```
-- Rectangles.Mesa  Edited by Sandman on September 27, 1977  11:26 AM

DIRECTORY
  AltoDefs: FROM "altodefs",
  ControlDefs: FROM "controldefs",
  ImageDefs: FROM "imagedefs",
  InlineDefs: FROM "inlinedefs",
  IODefs: FROM "iodefs",
  MiscDefs: FROM "miscdefs",
  NovaOps: FROM "novaops",
  OsStaticDefs: FROM "osstaticdefs",
  SystemDefs: FROM "systemdefs",
  SegmentDefs: FROM "segmentdefs",
  StreamDefs: FROM "streamdefs",
  RectangleDefs: FROM "rectangledefs";

DEFINITIONS FROM InlineDefs, SystemDefs, SegmentDefs, StreamDefs, RectangleDefs;

Rectangles: PROGRAM[pagesformap, mapwordsperline: CARDINAL]
    IMPORTS ImageDefs, MiscDefs, SystemDefs, SegmentDefs, StreamDefs
    EXPORTS RectangleDefs SHARES RectangleDefs =
  BEGIN

-- CHARACTER constants
  CR: CHARACTER = IODefs.CR;
  Space: CHARACTER = IODefs.SP;
  DEL: CHARACTER = IODefs.DEL;

-- GLOBAL PUBLIC Data (all PUBLIC for initialization guy ??)

  savedfirstDCB: DCBptr ← NIL;
  tempDCB: UNSPECIFIED;
  bitmaps: PUBLIC BMHandle ← NIL;
  defaultmapdata: PUBLIC BMHandle ← NIL;
  defaultfont: PUBLIC Fptr ← NIL;        -- points to start of font
  defaultpfont: PUBLIC FAptr ← NIL;      -- points to self relative ptrs
  defaultfontsegment: FileSegmentHandle ← NIL;
  defaultlineheight: PUBLIC INTEGER; -- assuming all lines equal;
  defaultcharwidths: STRING ← [128]; -- should be byte ARRAY (later!!)

-- GLOBAL Data

  wordsinpage: INTEGER = AltoDefs.PageSize;
  bbtable: ARRAY [0..SIZE[BBTable]+1] OF WORD;
  bbptr: BBptr ← LOOPHOLE[@bbtable];

-- Bitmap Rectangle Routines

  CreateRectangle: PUBLIC PROCEDURE
    [bitmap: BMHandle, x0, width: xCoord, y0, height: yCoord] RETURNS[Rptr] =
    BEGIN
    -- define locals
      rectangle: Rptr;
    -- allocate rectangle object and init it
      rectangle ← AllocateHeapNode[SIZE[Rectangle]];
      rectangle↑ ← Rectangle[NIL, FALSE,, bitmap, x0, width, 0, y0, height, 0];
      rectangle.options ← ROptions[FALSE,FALSE];
    -- link it into the listof rectangles and fix it up
      rectangle.link ← bitmap.rectangles;
      bitmap.rectangles ← rectangle;
      FixupRectangle[rectangle];
      RETURN[rectangle];
    END;

  DestroyRectangle: PUBLIC PROCEDURE [rectangle: Rptr] =
    BEGIN
    -- define locals
      prev: Rptr;
      bitmap: BMHandle ← rectangle.bitmap;
    -- delink it from the list of rectangles
      IF bitmap.rectangles = rectangle THEN
        bitmap.rectangles ← rectangle.link
      ELSE
        BEGIN
        prev ← bitmap.rectangles;
        UNTIL rectangle = prev.link DO
```

```
        IF prev = NIL THEN ERROR;
        prev ← prev.link;
        ENDLOOP;
      prev.link ← rectangle.link;
      END;
  -- deallocate rectangle object
    FreeHeapNode[rectangle];
  END;

MoveRectangle: PUBLIC PROCEDURE [rectangle: Rptr, x: xCoord, y: yCoord] =
  BEGIN
  -- define locals
    oldx: INTEGER = rectangle.x0;
    oldw: INTEGER = rectangle.cw;
    oldy: INTEGER = rectangle.y0;
    oldh: INTEGER = rectangle.ch;
    mapaddr: BMptr = rectangle.bitmap.addr;
    wordsperline: INTEGER = rectangle.bitmap.wordsperline;
    dlx, dty, dh, dw: INTEGER;
  -- this is a NOP if not moved
    IF x = oldx AND y = oldy THEN RETURN;
  -- and update rectangle to reflect move
    rectangle.x0 ← x;
    rectangle.y0 ← y;
    FixupRectangle[rectangle];
    IF rectangle.visible = FALSE THEN RETURN;
  -- ok, now physically move it
    dw ← MIN[ rectangle.cw, oldw];
    dh ← MIN[ rectangle.ch, oldh];
    bbptr↑ ← BBTable[0, block, replace, 0, mapaddr, wordsperline, x, y, dw, dh, mapaddr, wordsperline
**, oldx, oldy, 0, 0, 0, 0];
    BitBlt[bbptr];
  -- now figure out what was left behind and clear it
    IF x # oldx THEN -- first check if moved in x
      BEGIN
      IF x > oldx THEN
        BEGIN
        dlx ← oldx;
        dw ← MIN[x-oldx, oldw];
        END
      ELSE
        BEGIN
        dlx ← MAX[oldx, x+MIN[rectangle.cw, oldw]];
        dw ← (oldx+oldw) - dlx;
        END;
      dty ← oldy;
      dh ← oldh;
      bbptr↑ ← BBTable[, gray, replace,,,, dlx, dty, dw, dh,,,,,,,,];
      BitBlt[bbptr];
      END;
    IF y # oldy THEN -- now see if moved in y
      BEGIN
      IF y > oldy THEN
        BEGIN
        dty ← oldy;
        dh ← MIN[y-oldy, oldh];
        END
      ELSE
        BEGIN
        dty ← MAX[oldy, y+MIN[ rectangle.ch, oldh]];
        dh ← (oldy+oldh)-dty;
        END;
      dlx ← oldx;
      dw ← oldw;
      bbptr↑ ← BBTable[, gray, replace,,,, dlx, dty, dw, dh,,,,,,,,];
      BitBlt[bbptr];
      END;
  END;

GrowRectangle: PUBLIC PROCEDURE [rectangle: Rptr, width: xCoord, height: yCoord] =
  BEGIN
  -- define locals
    mapaddr: BMptr = rectangle.bitmap.addr;
    wordsperline: INTEGER = rectangle.bitmap.wordsperline;
    clearwords: GrayArray ← [0, 0, 0, 0];
    graywords: GrayArray ← [125252B, 52525B, 125252B, 52525B];
```

```
    clear: GrayPtr = @clearwords;
    gray: GrayPtr = @graywords;
  -- if it did not change then ignore
    IF width = rectangle.width
    AND height = rectangle.height THEN RETURN;
  -- clear it, change it, and then paint it gray
    ClearBoxInRectangle[rectangle, 0, rectangle.cw, 0, rectangle.ch, clear];
    rectangle.width ← width;
    rectangle.height ← height;
    FixupRectangle[rectangle];
    ClearBoxInRectangle[rectangle, 0, rectangle.cw, 0, rectangle.ch, gray];
  RETURN;
  END;

ClearBoxInRectangle: PUBLIC PROCEDURE
  [rectangle: Rptr, x0, width: xCoord, y0, height: yCoord, gray: GrayPtr] =
  BEGIN
  -- declare locals
    mapaddr: BMptr ← rectangle.bitmap.addr;
    wordsperline: INTEGER = rectangle.bitmap.wordsperline;
    dlx: INTEGER ← rectangle.x0+x0;
    dty: INTEGER ← rectangle.y0+y0;
    dw: INTEGER ← MIN[rectangle.cw, width];
    dh: INTEGER ← MIN[rectangle.ch, height];
  -- construct a BITBLT table and clear it
    bbptr↑ ← BBTable[0, gray, replace, 0, mapaddr,
    wordsperline, dlx, dty, dw, dh, mapaddr, wordsperline,
    dlx, dty, gray↑[0], gray↑[1], gray↑[2], gray↑[3]];
    BitBlt[bbptr];
  END;

DrawBoxInRectangle: PUBLIC PROCEDURE
  [rectangle: Rptr, x0, width: xCoord, y0, height: yCoord] =
  BEGIN
  -- declare locals
    mapaddr: BMptr ← rectangle.bitmap.addr;
    wordsperline: INTEGER = rectangle.bitmap.wordsperline;
    dlx: INTEGER ← rectangle.x0+x0;
    dty: INTEGER ← rectangle.y0+y0;
    dw: INTEGER ← MIN[rectangle.cw, width];
    dh: INTEGER ← MIN[rectangle.ch, height];
  -- draw the top line;
    bbptr↑ ← BBTable[0, gray, replace, 0, mapaddr,
    wordsperline, dlx, dty, dw, 1, mapaddr, wordsperline,
    dlx, dty, -1, -1, -1, -1];
    BitBlt[bbptr];
  -- draw two sides;
    bbptr↑ ← BBTable[, gray,,,,,,, 1, dh,,,,,,,,];
    BitBlt[bbptr];
    bbptr↑ ← BBTable[,,,,,, dlx+dw-1, dty, 1, dh,,,,,,,,];
    BitBlt[bbptr];
  -- and the bottom
    bbptr↑ ← BBTable[,,,,,, dlx, dty+dh-1, dw, 1,,,,,,,,];
    BitBlt[bbptr];
  END;

InvertBoxInRectangle: PUBLIC PROCEDURE
  [rectangle: Rptr, x0, width: xCoord, y0, height: yCoord] =
  BEGIN
  -- declare locals
    mapaddr: BMptr ← rectangle.bitmap.addr;
    wordsperline: INTEGER = rectangle.bitmap.wordsperline;
    dlx: INTEGER ← rectangle.x0+x0;
    dty: INTEGER ← rectangle.y0+y0;
    dw: INTEGER ← MIN[rectangle.cw, width];
    dh: INTEGER ← MIN[rectangle.ch, height];
  -- invert it
    bbptr↑ ← BBTable[0, compliment, replace, 0, mapaddr,
    wordsperline, dlx, dty, dw, dh, mapaddr, wordsperline,
    dlx, dty,,,,];
    BitBlt[bbptr];
  END;

ScrollBoxInRectangle: PUBLIC PROCEDURE
  [rectangle: Rptr, x0, width: xCoord, y0, height: yCoord, incr: INTEGER] =
  BEGIN
```

```
    -- declare locals
      mapaddr: BMptr ← rectangle.bitmap.addr;
      wordsperline: CARDINAL = rectangle.bitmap.wordsperline;
      dlx: INTEGER ← rectangle.x0+x0;
      dw: INTEGER ← MIN[rectangle.cw, width];
      dh: INTEGER ← MIN[rectangle.ch, (height-incr)];
      dty: INTEGER;
      sty: INTEGER;
    -- decide which way to scroll
      IF incr > 0 THEN
        BEGIN
        dty ← rectangle.y0+y0;
        sty ← dty+incr;
        END
      ELSE
        BEGIN
        sty ← rectangle.y0+y0;
        dty ← sty+incr;
        END;
    -- move it all up/down the specified amount
      bbptr↑ ← BBTable[0, block, replace, 0, mapaddr,
      wordsperline, dlx, dty, dw, dh, mapaddr, wordsperline,
      dlx, sty,,,,];
      BitBlt[bbptr];
    END;

  FixupRectangle: PROCEDURE[rectangle: Rptr] =
    BEGIN
    -- check if all ok first
      IF rectangle.bitmap = NIL
        OR rectangle.bitmap.addr = NIL THEN
          BEGIN
          rectangle.visible ← FALSE;
          RETURN;
          END;
    -- check for clipping on the right
      IF rectangle.x0+rectangle.width > rectangle.bitmap.width THEN
        rectangle.cw ← MAX[0, rectangle.bitmap.width-rectangle.x0]
      ELSE
        rectangle.cw ← rectangle.width;
    -- check for clipping on the bottom
      IF rectangle.y0+rectangle.height > rectangle.bitmap.height THEN
        rectangle.ch ← MAX[0, rectangle.bitmap.height-rectangle.y0]
      ELSE
        rectangle.ch ← rectangle.height;
    -- now check if visible
      IF rectangle.x0+minwidth > rectangle.bitmap.width OR
        rectangle.y0+minheight > rectangle.bitmap.height THEN
          rectangle.visible ← FALSE
      ELSE
        rectangle.visible ← TRUE;
    END;

  WriteRectangleChar: PUBLIC PROCEDURE
    [rectangle: Rptr, x: xCoord, y: yCoord, char: CHARACTER, pfont: FAptr]
    RETURNS[xCoord, yCoord] =
    -- Note: funny ywordoffset due to use of CONVERT!!!
    BEGIN
    -- define locals and init them
      dba: INTEGER;
      wad: BMptr;
      cwidth:  xCoord;
      wordsperline: INTEGER = rectangle.bitmap.wordsperline;
      xoffset: xCoord;
      ywordoffset: INTEGER = (rectangle.y0 + y - 1)*wordsperline;
      -- following is awful, undo later   signed: Smokey
      lineheight: INTEGER = LOOPHOLE[(pfont-SIZE[FontHeader])↑[0]];
    -- compute (or get char width)
      IF pfont = defaultpfont  AND char <= DEL THEN
        cwidth ← LOOPHOLE[defaultcharwidths[LOOPHOLE[char,CARDINAL]],xCoord]
      ELSE
        cwidth ← ComputeCharWidth[char, pfont];
    -- check for rectangle is visible and overflow
      IF rectangle.visible = FALSE THEN
        IF rectangle.options.NoteInvisible THEN
          SIGNAL RectangleError[rectangle, NotVisible]
```

```
        ELSE RETURN[x, y];
      IF y+lineheight >= rectangle.ch THEN
        IF rectangle.options.NoteOverflow THEN
          SIGNAL RectangleError[rectangle,BottomOverflow]
        ELSE RETURN[x, y];
      IF x+cwidth > rectangle.cw THEN
        IF rectangle.options.NoteOverflow THEN
          SIGNAL RectangleError[rectangle, RightOverflow]
        ELSE RETURN[x, y];
    -- compute some more stuff
      xoffset ← rectangle.x0 + x;
      dba ← BITAND[BITNOT[xoffset], 17B];
      wad ← rectangle.bitmap.addr+(xoffset/16)+ywordoffset;
    -- do scan convert
      [cwidth, dba, wad] ← CONVERT[char, pfont, wad, wordsperline,dba];
      RETURN[x+cwidth, y];
    END;

  WriteRectangleString: PUBLIC PROCEDURE
    [rectangle: Rptr, x: xCoord, y: yCoord, str: STRING, pfont: FAptr]
    RETURNS[xCoord, yCoord] =
    BEGIN
    -- define locals
      i: INTEGER;
      length: INTEGER = str.length;
    -- for now call write character  (make faster later!)
      FOR i IN [0..length) DO
        [x, y] ← WriteRectangleChar[rectangle, x, y, str[i], pfont];
        ENDLOOP;
    RETURN[x,y]
    END;

  RectangleToMapCoords: PUBLIC PROCEDURE [rectangle: Rptr, x: xCoord, y: yCoord]
    RETURNS[mapx: xCoord, mapy: yCoord] =
    BEGIN
    -- compute it
      mapx ← rectangle.x0 + MAX[0, MIN[rectangle.cw, x]];
      mapy ← rectangle.y0 + MAX[0, MIN[rectangle.ch, y]];
    RETURN[mapx, mapy]
    END;

  RectangleError: PUBLIC SIGNAL [rectangle: Rptr, error: RectangleErrorCode] = CODE;

-- Bitmap Routines

  GetDefaultBitmap: PUBLIC PROCEDURE RETURNS [BMHandle] =
    BEGIN
    RETURN[defaultmapdata];
    END;

  EVEN: PROCEDURE[v: UNSPECIFIED] RETURNS [UNSPECIFIED] =
    BEGIN
    -- make an even value by rounding v up
    RETURN[v+BITAND[v, 1]];
    END;

  CreateBitmap: PUBLIC PROCEDURE [pagesformap, wordsperline: CARDINAL] RETURNS[BMHandle] =
    BEGIN
    -- define locals
      mapdata: BMHandle;
      dcb: DCBptr;
    -- now allocate bitmap data records and init it
      mapdata ← AllocateHeapNode[SIZE[BitmapObject]];
      mapdata↑ ← BitmapObject[NIL, NIL, NIL, NIL, 0, 0, 0, 0, 0, 0, 0,  high, white];
    -- allocate a dcb for this guy
      -- NOTE: lots'a funnies because DCB's must be even
      -- and someone has to deallocate him eventually!!)
      dcb ← EVEN[mapdata.dcb ← AllocateHeapNode[SIZE[DCB]+1]];
      dcb.next ← DCBnil;
      ReallocateBitmap[mapdata, pagesformap, wordsperline];
    -- put him in the list of all bitmaps
      mapdata.link ← bitmaps;
      bitmaps ← mapdata;
      RETURN[mapdata];
    END;
```

```
DestroyBitmap: PUBLIC PROCEDURE[mapdata: BMHandle] RETURNS [POINTER] =
  BEGIN
  -- define locals
    addr: POINTER;
    prev: BMHandle;
  -- check to see if all Rectangles are gone
    IF mapdata.rectangles # NIL THEN
      SIGNAL BitmapError[mapdata, BitmapOperation];
  -- now actually destroy it
    IF mapdata.addr # NIL THEN FreePages[mapdata.addr];
    IF mapdata.dcb # NIL THEN FreeHeapNode[mapdata.dcb];
   addr ← mapdata.addr;
  -- take it out of the list of bitmaps
    IF mapdata = bitmaps THEN
      bitmaps ←mapdata.link
    ELSE
      BEGIN
      prev ← bitmaps;
      UNTIL mapdata = prev.link DO
        IF prev = NIL THEN ERROR;
        prev ← prev.link;
        ENDLOOP;
      prev.link ← mapdata.link;
      END;
    FreeHeapNode[mapdata];
    RETURN[addr];
  END;

UpdateBitmap: PUBLIC PROCEDURE [mapdata: BMHandle] RETURNS [DCBptr] =
  BEGIN
  -- reflects any changes in the bitmap object in the hardware
  -- define locals
    dcb: DCBptr = EVEN[mapdata.dcb];
  -- now fix up the DCB
    dcb.bitmap ← mapdata.addr;
    dcb.height ← mapdata.height/2;
    dcb.width ← mapdata.wordsperline;
    dcb.indenting  ← mapdata.indenting;
    dcb.resolution  ← mapdata.resolution;
    dcb.background  ← mapdata.background;
    RETURN[dcb];
  END;

ReallocateBitmap: PUBLIC PROCEDURE
  [mapdata: BMHandle, pagesformap, wordsperline: CARDINAL] =
  BEGIN
  -- physically alters a display bitmap
  -- define locals
    map: BMptr ← mapdata.addr;
    rectangle: Rptr;
    wordsformap: CARDINAL = pagesformap*AltoDefs.PageSize;
  -- check if need to discard old one
    IF mapdata.addr # NIL AND wordsformap # mapdata.words THEN
      BEGIN
      mapdata.dcb.width ← 0; -- ensure no trash on screen
      FreePages[mapdata.addr];
      map ← NIL;
      END;
  -- now setup and clear the new map
    IF pagesformap # 0 THEN
      BEGIN
      -- NOTE: assumes pages allocated on EVEN word boundary
      IF map = NIL THEN
        map ← AllocatePages[pagesformap];
      MiscDefs.Zero[map, wordsformap];
      mapdata.addr ← map;
      mapdata.words ← wordsformap;
      mapdata.wordsperline ← wordsperline;
      mapdata.width ← wordsperline*16;
      mapdata.height ← wordsformap/wordsperline;
      IF BITAND[mapdata.height, 1] = 1 THEN
        mapdata.height ← mapdata.height-1;
      [] ← UpdateBitmap[mapdata];
      END
    ELSE
      BEGIN
```

```
        mapdata.addr ← NIL;
        mapdata.width ← 0;
        mapdata.height ← 0;
        END;
    -- now go setup all the streams for this map
      rectangle ← mapdata.rectangles;
      UNTIL rectangle = NIL DO
        FixupRectangle[rectangle];
        rectangle ← rectangle.link
        ENDLOOP;
    END;

  DisplayBitmap: PUBLIC PROCEDURE [mapdata: BMHandle] =
    BEGIN
    -- links a bitmap into the displaychain using a BitmapObject
    -- Assumes Bitmap Record is correct
      -- eg: map is even word alligned etc...
    -- fills in the x0,y0 fields in the bitmap record too!
    -- define locals
      dcb, nextdcb: DCBptr;
    -- ensure DCB is correct
      dcb ← UpdateBitmap[mapdata];
    -- now link into display
      nextdcb ← DCBchainHead.next;
      mapdata.y0 ← 0;
      IF nextdcb # DCBnil THEN
        BEGIN
        WHILE nextdcb.next # DCBnil DO
          mapdata.y0 ← mapdata.y0 + (nextdcb.height)*2;
          nextdcb ← nextdcb.next;
          ENDLOOP;
        mapdata.y0 ← mapdata.y0 + (nextdcb.height)*2;
        END;
      nextdcb.next ← dcb;
      mapdata.x0 ← mapdata.indenting*16;
    END;

  UnDisplayBitmap: PUBLIC PROCEDURE[mapdata: BMHandle] =
    BEGIN
    -- nop for now
    END;

  CursorToMapCoords: PUBLIC PROCEDURE [mapdata: BMHandle, x: xCoord, y: yCoord]
    RETURNS[mapx: xCoord, mapy: yCoord] =
    BEGIN
    -- NOTE!! if bitmap ptr not supplied then use system default...
    IF mapdata = NIL THEN
      mapdata ← defaultmapdata;
    -- compute it
      mapx ← MAX[0, MIN[mapdata.width, x - mapdata.x0]];
      mapy ← MAX[0, MIN[mapdata.height, y - mapdata.y0]];
    RETURN[mapx, mapy]
    END;

CursorToRecCoords: PUBLIC PROCEDURE [rectangle: Rptr, x: xCoord, y: yCoord]
  RETURNS[xCoord, yCoord] =
  BEGIN
  -- define locals
    rx: xCoord;
    ry: yCoord;
  -- convert cursor coordinates to window coordinates
    rx ← x - (rectangle.x0 + rectangle.bitmap.x0);
    ry ← y - (rectangle.y0 + rectangle.bitmap.y0);
    RETURN[rx, ry];
  END;

  BitmapError: PUBLIC SIGNAL [bitmap: BMHandle, error: BitmapErrorCode] = CODE;

-- Global Display On/Off Routines

  DisplayOn: PUBLIC PROCEDURE =
    BEGIN
    -- locals
      ds: DisplayHandle;
      mapdata: BMHandle ← bitmaps;
      newpfontaddr: FAptr;
```

```
    -- reallocate the display bitmaps
      -- NOTE: this code relies on the fields "words" and
      -- "wordsperline" in the BitmapObject being valid
      UNTIL mapdata = NIL DO
        ReallocateBitmap[mapdata, mapdata.words/256, mapdata.wordsperline];
        mapdata ← mapdata.link;
        ENDLOOP;
    -- get default font back and fix up users of same
      SwapIn[defaultfontsegment];
      defaultfont ← FileSegmentAddress[defaultfontsegment];
      newpfontaddr ← LOOPHOLE[@defaultfont.FCDptrs, FAptr];
      ds ← StreamDefs.GetDisplayStreamList[];
      WHILE ds # NIL DO
        IF ds.pfont = defaultpfont THEN
          ds.pfont ← newpfontaddr;
        ds ← ds.link;
        ENDLOOP;
      defaultpfont ← newpfontaddr;
    -- now really turn it on
      DCBchainHead.next ← savedfirstDCB;
      FreeHeapNode[tempDCB];
    END;

  DisplayOff: PUBLIC PROCEDURE [background: backgtype] =
    BEGIN
    -- locals
      mapdata: BMHandle ← bitmaps;
      dcb: DCBptr;
    -- first really turn it off
      savedfirstDCB ← DCBchainHead.next;
      tempDCB ← AllocateHeapNode[SIZE[DCB]+1];
      dcb ← EVEN[tempDCB];
      MiscDefs.Zero[dcb, SIZE[DCB]];
      dcb.background ← background;
      dcb.resolution ← high;
      DCBchainHead.next ← dcb;
    -- deallocate the display bitmap space
      -- NOTE: Turn ON code relies on the fields "words" and
      -- "wordsperline" in the BitmapObject being valid
      UNTIL mapdata = NIL DO
        ReallocateBitmap[mapdata, 0, 0];
        mapdata ← mapdata.link;
        ENDLOOP;
    -- swapout the default font segment
      Unlock[defaultfontsegment];
      SwapOut[defaultfontsegment];
    END;

-- Font Stuff

  ComputeCharWidth: PUBLIC PROCEDURE [char: CHARACTER, font: POINTER] RETURNS [CARDINAL] =
    BEGIN
    -- define and setup locals
      w: INTEGER ← 0;
      code: CARDINAL;
      cw: FCDptr;
      temp: UNSPECIFIED;  -- because FCDptr's are self relative
      fontdesc: DESCRIPTOR FOR ARRAY OF FCDptr
      ← DESCRIPTOR[font, 256];
    -- checkfor control characters
      IF char = CR THEN char ← Space;
      IF char < Space THEN
        RETURN[ComputeCharWidth['↑, font] +
               ComputeCharWidth[
                 LOOPHOLE[LOOPHOLE[char,INTEGER]+100B,CHARACTER], font]];
    -- check if default guy
      code ← LOOPHOLE[char];
      IF font = defaultpfont AND char <= DEL THEN
        RETURN[LOOPHOLE[defaultcharwidths[LOOPHOLE[char,CARDINAL]],CARDINAL]]
      ELSE -- now compute the width of this character
        DO
          temp ← font + LOOPHOLE[code,CARDINAL];
          cw ← LOOPHOLE[fontdesc[LOOPHOLE[code,CARDINAL]]+temp, FCDptr];
          IF cw.HasNoExtension THEN EXIT;
          w ← w+16;
          code ← cw.widthORext;
```

```
        ENDLOOP;
      RETURN [w + cw.widthORext];
    END;

  GetDefaultFont: PUBLIC PROCEDURE RETURNS [FAptr, CARDINAL] =
    BEGIN
      RETURN[defaultpfont, defaultlineheight];
    END;

  GetFont: PUBLIC PROCEDURE [filename: STRING] RETURNS [FileSegmentHandle] =
    BEGIN
    RETURN[
      NewFileSegment[NewFile[filename, Read, OldFileOnly], DefaultBase, DefaultPages, Read]];
    END;

  LoadFont: PUBLIC PROCEDURE [segment: FileSegmentHandle] RETURNS [Fptr] =
    BEGIN
      SwapIn[segment];
      RETURN[FileSegmentAddress[segment]];
    END;

-- BitBlt Interface

  BitBlt: PUBLIC PROCEDURE [ptr: BBptr] =
    BEGIN
    HardwareBitBlt[ptr]
    END;

-- Mesa System Bitmap Initialization Routine

  initbitmap: PROCEDURE[pagesformap, mapwordsperline: CARDINAL] =
    BEGIN
    -- Declare Locals
      dcb: DCBptr;
      mapdata: BMHandle;
    -- setup BitBlt table for all to use
      -- BBTables must be on even word boundaries!!
      bbptr ← EVEN[bbptr];
    -- setup font stuff
      defaultfont ← LoadFont[defaultfontsegment];
      defaultlineheight ← defaultfont.FHeader.MaxHeight;
      SetUpCharWidths[];
    -- setup dummy spacing if at top of screen
      IF DCBchainHead.next = DCBnil THEN
        BEGIN
        -- assumes dummy dcb will never deallocated
        dcb ← EVEN[AllocateHeapNode[SIZE[DCB]+1]];
        MiscDefs.Zero[dcb, SIZE[DCB]];
        dcb.background ← white;
        dcb.resolution ← high;
        dcb.height ← blanklines*defaultlineheight/2;
        DCBchainHead.next ← dcb;
        END;
    -- allocate and clear space for the system default Bitmap
      mapdata ← CreateBitmap[pagesformap, mapwordsperline];
    -- indent the bitmap 3 words
      mapdata.indenting ← 3;
    -- link it and make it the system default
      DisplayBitmap[mapdata];
      defaultmapdata ← mapdata;
    END;

  SetUpCharWidths:  PROCEDURE=
    BEGIN
      i:  INTEGER;
      pfont:  FAptr ← LOOPHOLE[@defaultfont.FCDptrs, FAptr];
      defaultcharwidths.length ← 128;
      defaultpfont ← NIL;
      FOR i IN [0..128) DO
        -- NOTE:  ComputeCharWidth counts on the fact
        -- "defaultpfont" is NIL at this time
        defaultcharwidths[i] ←
          LOOPHOLE[ComputeCharWidth[LOOPHOLE[i,CHARACTER], pfont],CHARACTER];
        ENDLOOP;
      defaultpfont ← pfont;
    END;
```

```
  PatchUpLineHeight: PROCEDURE =
    BEGIN  ds: DisplayHandle;
    defaultlineheight ← defaultfont.FHeader.MaxHeight;
    FOR ds ← StreamDefs.GetDisplayStreamList[], ds.link UNTIL ds = NIL DO
      IF ds.pfont = defaultpfont THEN
        BEGIN
        ds.lineheight ← defaultlineheight;
        SetDisplayLine[ds, 1, leftmargin];
        END;
      ENDLOOP;
    RETURN
    END;

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

  CleanupRectangles: ImageDefs.CleanupProcedure =
    BEGIN
    SELECT why FROM
      Finish, Abort => DCBchainHead.next ← DCBnil;
      Save =>
        BEGIN
        DisplayOff[black];
        DeleteFileSegment[defaultfontsegment];
        mesapreopen.file ← NIL;
        syspreopen.file ← NIL;
        ImageDefs.AddFileRequest[@mesapreopen];
        ImageDefs.AddFileRequest[@syspreopen];
        END;
      Restore =>
        BEGIN
        InitFontFile[];
        DisplayOn[];
        SetUpCharWidths[];
        PatchUpLineHeight[];
        END;
      ENDCASE;
    RETURN
    END;

  InitFontFile: PROCEDURE =
    BEGIN
    IF syspreopen.file = NIL THEN ERROR;
    defaultfontsegment ← NewFileSegment[
      IF mesapreopen.file # NIL THEN mesapreopen.file ELSE syspreopen.file,
      DefaultBase, DefaultPages, Read];
    IF mesapreopen.file # NIL THEN ReleaseFile[syspreopen.file];
    END;


-- MAIN BODY CODE

  -- make file request on second START
  mesapreopen: short ImageDefs.FileRequest ← ImageDefs.FileRequest [
    file: NIL, access: Read, link:,
    body: short[fill:, name: "MesaFont.Al."]];
  syspreopen: short ImageDefs.FileRequest ← ImageDefs.FileRequest [
    file: NIL, access: Read, link:,
    body: short[fill:, name: "SysFont.Al."]];

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

  IF mesapreopen.file = NIL THEN
    mesapreopen.file ← NewFile[mesapreopen.name, Read, DefaultVersion
      ! FileNameError, FileError => CONTINUE];
  IF syspreopen.file = NIL THEN
    syspreopen.file ← NewFile[syspreopen.name, Read, DefaultVersion
      ! FileNameError, FileError => CONTINUE];

  -- now really do it
  InitFontFile[];
```

```
  initbitmap[pagesformap, mapwordsperline];
  ImageDefs.AddCleanupProcedure[@CleanupItem]

  END. of Rectangles
```