```
-- File: Menus.mesa
-- Edited by Sandman, May 12, 1978  2:41 PM

DIRECTORY
  AltoFileDefs: FROM "altofiledefs" USING [FA],
  BitBltDefs: FROM "bitbltdefs" USING [BBptr, BBTable, BITBLT],
  InlineDefs: FROM "inlinedefs" USING [BITAND],
  MenuDefs: FROM "menudefs" USING [
    MenuArray, MenuHandle, MenuLeftMargin, MenuObject],
  RectangleDefs: FROM "rectangledefs" USING [
    BMHandle, BMptr, ClearBoxInRectangle, ComputeCharWidth, CreateRectangle,
    CursorToRecCoords, DestroyRectangle, DrawBoxInRectangle, FAptr,
    GetDefaultFont, GrayArray, GrayPtr, InvertBoxInRectangle, leftmargin,
    RectangleError, Rptr, WriteRectangleString, xCoord, yCoord],
  StreamDefs: FROM "streamdefs" USING [
    EqualIndex, GetFA, GrEqualIndex, GrIndex, JumpToFA, ModifyIndex, SetIndex,
    StreamError],
  StringDefs: FROM "stringdefs" USING [AppendChar],
  SystemDefs: FROM "systemdefs" USING [
    AllocateHeapNode, AllocateHeapString, AllocateSegment, FreeHeapNode,
    FreeSegment],
  WindowDefs: FROM "windowdefs" USING [
    GetLineTable, Selection, StreamIndex, WindowHandle];

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

Menus: PROGRAM
  IMPORTS RectangleDefs, StreamDefs, StringDefs, SystemDefs, WindowDefs
  EXPORTS MenuDefs, RectangleDefs, WindowDefs SHARES MenuDefs, RectangleDefs =
  BEGIN

-- GLOBAL Data

  defaultpfont: FAptr ← NIL;
  defaultlineheight: CARDINAL;
  nullindex: StreamIndex = StreamIndex[0,-1];
  originindex: StreamIndex = StreamIndex[0,0];
  CR: CHARACTER = 15C;

-- Mesa Display Menu Routines

  CreateMenu: PUBLIC PROCEDURE [array: MenuArray] RETURNS [MenuHandle] =
    BEGIN
    i, j: CARDINAL;
    menu: MenuHandle;
    width, widest: xCoord;
    -- compute width of widest command word
    [defaultpfont, defaultlineheight] ← GetDefaultFont[];
    widest ← 0;
    FOR i IN [0..LENGTH[array]) DO
      width ← 0;
      FOR j IN [0..array[i].keyword.length) DO
        width ← width + ComputeCharWidth[array[i].keyword[j], defaultpfont];
        ENDLOOP;
      IF width > widest THEN widest ← width;
      ENDLOOP;
    -- now create menu object and init it
    widest ← widest + MenuLeftMargin*2;
    menu ← SystemDefs.AllocateHeapNode[SIZE[MenuObject]];
    menu↑ ← MenuObject[NIL, -1, widest, NIL, NIL, NIL, array];
    RETURN[menu];
    END;

  DestroyMenu: PUBLIC PROCEDURE [menu: MenuHandle] =
    BEGIN
    -- NOTE: need to unlink from menu list later
    SystemDefs.FreeHeapNode[menu];
    END;

  DisplayMenu: PUBLIC PROCEDURE
    [menu: MenuHandle, mapdata: BMHandle, x: xCoord, y: yCoord] =
    BEGIN
    i: CARDINAL;
    cx: INTEGER ← MAX[0, INTEGER[x-(menu.width+2)]];
    cy: INTEGER;
```

```
    nitems: CARDINAL ← 0;
    length: CARDINAL = LENGTH[menu.array];
    clearwords: GrayArray ← [0, 0, 0, 0];
    clear: GrayPtr = @clearwords;
    [defaultpfont, defaultlineheight] ← GetDefaultFont[];
    -- save data first then clear it
    IF menu.index # -1 THEN nitems ← menu.index;
    nitems ← MAX[length/2, nitems];
    cy ← y-(nitems*defaultlineheight+(defaultlineheight/2));
    cy ← MAX[0, cy];
    menu.rectangle ← CreateRectangle[
      mapdata, cx, menu.width, cy, defaultlineheight*length+2];
    menu.rectangle.options.NoteOverflow ← TRUE;
    --test page size of menu before saving
    IF (((menu.rectangle.cw + 15) /16 +1) * menu.rectangle.ch ) / 256 <= 4
      THEN menu.dataseg ← LOOPHOLE[SaveRectangle[menu.rectangle]];
    menu.index ← -1;
    ClearBoxInRectangle[
      menu.rectangle, 0, menu.rectangle.cw, 0, menu.rectangle.ch, clear];
    -- now paint it up on screen
    DrawBoxInRectangle[
      menu.rectangle, 0, menu.rectangle.cw, 0, menu.rectangle.ch];
    FOR i IN [0..length) DO
      [cx,cy] ← WriteRectangleString[menu.rectangle, MenuLeftMargin,
  (i*defaultlineheight + 1), menu.array[i].keyword, defaultpfont
        ! RectangleError =>
          SELECT error FROM
            RightOverflow => CONTINUE;
            NotVisible , BottomOverflow => EXIT;
            ENDCASE];
      ENDLOOP;
    END;

  ClearMenu: PUBLIC PROCEDURE [menu: MenuHandle] =
    BEGIN
    clearwords: GrayArray ← [0, 0, 0, 0];
    clear: GrayPtr = @clearwords;
    IF menu.rectangle # NIL THEN
      BEGIN
      ClearBoxInRectangle[menu.rectangle, 0, menu.rectangle.cw, 0,
  menu.rectangle.ch, clear];
      IF menu.dataseg # NIL THEN
  BEGIN
  RestoreRectangle[menu.rectangle, LOOPHOLE[menu.dataseg]];
  menu.dataseg ← NIL;
  END;
      DestroyRectangle[menu.rectangle];
      menu.rectangle ← NIL;
      END;
    END;

  MarkMenuItem: PUBLIC PROCEDURE [menu: MenuHandle, index: INTEGER] =
    BEGIN
    i:  CARDINAL ← 1;
    oldIndex: CARDINAL ← menu.index;
    r: Rptr = menu.rectangle;
    [defaultpfont, defaultlineheight] ← GetDefaultFont[];
    IF menu.index = index THEN RETURN;
    IF menu.index = LENGTH[menu.array]-1 THEN i ← 0;
    IF menu.index # -1 THEN -- turn old guy off
      InvertBoxInRectangle[
  r, 1, r.cw-2, oldIndex*defaultlineheight+1, defaultlineheight-i];
    i ← IF index = LENGTH[menu.array]-1 THEN 0 ELSE 1;
    oldIndex ← index;
    IF index # -1 THEN -- turn new guy on
      InvertBoxInRectangle[
  r, 1, r.cw-2, oldIndex*defaultlineheight+1, defaultlineheight-i];
    menu.index ← index;
    END;

SaveRectangle: PUBLIC PROCEDURE [rectangle: Rptr] RETURNS [POINTER] =
  BEGIN
  --declare locals
  bbTable:  ARRAY[0..SIZE[BBTable]] OF WORD;
  bbptr:  BBptr = EVEN[BASE[bbTable]];
  SegPtr:  POINTER;
```

```
  wordsperline:  CARDINAL = rectangle.bitmap.wordsperline;
  mapaddr:  BMptr = rectangle.bitmap.addr;
  dw:  CARDINAL ← rectangle.cw;
  dh:  CARDINAL ← rectangle.ch;
  dwWords:  CARDINAL ← (dw + 15)/16 + 1;
  totalWords:  CARDINAL ← dwWords * dh;
  SegPtr ← SystemDefs.AllocateSegment[totalWords];
  bbptr↑ ← BBTable[0,FALSE,FALSE,block,replace,0,SegPtr,dwWords,
    0,0,dw,dh,mapaddr,wordsperline,rectangle.x0,
    rectangle.y0,0,0,0,0];
  BITBLT[bbptr];
  RETURN[SegPtr];
  END;


RestoreRectangle: PUBLIC PROCEDURE [rectangle: Rptr, SegPtr: POINTER] =
  BEGIN
  --declare locals
  bbTable:  ARRAY[0..SIZE[BBTable]] OF WORD;
  bbptr:  BBptr = EVEN[BASE[bbTable]];
  wordsperline:  INTEGER = rectangle.bitmap.wordsperline;
  mapaddr:  BMptr = rectangle.bitmap.addr;
  dw:  CARDINAL ← rectangle.cw;
  dh:  CARDINAL ← rectangle.ch;
  dwWords:  CARDINAL ← (dw + 15)/16 + 1;
  bbptr↑ ← BBTable[0,FALSE,FALSE,block,replace,0,mapaddr,wordsperline,
    rectangle.x0,rectangle.y0,dw,dh,SegPtr,
    dwWords,0,0,0,0,0,0];
  BITBLT[bbptr];
  SystemDefs.FreeSegment[SegPtr];
  RETURN;
  END;

EVEN: PROCEDURE [v: UNSPECIFIED] RETURNS [UNSPECIFIED] =
  BEGIN
  RETURN[v+ InlineDefs.BITAND[v,1]];
  END;

-- Selection routines

ResolveBugToPosition: PUBLIC PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord]
  RETURNS [line: CARDINAL, xpos: xCoord, width: CARDINAL, index: StreamIndex] =
  BEGIN
  -- Declare Locals
  char: CHARACTER;
  nlines, newpos: CARDINAL;
  -- NOTE: following array is ONE origin
  linestarts: DESCRIPTOR FOR ARRAY OF StreamIndex;
  IF w.file = NIL THEN RETURN;
  nlines ← (w.rectangle.ch/w.ds.lineheight)-1;
  linestarts ← DESCRIPTOR[GetLineTable[], nlines+1];
  IF EqualIndex[linestarts[0] ,nullindex] THEN -- empty window
    RETURN[1, leftmargin, 0, originindex];
  [x, y] ← CursorToRecCoords[w.rectangle, x, y];
  -- NOTE:  real line number = line + 1
  line ← MIN[MAX[y/w.ds.lineheight, 1], nlines];
  -- back up until real text in window
  UNTIL NOT EqualIndex[linestarts[line-1],nullindex] DO
    line ← line-1;
    ENDLOOP;
  index ← linestarts[line-1];
  SetIndex[w.file, index];
  xpos ← leftmargin;
  width ← 0;
  IF x <= xpos THEN
    BEGIN
    char ← w.file.get[w.file ! StreamError => GOTO Exit];
    width ← IF char = 11C THEN ComputeTabWidth[w.ds.pfont,xpos]
      ELSE ComputeCharWidth[char, w.ds.pfont];
    EXITS
      Exit => NULL;
    END
  ELSE
    DO
      char ← w.file.get[w.file ! StreamError => EXIT];
      width ← IF char = 11C THEN ComputeTabWidth[w.ds.pfont,xpos]
```

```
        ELSE ComputeCharWidth[char, w.ds.pfont];
      IF EqualIndex[index, w.eofindex] OR char = CR OR
        EqualIndex[ModifyIndex[index, 1],linestarts[line]] THEN EXIT;
      IF x < (newpos ← xpos + width) THEN EXIT;
      xpos ← newpos;
      index ← ModifyIndex[index, 1];
      ENDLOOP;
  RETURN[line, xpos, width, index];
  END;

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

  GetSelection: PUBLIC PROCEDURE [w: WindowHandle] RETURNS [STRING] =
    BEGIN
    -- Declare Locals
    count: CARDINAL;
    fa: AltoFileDefs.FA;
    str: STRING;
    -- put the selection into a string
    IF EqualIndex[w.selection.rightindex, nullindex] THEN RETURN[NIL];
    GetFA[w.file, @fa];
    count ← (w.selection.rightindex.page-w.selection.leftindex.page)*512 +
      (w.selection.rightindex.byte+1)-w.selection.leftindex.byte;
    str ← SystemDefs.AllocateHeapString[count];
    SetIndex[w.file, w.selection.leftindex];
    THROUGH [0..count) DO
      StringDefs.AppendChar[str, w.file.get[w.file]];
      ENDLOOP;
    JumpToFA[w.file, @fa];
    RETURN[str];
    END;

MakeSelection: PUBLIC PROCEDURE [w: WindowHandle, sel: POINTER TO Selection] =
  BEGIN
  -- unmark the old one if one exists and is visible
  IF NOT (EqualIndex[w.selection.leftindex, nullindex]
    OR w.selection.leftline = 0) THEN MarkSelection[w];
  -- mark the new one
  w.selection ← sel↑;
  MarkSelection[w];
  END;

MarkSelection: PUBLIC PROCEDURE [w: WindowHandle] =
  BEGIN
  i: CARDINAL;
  IF EqualIndex[w.selection.leftindex,nullindex]
    OR w.selection.leftline = 0 THEN RETURN;
  IF w.selection.leftline = w.selection.rightline THEN
    InvertBoxInRectangle[w.rectangle, w.selection.leftx,
      w.selection.rightx-w.selection.leftx,
      w.selection.leftline*w.ds.lineheight, w.ds.lineheight]
  ELSE
    BEGIN
    InvertBoxInRectangle[w.rectangle, w.selection.leftx,
      (w.rectangle.cw-1)-w.selection.leftx,
      w.selection.leftline*w.ds.lineheight, w.ds.lineheight];
    i ← w.selection.rightline-w.selection.leftline;
    IF i > 1 THEN
      InvertBoxInRectangle[
  w.rectangle, leftmargin, w.rectangle.cw-(leftmargin+1),
  (w.selection.leftline+1)*w.ds.lineheight, (i-1)*w.ds.lineheight];
    InvertBoxInRectangle[
      w.rectangle, leftmargin, w.selection.rightx-leftmargin,
      w.selection.rightline*w.ds.lineheight, w.ds.lineheight];
    END;
  END;

UpdateSelection:  PUBLIC PROCEDURE[w:  WindowHandle] =
  BEGIN
  --local data
  lastindex:  StreamIndex ← nullindex;
```

```
  fa: AltoFileDefs.FA;
  line:  CARDINAL;
  nlines:  CARDINAL ← (w.rectangle.ch/w.ds.lineheight)  - 1;
  i:  CARDINAL ← 0;
  linestarts : DESCRIPTOR FOR ARRAY OF StreamIndex;
  linestarts ← DESCRIPTOR[GetLineTable[], nlines +1];
  --figure out the real last line
  FOR line IN [1..nlines) DO
    IF EqualIndex[nullindex,linestarts[line]] THEN
      BEGIN lastindex ← w.eofindex; nlines ← line-1; EXIT; END;
    REPEAT
      FINISHED => lastindex ← linestarts[nlines];
    ENDLOOP;
  -- for no selection or out of bounds
  IF EqualIndex[w.selection.leftindex, nullindex] OR
    GrEqualIndex[w.selection.leftindex, lastindex] OR
    GrIndex[linestarts[0], w.selection.rightindex] THEN
    BEGIN w.selection.leftline ← 0; RETURN; END;
  GetFA[w.file, @fa];
  -- find line numbers
  IF GrEqualIndex[w.selection.leftindex, linestarts[0]] THEN
    FOR i ← 0, i+1 UNTIL i = nlines DO
      IF (GrEqualIndex[w.selection.leftindex, linestarts[i]]
      AND GrIndex[linestarts[i+1], w.selection.leftindex]) THEN EXIT;
      ENDLOOP;
  w.selection.leftline ← MAX[1, i+1];
  IF GrIndex[lastindex, w.selection.rightindex] THEN
    FOR i ← i, i+1 UNTIL i = nlines DO
      IF (GrEqualIndex[w.selection.rightindex, linestarts[i]]
      AND GrIndex[linestarts[i+1], w.selection.rightindex]) THEN EXIT;
      ENDLOOP;
  w.selection.rightline ← MIN[nlines+1, i+1];
  --find xcoords
  IF GrEqualIndex[w.selection.leftindex, linestarts[0]] THEN
    w.selection.leftx ←  GetPos[w, linestarts[w.selection.leftline-1],
      w.selection.leftindex, TRUE]
  ELSE w.selection.leftx ← leftmargin;
  IF GrIndex[lastindex, w.selection.rightindex] THEN
    w.selection.rightx ← GetPos[w, linestarts[w.selection.rightline-1],
      w.selection.rightindex, FALSE]
  ELSE w.selection.rightx ← w.rectangle.cw;
  JumpToFA[w.file, @fa];
  MarkSelection[w];
  END;

GetPos: PROCEDURE [
  w: WindowHandle, linestart: StreamIndex, index2: StreamIndex, left: BOOLEAN]
  RETURNS [xCoord] =
  BEGIN
  char:  CHARACTER;
  xpos:  xCoord ← leftmargin;
  width: CARDINAL;
  IF GrIndex[linestart, index2] THEN RETURN[xpos];
  IF EqualIndex[linestart, index2] AND left THEN RETURN[xpos];
  SetIndex[w.file, linestart];
  DO
    char ← w.file.get[w.file];
    width ← IF char = 11C THEN ComputeTabWidth[w.ds.pfont,xpos]
      ELSE ComputeCharWidth[char, w.ds.pfont];
    xpos ← xpos + width;
    IF EqualIndex[linestart, index2] THEN EXIT;
    linestart ← ModifyIndex[linestart, 1];
    ENDLOOP;
  IF left THEN xpos ← xpos - width;
  RETURN[xpos];
  END;

END. of Menus
```