] RETURNS [p: OrderedPOINTER] =
    BEGIN
    t: INTEGER;
    DO
      t ← bmTail - bmNext;
      IF t < 0 THEN
        IF bmLastLine >= bmNext+nwords THEN EXIT
        ELSE bmNext ← bmFirst
      ELSE
        IF t >= nwords THEN EXIT
        ELSE RETURN[OrderedNIL];
      ENDLOOP;
    p ← bmNext;
    bmNext ← bmNext + nwords;
    RETURN
    END;

  DPutChar: PROCEDURE [stream: StreamHandle, char: UNSPECIFIED] =
    BEGIN
    IF stream # systemDS THEN
      SIGNAL StreamDefs.StreamError[stream,StreamType];
    IF ~displayOn THEN RETURN;
    IF char > 377B THEN RETURN;
    IF blinkOn THEN [] ← BlinkCursor[];
    IF char < 40B OR
      font.charWidth[font, char] + bmState.x > RightMargin THEN Scroll[char]
    ELSE font.paintChar[font,char,@bmState];
    RETURN
    END;
```

```
DPutCharTS: PROCEDURE [stream: StreamHandle, char: UNSPECIFIED] =
  BEGIN
  DPutChar[stream, char];
  typescript.put[typescript, char];
  RETURN
  END;

ClearChar: PROCEDURE [stream: StreamHandle, char: UNSPECIFIED] =
  BEGIN
  IF stream # systemDS THEN
    SIGNAL StreamDefs.StreamError[stream,StreamType];
  IF displayOn THEN
    BEGIN
    IF blinkOn THEN [] ← BlinkCursor[];
    SELECT char FROM
      NUL, CR, > 377B => RETURN;
      TAB =>
        BEGIN
        IF DSP.TABindex > 0 THEN
          BEGIN
          DSP.TABindex ← DSP.TABindex -1;
          bmState.x ← DSP.TABs[DSP.TABindex];
          END;
        RETURN
        END;
      < 40B =>
        BEGIN
        ClearDisplayChar[stream, char+100B];
        char ← '↑;
        END;
      ENDCASE => NULL;
    font.clearChar[font,char,@bmState];
    END;
  RETURN
  END;

ClearDisplayChar: PUBLIC PROCEDURE [stream: StreamHandle, char: UNSPECIFIED] =
  BEGIN
  ClearChar[stream, char];
  IF typescript # NIL THEN
    BEGIN OPEN StreamDefs;
    SetIndex[typescript, ModifyIndex[GetIndex[typescript],-1]];
    END;
  RETURN
  END;

GetNop: PROCEDURE [stream: StreamHandle] RETURNS [UNSPECIFIED] =
  BEGIN
  ERROR StreamDefs.StreamError[stream,StreamAccess]
  END;

PutbackNop: PROCEDURE [stream: StreamHandle, char: UNSPECIFIED] =
  BEGIN
  ERROR StreamDefs.StreamError[stream,StreamAccess]
  END;

EndofNop: PROCEDURE [stream: StreamHandle] RETURNS [BOOLEAN] =
  BEGIN
  IF stream # systemDS THEN
    SIGNAL StreamDefs.StreamError[stream,StreamType];
  RETURN[FALSE]
  END;

DestroyNop: PROCEDURE [stream: StreamHandle] =
  BEGIN
  ERROR StreamDefs.StreamError[stream,StreamAccess]
  END;

ddarray: ARRAY [0..SIZE[DCB]] OF UNSPECIFIED;

InsertDCB: PROCEDURE [new: DCBptr, before: DCBptr] =
  BEGIN
  dcb: DCBptr;
  FOR dcb ← new, dcb.next DO
    IF dcb.next = DCBnil THEN
```

```
      BEGIN dcb.next ← before; EXIT END;
    ENDLOOP;
  FOR dcb ← DCBchainHead, dcb.next DO
    IF dcb.next = before THEN
      BEGIN dcb.next ← new; EXIT END;
    ENDLOOP;
  END;

RemoveDCB: PROCEDURE [dcb: DCBptr] =
  BEGIN
  prev: DCBptr;
  FOR prev ← DCBchainHead, prev.next UNTIL prev.next = DCBnil DO
    IF prev.next = dcb THEN
      BEGIN prev.next ← dcb.next; EXIT END;
    ENDLOOP;
  dcb.next ← DCBnil;
  END;

Even: PROCEDURE [a: UNSPECIFIED] RETURNS [UNSPECIFIED] =
  BEGIN RETURN[a + CARDINAL[a] MOD 2] END;

SetFont: PUBLIC PROCEDURE [f: FontDefs.FontHandle] =
  BEGIN OPEN SegmentDefs;
  font ← f;
  DSP.lineheight ← Even[font.charHeight[font,'A]];
  DSP.pfont ← LOOPHOLE[f];
  tabWidth ← font.charWidth[font, SP]*8;
  RETURN
  END;

SetTypeScript: PUBLIC PROCEDURE [ts: StreamDefs.DiskHandle] =
  BEGIN
  DSP.put ← IF (typescript ← ts) = NIL THEN DPutChar ELSE DPutCharTS;
  DSP.rectangle ← LOOPHOLE[ts];
  RETURN
  END;

blinkOn: BOOLEAN ← FALSE;

BlinkCursor: PUBLIC PROCEDURE RETURNS [BOOLEAN] =
  BEGIN OPEN InlineDefs;
  mask: WORD;
  p: POINTER;
  IF ~displayOn THEN RETURN[blinkOn];
  mask ← BITSHIFT[3, 14 - CARDINAL[bmState.x+1] MOD 16];
  p ← bmState.origin + (bmState.x+1)/16 + bmState.wordsPerLine;
  THROUGH [2..systemDS.lineheight) DO
    p↑ ← BITXOR[p↑, mask];
    IF mask = 1 THEN (p+1)↑ ← BITXOR[(p+1)↑, 100000B];
    p ← p + bmState.wordsPerLine;
    ENDLOOP;
  RETURN[blinkOn ← ~blinkOn]
  END;

InitDisplay: PUBLIC PROCEDURE [dummySize, textLines, nPages: CARDINAL, f: FontDefs.FontHandle] =
  BEGIN OPEN SegmentDefs;
  IF font # NIL THEN font.destroy[font];
  SetFont[f];
  SetDummyDisplaySize[dummySize];
  SetSystemDisplaySize[textLines, nPages];
  RETURN
  END;

CleanupDisplay: ImageDefs.CleanupItem ← [link:, proc: Cleanup,
  mask: ImageDefs.CleanupMask[Finish] + ImageDefs.CleanupMask[Abort]];

Cleanup: ImageDefs.CleanupProcedure =
  BEGIN
  SELECT why FROM
    Finish, Abort => DCBchainHead.next ← DCBnil;
    ENDCASE;
  END;

ImageDefs.AddCleanupProcedure[@CleanupDisplay];
dummyDCB ← Even[BASE[ddarray]];
dummyDCB↑ ← [DCBnil,high,white,0,0,LOOPHOLE[0,OrderedPOINTER],0];
```

```
  SetSystemDisplayWidth[16,36*16];
  END.
```