```
-- RectanglesA.Mesa  Edited by Sandman on May 12, 1978  2:48 PM

DIRECTORY
  BitBltDefs: FROM "bitbltdefs" USING [BBptr, BBTable, BITBLT],
  InlineDefs: FROM "inlinedefs" USING [BITAND],
  IODefs: FROM "iodefs" USING [CR, DEL, SP],
  RectangleDefs: FROM "rectangledefs" USING [
    BMHandle, BMptr, FAptr, FCDptr, FontHeader, GrayArray, GrayPtr, minheight,
    minwidth, RectangleErrorCode, Rptr, xCoord, yCoord];

RectanglesA: PROGRAM
  EXPORTS RectangleDefs SHARES RectangleDefs =
  BEGIN OPEN RectangleDefs;

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

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

  defaultmapdata: PUBLIC BMHandle ← NIL;
  defaultpfont: PUBLIC FAptr ← NIL;      -- points to self relative ptrs
  defaultlineheight: PUBLIC CARDINAL; -- assuming all lines equal;
  SevenBitCharacter: TYPE = CHARACTER[0C..177C];
  defaultcharwidths: PACKED ARRAY SevenBitCharacter OF [0..256);

-- Bitmap Rectangle Routines

  MoveRectangle: PUBLIC PROCEDURE [rectangle: Rptr, x: xCoord, y: yCoord] =
    BEGIN
    bbtable: ARRAY [0..SIZE[BitBltDefs.BBTable]] OF WORD;
    bbptr: BitBltDefs.BBptr = EVEN[BASE[bbtable]];
    oldx: xCoord = rectangle.x0;
    oldw: xCoord = rectangle.cw;
    oldy: yCoord = rectangle.y0;
    oldh: yCoord = rectangle.ch;
    mapaddr: BMptr = rectangle.bitmap.addr;
    wordsperline: CARDINAL = rectangle.bitmap.wordsperline;
    dlx, dw: xCoord;
    dty, dh: yCoord;
    IF x = oldx AND y = oldy THEN RETURN;
    rectangle.x0 ← x;
    rectangle.y0 ← y;
    FixupRectangle[rectangle];
    IF rectangle.visible = FALSE THEN RETURN;
    dw ← MIN[rectangle.cw, oldw];
    dh ← MIN[rectangle.ch, oldh];
    bbptr↑ ← [0, FALSE, FALSE, block, replace, 0, mapaddr, wordsperline, x,
      y, dw, dh, mapaddr, wordsperline, oldx, oldy, 0, 0, 0, 0];
    BitBltDefs.BITBLT[bbptr];
    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↑ ← [,,, gray, replace,,,, dlx, dty, dw, dh,,,,,,,,];
      BitBltDefs.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↑ ← [,,, gray, replace,,,, dlx, dty, dw, dh,,,,,,,,];
      BitBltDefs.BITBLT[bbptr];
      END;
    END;

  GrowRectangle: PUBLIC PROCEDURE [rectangle: Rptr, width: xCoord, height: yCoord] =
    BEGIN
    mapaddr: BMptr = rectangle.bitmap.addr;
    wordsperline: CARDINAL = rectangle.bitmap.wordsperline;
    clearwords: GrayArray ← [0, 0, 0, 0];
    graywords: GrayArray ← [125252B, 52525B, 125252B, 52525B];
    clear: GrayPtr = @clearwords;
    gray: GrayPtr = @graywords;
    IF width = rectangle.width AND height = rectangle.height THEN RETURN;
    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
    bbtable: ARRAY [0..SIZE[BitBltDefs.BBTable]] OF WORD;
    bbptr: BitBltDefs.BBptr = EVEN[BASE[bbtable]];
    mapaddr: BMptr ← rectangle.bitmap.addr;
    wordsperline: CARDINAL = rectangle.bitmap.wordsperline;
    dlx: xCoord ← rectangle.x0+x0;
    dty: yCoord ← rectangle.y0+y0;
    dw: xCoord ← MIN[rectangle.cw, width];
    dh: yCoord ← MIN[rectangle.ch, height];
    bbptr↑ ← [0, FALSE, FALSE, gray, replace, 0, mapaddr, wordsperline,
      dlx, dty, dw, dh, mapaddr, wordsperline, dlx, dty, gray[0],
      gray[1], gray[2], gray[3]];
    BitBltDefs.BITBLT[bbptr];
    END;

  DrawBoxInRectangle: PUBLIC PROCEDURE [
    rectangle: Rptr, x0, width: xCoord, y0, height: yCoord] =
    BEGIN
    bbtable: ARRAY [0..SIZE[BitBltDefs.BBTable]] OF WORD;
    bbptr: BitBltDefs.BBptr = EVEN[BASE[bbtable]];
    mapaddr: BMptr ← rectangle.bitmap.addr;
    wordsperline: INTEGER = rectangle.bitmap.wordsperline;
    dlx: xCoord ← rectangle.x0+x0;
    dty: yCoord ← rectangle.y0+y0;
    dw: xCoord ← MIN[rectangle.cw, width];
    dh: yCoord ← MIN[rectangle.ch, height];
    bbptr↑ ← [0, FALSE, FALSE, gray, replace, 0, mapaddr, wordsperline,
      dlx, dty, dw, 1, mapaddr, wordsperline, dlx, dty, -1, -1, -1, -1];
    BitBltDefs.BITBLT[bbptr];
    bbptr↑ ← [,,, gray,,,,,,, 1, dh,,,,,,,,];
    BitBltDefs.BITBLT[bbptr];
    bbptr↑ ← [,,,,,,,, dlx+dw-1, dty, 1, dh,,,,,,,,];
    BitBltDefs.BITBLT[bbptr];
    bbptr↑ ← [,,,,,,,, dlx, dty+dh-1, dw, 1,,,,,,,,];
    BitBltDefs.BITBLT[bbptr];
    END;

  InvertBoxInRectangle: PUBLIC PROCEDURE [
    rectangle: Rptr, x0, width: xCoord, y0, height: yCoord] =
    BEGIN
    bbtable: ARRAY [0..SIZE[BitBltDefs.BBTable]] OF WORD;
    bbptr: BitBltDefs.BBptr = EVEN[BASE[bbtable]];
    mapaddr: BMptr ← rectangle.bitmap.addr;
    wordsperline: CARDINAL = rectangle.bitmap.wordsperline;
    dlx: xCoord ← rectangle.x0+x0;
    dty: yCoord ← rectangle.y0+y0;
    dw: xCoord ← MIN[rectangle.cw, width];
    dh: yCoord ← MIN[rectangle.ch, height];
    bbptr↑ ← [0, FALSE, FALSE, compliment, replace, 0, mapaddr, wordsperline,
      dlx, dty, dw, dh, mapaddr, wordsperline, dlx, dty,,,,];
    BitBltDefs.BITBLT[bbptr];
    END;
```

```
ScrollBoxInRectangle: PUBLIC PROCEDURE [
  rectangle: Rptr, x0, width: xCoord, y0, height: yCoord, incr: INTEGER] =
  BEGIN
  bbtable: ARRAY [0..SIZE[BitBltDefs.BBTable]] OF WORD;
  bbptr: BitBltDefs.BBptr = EVEN[BASE[bbtable]];
  mapaddr: BMptr ← rectangle.bitmap.addr;
  wordsperline: CARDINAL = rectangle.bitmap.wordsperline;
  dlx: xCoord ← rectangle.x0+x0;
  dw: xCoord ← MIN[rectangle.cw, width];
  dh: yCoord ← MIN[rectangle.ch, (height-incr)];
  dty: yCoord;
  sty: yCoord;
  IF incr > 0 THEN
    BEGIN
    dty ← rectangle.y0+y0;
    sty ← dty+incr;
    END
  ELSE
    BEGIN
    sty ← rectangle.y0+y0;
    dty ← sty+incr;
    END;
  bbptr↑ ← [0, FALSE, FALSE, block, replace, 0, mapaddr, wordsperline,
    dlx, dty, dw, dh, mapaddr, wordsperline, dlx, sty,,,,];
  BitBltDefs.BITBLT[bbptr];
  END;

FixupRectangle: PROCEDURE [rectangle: Rptr] =
  BEGIN
  bmw: xCoord;
  bmh: yCoord;
  IF rectangle.bitmap = NIL OR rectangle.bitmap.addr = NIL THEN
    BEGIN
    rectangle.visible ← FALSE;
    RETURN;
    END;
  bmh ← rectangle.bitmap.height;
  bmw ← rectangle.bitmap.width;
  IF rectangle.x0+rectangle.width > bmw THEN
    rectangle.cw ← IF rectangle.x0 > bmw THEN 0 ELSE bmw-rectangle.x0
  ELSE
    rectangle.cw ← rectangle.width;
  IF rectangle.y0+rectangle.height > bmh THEN
    rectangle.ch ← IF rectangle.y0 > bmh THEN 0 ELSE bmh-rectangle.y0
  ELSE
    rectangle.ch ← rectangle.height;
  IF rectangle.x0+minwidth > bmw OR rectangle.y0+minheight > bmh 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] =
  BEGIN
  -- define locals and init them
  bbtable: ARRAY [0..SIZE[BitBltDefs.BBTable]] OF WORD;
  bbptr: BitBltDefs.BBptr = EVEN[BASE[bbtable]];
  cw: FCDptr;
  code: CARDINAL ← LOOPHOLE[char];
  cwidth:  xCoord;
  -- following is awful, undo later   signed: Smokey
  lineheight: CARDINAL = LOOPHOLE[(pfont-SIZE[FontHeader])↑[0]];
  IF pfont = defaultpfont  AND char <= DEL THEN
    cwidth ← defaultcharwidths[char]
  ELSE cwidth ← ComputeCharWidth[char, pfont];
  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];
    bbptr↑ ← [
      pad: 0,
      sourcealt: FALSE,
      destalt: FALSE,
      sourcetype: block,
      function: paint,
      unused:,
      dbca: rectangle.bitmap.addr,
      dbmr: rectangle.bitmap.wordsperline,
      dlx: x+rectangle.x0,
      dty:,
      dw: 16,
      dh:,
      sbca:,
      sbmr: 1,
      slx: 0,
      sty: 0,
      gray0:, gray1:, gray2:, gray3:];
    DO
      cw ← LOOPHOLE[pfont[code]+LOOPHOLE[pfont,CARDINAL]+code];
      bbptr.dty ← y + rectangle.y0 + 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;
      code ← cw.widthORext;
      ENDLOOP;
    RETURN[x+cwidth, y];
    END;

  WriteRectangleString: PUBLIC PROCEDURE [
    rectangle: Rptr, x: xCoord, y: yCoord, str: STRING, pfont: FAptr]
    RETURNS[xCoord, yCoord] =
    BEGIN
    i: CARDINAL;
    FOR i IN [0..str.length) DO
      [x, y] ← WriteRectangleChar[rectangle, x, y, str[i], pfont];
      ENDLOOP;
    RETURN[x,y]
    END;

  CursorToRecCoords: PUBLIC PROCEDURE [rectangle: Rptr, x: xCoord, y: yCoord]
    RETURNS[xCoord, yCoord] =
    BEGIN
    rx: xCoord;
    ry: yCoord;
    rx ← x - (rectangle.x0 + rectangle.bitmap.x0);
    ry ← y - (rectangle.y0 + rectangle.bitmap.y0);
    RETURN[rx, ry];
    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;
    mapx ← MAX[0, MIN[mapdata.width, INTEGER[x - mapdata.x0]]];
    mapy ← MAX[0, MIN[mapdata.height, INTEGER[y - mapdata.y0]]];
    RETURN[mapx, mapy]
    END;

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

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

-- Font Stuff

  ComputeCharWidth: PUBLIC PROCEDURE [char: CHARACTER, font: POINTER] RETURNS [CARDINAL] =
    BEGIN
    w: INTEGER ← 0;
    code: CARDINAL;
    cw: FCDptr;
    temp: UNSPECIFIED;  -- because FCDptr's are self relative
    fontdesc: DESCRIPTOR FOR ARRAY OF FCDptr ← DESCRIPTOR[font, 256];
    IF char = CR THEN char ← Space;
    IF char < Space THEN
      RETURN[ComputeCharWidth['↑, font] +
            ComputeCharWidth[
    LOOPHOLE[LOOPHOLE[char,INTEGER]+100B,CHARACTER], font]];
    code ← LOOPHOLE[char];
    IF font = defaultpfont AND char <= DEL THEN
      RETURN[defaultcharwidths[char]]
    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;


  END. of RectanglesA
```