```
--File:  WManSelection.mesa
--Edited by Sandman          October 7, 1977  9:20 AM

DIRECTORY
 WindowDefs:  FROM "windowdefs",
 StreamDefs:  FROM "streamdefs",
 SystemDefs:  FROM "systemdefs",
 MenuDefs:  FROM "menudefs",
 RectangleDefs:  FROM "rectangledefs",
 WManagerDefs:  FROM "wmanagerdefs";

DEFINITIONS FROM StreamDefs, MenuDefs, WindowDefs, RectangleDefs, WManagerDefs;

WManSelection:  PROGRAM[WMState:  WMDataHandle]
  IMPORTS WindowDefs, StreamDefs, SystemDefs, MenuDefs, RectangleDefs, WManagerDefs
  EXPORTS WManagerDefs
  SHARES StreamDefs, MenuDefs, WManagerDefs =
BEGIN

OPEN WMState;

CR: CHARACTER = 15C;
Space: CHARACTER = 40C;

  MenuSelect: PUBLIC PROCEDURE
    [w: WindowHandle, x: xCoord, y: yCoord]=
    BEGIN
    -- define locals
    index: INTEGER ← -1;
    mapx: xCoord;
    mapy: yCoord;
    defaultmenu:  DESCRIPTOR FOR ARRAY OF MenuItem =
      DESCRIPTOR[BASE[menuarray], LENGTH[menuarray]];
    -- check if a menu
    IF w.menu = NIL THEN
      w.menu ← CreateMenu[defaultmenu];
    -- paste it up there
    [mapx, mapy] ← CursorToMapCoords[defaultmapdata, x, y];
    mapy ← MIN[mapy, MAX[0,(w.rectangle.bitmap.height)
       -(LENGTH[w.menu.array]*defaultlineheight+2)]];
    DisplayMenu[w.menu, w.rectangle.bitmap, mapx, mapy];
    -- while the button is down select menu items
    WHILE GetMouseButton[] = Blue DO
      -- convert to rectangle coords
      x ← xcursorloc↑;
      y ← ycursorloc↑;
      -- and see if in menu
      [x, y] ← CursorToRectangleCoords[w.menu.rectangle, x, y];
      IF x > 0 AND x <= w.menu.rectangle.cw
           AND y > 0 AND y <= w.menu.rectangle.ch
        THEN index ← y/defaultlineheight
        ELSE index ← -1;
        MarkMenuItem[w.menu, index];
    ENDLOOP;
    -- and restore menus region and contents underneath
    ClearMenu[w.menu];
    -- see if command selected
    IF index # -1 THEN
        w.menu.array[index].proc[w, xcursorloc↑, ycursorloc↑];
    END;


 TextSelect: PUBLIC PROCEDURE
    [w: WindowHandle, x: xCoord, y: yCoord]=
   BEGIN
   -- Declare locals
   line, width: INTEGER;
   xpos: xCoord;
   saveindex, index: StreamIndex;
   sel: POINTER TO Selection;
   exsel: POINTER TO Selection;
   IF w.file # NIL THEN
     BEGIN
     -- first find character under the bug and then mark the selection
     sel ← SystemDefs.AllocateHeapNode[SIZE[Selection]];
     exsel ← SystemDefs.AllocateHeapNode[SIZE[Selection]];
```

```
      [line, xpos, width, index] ← ResolveBugToPosition[w, x, y];
      saveindex ← index;
      sel↑ ← Selection[xpos, xpos+width, line, line, index, index];
      MakeSelection[w, sel];
      --  check for extensions
      WHILE GetMouseButton[] = Red DO
        IF x # xcursorloc↑ OR y # ycursorloc↑ THEN
          BEGIN
          x ← xcursorloc↑; y ← ycursorloc↑;
          [line, xpos, width, index] ← ResolveBugToPosition[w, x, y];
          IF NOT EqualIndex[saveindex, index] THEN
            BEGIN
            IF (line >= w.selection.leftline) AND
             (xpos >= w.selection.leftx)
            THEN exsel↑ ← Selection[sel.leftx, xpos+width,
              sel.leftline, line, sel.leftindex, index]
            ELSE exsel↑ ← Selection[xpos, sel.rightx, line,
              sel.rightline, index, sel.rightindex];
            MakeSelection[w, exsel];
            saveindex ← index;
            END;
          END;
        ENDLOOP;
        SystemDefs.FreeHeapNode[sel];
        SystemDefs.FreeHeapNode[exsel];
        END;
      END;

WordSelect: PUBLIC PROCEDURE
    [w: WindowHandle, x: xCoord, y: yCoord]=
  BEGIN
    -- Declare Locals
    line: INTEGER;
    saveindex, index: StreamIndex;
    sel: POINTER TO Selection;
    exsel: POINTER TO Selection;
    IF w.file # NIL THEN
      BEGIN
      -- first find word under the bug and then mark the selection
      sel ← SystemDefs.AllocateHeapNode[SIZE[Selection]];
      exsel ← SystemDefs.AllocateHeapNode[SIZE[Selection]];
      [line, , , index] ← ResolveBugToPosition[w, x, y];
      saveindex ← index;
      --check both ways for space to find whole word
      sel.leftline ← line;
      ExtendTheWord[w,sel,index];
      MakeSelection[w, sel];
      --  check for extensions
      WHILE GetMouseButton[] = Yellow DO
        IF x # xcursorloc↑ OR y # ycursorloc↑ THEN
        BEGIN
        x ← xcursorloc↑; y ← ycursorloc↑;
        [line, , , index] ← ResolveBugToPosition[w, x, y];
        IF NOT EqualIndex[saveindex, index] THEN
        BEGIN
          --extend the word and the selection
          exsel.leftline ← line;
          ExtendTheWord[w,exsel,index];
          IF (exsel.leftline >= w.selection.leftline) AND
           (exsel.leftx >= w.selection.leftx)
          THEN exsel↑ ← Selection[sel.leftx, , sel.leftline, , sel.leftindex, ]
          ELSE exsel↑ ← Selection[, sel.rightx, , sel.rightline, , sel.rightindex];
          MakeSelection[w, exsel];
          saveindex ← index;
          END;
        END;
      ENDLOOP;
      SystemDefs.FreeHeapNode[sel];
      SystemDefs.FreeHeapNode[exsel];
      END;
    END;

ExtendTheWord: PROCEDURE [w: WindowHandle, sel: POINTER TO Selection, pos: StreamIndex] =
  BEGIN
  -- declare locals
  -- note that oldend/start point to previous word
  -- and end/start refer to current word
  -- save points to index of current character
  savedindex, oldend, oldstart, save:  StreamIndex;
  start, end:  StreamIndex;
  leftpos, rightpos:  xCoord;
  oldleft, oldright:  xCoord;
  savewidth, nlines, width, lineno:  INTEGER;
  char:  CHARACTER;
  firsttime:  BOOLEAN ← TRUE;
  lastline, overtheline:  BOOLEAN ← FALSE;
  oldchar, newchar:  {sp, ch, cc, xtra};
  linestarts:  DESCRIPTOR FOR ARRAY OF StreamIndex;
  nlines ← (w.rectangle.ch/w.ds.lineheight)-1;
  linestarts ← DESCRIPTOR[GetLineTable[],nlines];
  savedindex ← GetIndex[w.file]; lineno ← sel.leftline;
  IF lineno = nlines THEN lastline ← TRUE;
  SetIndex[w.file, linestarts[lineno - 1]];
  oldend ← oldstart ← start ← end ← GetIndex[w.file];
  savewidth ← oldleft ← oldright ← leftpos ← rightpos ← leftmargin;
  WHILE GrEqualIndex[pos,start] DO
    IF NOT firsttime THEN oldchar ← newchar;
    save ← GetIndex[w.file];
    char ← w.file.get[w.file
             ! StreamError => EXIT];
    width ← IF char = 11C THEN ComputeTabWidth[w.ds.pfont,rightpos]
      ELSE ComputeCharWidth[char, w.ds.pfont];
    IF NOT lastline THEN
      BEGIN
      IF save = linestarts[lineno] THEN
        BEGIN
        overtheline ← TRUE;
        lineno ← lineno + 1;
        IF lineno = nlines THEN
          BEGIN
          lastline ← TRUE;
          savewidth ← savewidth + width;
          END;
        rightpos ← leftmargin;
        END;
      END
    ELSE
      BEGIN
      savewidth ← savewidth + width;
      IF savewidth >= w.rectangle.cw THEN EXIT;
      END;
    IF char = CR THEN
      BEGIN
      IF overtheline AND rightpos = leftmargin THEN
        BEGIN
        rightpos ← oldright;
        lineno ← lineno - 1;
        END;
      EXIT;
      END;
    SELECT char FROM
      <Space                     => newchar ← cc;
      IN ['a..'z],IN ['A..'Z]  => newchar ← ch;
      IN ['0..'9]              => newchar ← ch;
      =Space                     => newchar ← sp;
      ENDCASE                  => newchar ← xtra;
    IF firsttime OR oldchar # newchar THEN
      BEGIN
      oldstart ← start; oldend ← end;
      oldleft ← leftpos; oldright ← rightpos;
      start ← end ← save;
      leftpos ← rightpos;
      rightpos ← rightpos + width;
      firsttime ← FALSE;
      END
    ELSE
      BEGIN
      rightpos ← rightpos + width;
      end ← save;
      IF EqualIndex[w.eofindex, GetIndex[w.file]] THEN EXIT;
      END;
    REPEAT
```

```
      FINISHED =>
        BEGIN
        start ← oldstart; end ← oldend;
        leftpos ← oldleft; rightpos ← oldright;
        END;
    ENDLOOP;
  SetIndex[w.file, savedindex];
  sel.rightline ← MAX[sel.leftline,lineno];
  sel↑ ← Selection[
      leftpos, rightpos, MIN[sel.leftline,lineno], , start, end];
  RETURN
  END;

  ComputeTabWidth: PROCEDURE [font: FAptr, x: xCoord]
    RETURNS [CARDINAL] =
    BEGIN
    tw: CARDINAL = ComputeCharWidth[' ,font] * 8;
    RETURN[tw - x MOD tw]
    END;

  CommandStuff: PUBLIC PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord]=
    BEGIN
    n: CARDINAL;
    IF ~useKeyset THEN RETURN;
    n ← GetKeySet[];
    IF w.ks # NIL THEN
      SELECT n FROM
        IN [1..26] => w.ks.putback[w.ks, 101B+n-1];
        27 => w.ks.putback[w.ks, '+];
        31 => w.ks.putback[w.ks, 1C]; -- Control A
        ENDCASE;
    END;

-- initialization for selection module

InitSelection:  PROCEDURE =
  BEGIN
  TextProcArray[RedYellowBlue] ← NullProc;
  TextProcArray[RedBlue] ← NullProc;
  TextProcArray[RedYellow] ← CommandStuff;
  TextProcArray[Red] ← TextSelect;
  TextProcArray[BlueYellow] ← NullProc;
  TextProcArray[Blue] ← MenuSelect;
  TextProcArray[Yellow] ← WordSelect;
  TextProcArray[None] ← NullProc;
  END;

-- MAIN BODY CODE
InitSelection[];

END. of wmanselection
```