Changes made to Original Oberon for Continuous Fractional Line Scrolling

Andreas Pirklbauer, 12.12.2015

Name:	Continuous Fractional Line Scroll, Edition 12.12.2015 (later adapted for Experimental Oberon)
Description:	Pixel-by-pixel scrolling of displayed texts, both the top and the bottom line can be fractional
Modified files:	`Display.Mod, TextFrames.Mod, MenuViewers.Mod`
Number of new lines:	**~200** (relative to Original Oberon)

```
(*---------------------------- Modified: Display.Mod ----------------------------------*)

MODULE Display;  (*NW 5.11.2013 / 3.7.2016 / AP 5.7.16*)
...

  PROCEDURE CopyPattern1*(col, patadr, x, y, top, bot, mode: INTEGER);  (*with top and bottom margins*)
    VAR a, a0, pwd: INTEGER;
      w, h, pbt: BYTE; pix: SET;
  BEGIN (*0 <= top < h & 0 <= bot < h & top + bot < h*)
    SYSTEM.GET(patadr, w); SYSTEM.GET(patadr+1, h); INC(patadr, 2 + bot*((w + 7) DIV 8));
    a := base + (x DIV 32)*4 + (y + bot)*128; h := h - top - bot;
    FOR a0 := a TO a + (h-1)*128 BY 128 DO
      (*build pattern line; w < 32*)
      SYSTEM.GET(patadr, pbt); INC(patadr); pwd := pbt;
      IF w > 8 THEN SYSTEM.GET(patadr, pbt); INC(patadr); pwd := pbt*100H + pwd;
        IF w > 16 THEN SYSTEM.GET(patadr, pbt); INC(patadr); pwd := pbt*10000H + pwd;
          IF w > 24 THEN SYSTEM.GET(patadr, pbt); INC(patadr); pwd := pbt*1000000H + pwd END
        END
      END ;
      SYSTEM.GET(a0, pix);
      IF mode = invert THEN SYSTEM.PUT(a0, SYSTEM.VAL(SET, LSL(pwd, x MOD 32)) / pix)
      ELSE SYSTEM.PUT(a0, SYSTEM.VAL(SET, LSL(pwd, x MOD 32)) + pix)
      END ;
      IF (x MOD 32) + w > 32 THEN (*spill over*)
        SYSTEM.GET(a0+4, pix);
        IF mode = invert THEN SYSTEM.PUT(a0+4, SYSTEM.VAL(SET, ASR(pwd, -(x MOD 32))) / pix)
        ELSE SYSTEM.PUT(a0+4, SYSTEM.VAL(SET, ASR(pwd, -(x MOD 32))) + pix)
        END
      END
    END
  END CopyPattern1;

  PROCEDURE CopyPattern*(col, patadr, x, y, mode: INTEGER);
  BEGIN CopyPattern1(col, patadr, x, y, 0, 0, mode)
  END CopyPattern;
  ...
END Display.


(*---------------------------- Modified: TextFrames.Mod ----------------------------------*)

MODULE TextFrames; (*JG 8.10.90 / NW 16.11.15 / AP 12.12.15*)
  IMPORT Modules, Input, Display, Viewers, Fonts, Texts, Oberon, MenuViewers;

  CONST replace* = 0; insert* = 1; delete* = 2; unmark* = 3; (*update message ids*)
    BS = 8X; TAB = 9X; CR = 0DX; DEL = 7FX; inf = 32767;

  TYPE Line = POINTER TO LineDesc;
    LineDesc = RECORD
      len: LONGINT;
      wid: INTEGER;
      eot: BOOLEAN;
      next: Line
    END;

    Location* = RECORD
      org*, pos*: LONGINT;
      dx*, x*, y*: INTEGER;
      lin: Line
    END;

    Frame* = POINTER TO FrameDesc;
    FrameDesc* = RECORD (Display.FrameDesc)
      text*: Texts.Text;
      org*, col*: INTEGER;
      lsp*: INTEGER;
      left*, right*, top*, bot*: INTEGER;
      markH*: INTEGER;
      time*: LONGINT;
      hasCar*, hasSel*, hasMark: BOOLEAN;
      carloc*: Location;
```

```
    selbeg*, selend*: Location;
    trailer: Line;
    voff: INTEGER; (*vertical offset relative to baseline*)
    pool: Line (*line pool to minimize heap usage*)
  END;

  UpdateMsg* = RECORD (Display.FrameMsg)
    id*: INTEGER;       (*replace, insert, delete, unmark*)
    text*: Texts.Text;
    beg*, end*: LONGINT
  END;

  CopyOverMsg = RECORD (Display.FrameMsg)
    text: Texts.Text;
    beg, end: LONGINT
  END;

VAR TBuf*, DelBuf: Texts.Buffer;
  menuH*, barW*, left*, right*, top*, bot*, lsp*: INTEGER; (*standard sizes*)
  asr, dsr, selH, markW, eolW: INTEGER;
  nextCh: CHAR;
  ScrollMarker: Oberon.Marker;
  W, KW: Texts.Writer; (*keyboard writer*)

PROCEDURE Min (i, j: INTEGER): INTEGER;
BEGIN IF i < j THEN j := i END ;
  RETURN j
END Min;

PROCEDURE Max (i, j: INTEGER): INTEGER;
BEGIN IF i > j THEN j := i END ;
  RETURN j
END Max;

PROCEDURE NewLine (F: Frame; VAR L: Line); (*reuse line from line pool if possible*)
BEGIN IF F.pool # NIL THEN L := F.pool; F.pool := L.next ELSE NEW(L) END
END NewLine;

PROCEDURE LastLine (F: Frame; L: Line); (*move lines after L to line pool*)
  VAR l: Line;
BEGIN (*L in closed F.trailer ring*)
  IF L.next # F.trailer THEN l := L;
    WHILE l.next # F.trailer DO l := l.next END;
    l.next := F.pool; F.pool:= L.next; L.next := F.trailer
  END
END LastLine;

(*------------------display support-------------------------*)

PROCEDURE ReplConst (col: INTEGER; F: Frame; X, Y, W, H: INTEGER; mode: INTEGER);
  VAR topY, botY: INTEGER;
BEGIN
  IF W > 0 THEN topY := F.Y + F.H - F.top; botY := F.Y + F.bot;
    IF Y < botY THEN H := H - botY + Y; Y := botY END;
    IF Y + H > topY THEN H := topY - Y END;
    IF H > 0 THEN
      IF X + W <= F.X + F.W THEN Display.ReplConst(col, X, Y, W, H, mode)
      ELSIF X < F.X + F.W THEN Display.ReplConst(col, X, Y, F.X + F.W - X, H, mode)
      END
    END
  END
END ReplConst;

PROCEDURE FlipSM(X, Y: INTEGER);
  VAR DW, DH, CL: INTEGER;
BEGIN DW := Display.Width; DH := Display.Height; CL := DW;
  IF X < CL THEN
    IF X < 3 THEN X := 3 ELSIF X > DW - 4 THEN X := DW - 4 END
  ELSE
    IF X < CL + 3 THEN X := CL + 4 ELSIF X > CL + DW - 4 THEN X := CL + DW - 4 END
  END ;
  IF Y < 6 THEN Y := 6 ELSIF Y > DH - 6 THEN Y := DH - 6 END;
  Display.CopyPattern(Display.white, Display.updown, X-4, Y-4, Display.invert)
END FlipSM;

PROCEDURE UpdateMark (F: Frame);  (*in scroll bar*)
  VAR oldH: INTEGER;
BEGIN oldH := F.markH; F.markH := F.org * F.H DIV (F.text.len + 1);
  IF F.hasMark & (F.left >= barW) & (F.markH # oldH) THEN
    Display.ReplConst(Display.white, F.X + 1, F.Y + F.H - 1 - oldH, markW, 1, Display.invert);
    Display.ReplConst(Display.white, F.X + 1, F.Y + F.H - 1 - F.markH, markW, 1, Display.invert)
  END
```

```
END UpdateMark;

PROCEDURE SetChangeMark (F: Frame; on: BOOLEAN);  (*in corner*)
BEGIN
  IF F.H > menuH THEN
    IF on THEN Display.CopyPattern(Display.white, Display.block, F.X+F.W-12, F.Y+F.H-12, Display.paint)
    ELSE Display.ReplConst(F.col, F.X+F.W-12, F.Y+F.H-12, 8, 8, Display.replace)
    END
  END
END SetChangeMark;

PROCEDURE Width (VAR R: Texts.Reader; len: LONGINT): INTEGER;
  VAR patadr, pos: LONGINT; ox, dx, x, y, w, h: INTEGER;
BEGIN pos := 0; ox := 0;
  WHILE pos < len DO
    Fonts.GetPat(R.fnt, nextCh, dx, x, y, w, h, patadr);
    ox := ox + dx; INC(pos); Texts.Read(R, nextCh)
  END;
  RETURN ox
END Width;

PROCEDURE DisplayLine (F: Frame; L: Line;
  VAR R: Texts.Reader; X, Y, topY, botY: INTEGER; len: LONGINT);
  VAR patadr, NX, dx, x, y, w, h, t0, b0, y0: INTEGER;
BEGIN NX := F.X + F.W - F.right;
  WHILE (nextCh # CR) & (R.fnt # NIL) DO
    Fonts.GetPat(R.fnt, nextCh, dx, x, y, w, h, patadr);
    IF (X + x + w <= NX) & (h # 0) THEN y0 := Y + y;
      IF y0 + h <= topY THEN t0 := 0 ELSIF y0 >= topY THEN t0 := h ELSE t0 := y0 + h - topY END;
      IF y0 >= botY THEN b0 := 0 ELSIF y0 + h <= botY THEN b0 := h ELSE b0 := botY - y0 END;
      IF t0 + b0 < h THEN Display.CopyPattern1(R.col, patadr, X + x, y0, t0, b0, Display.invert) END
    END;
    X := X + dx; INC(len); Texts.Read(R, nextCh)
  END;
  L.len := len + 1; L.wid := X + eolW - (F.X + F.left);
  L.eot := R.fnt = NIL; Texts.Read(R, nextCh)
END DisplayLine;

PROCEDURE Validate (T: Texts.Text; VAR pos: LONGINT);
  VAR R: Texts.Reader;
BEGIN
  IF pos > T.len THEN pos := T.len
  ELSIF pos > 0 THEN
    DEC(pos); Texts.OpenReader(R, T, pos);
    REPEAT Texts.Read(R, nextCh); INC(pos) UNTIL R.eot OR (nextCh = CR)
  ELSE pos := 0
  END
END Validate;

PROCEDURE Mark* (F: Frame; on: BOOLEAN);
BEGIN
  IF (F.H > 0) & (F.left >= barW) & ((F.hasMark & ~on) OR (~F.hasMark & on)) THEN
    Display.ReplConst(Display.white, F.X + 1, F.Y + F.H - 1 - F.markH, markW, 1, Display.invert)
  END;
  F.hasMark := on
END Mark;

(*------------------frame modifiers-------------------------*)

PROCEDURE Restore* (F: Frame);
  VAR R: Texts.Reader; L, l: Line;
    curY, topY, botY, botY: INTEGER;
BEGIN Display.ReplConst(F.col, F.X, F.Y, F.W, F.H, Display.replace);
  IF F.left >= barW THEN
    Display.ReplConst(Display.white, F.X + barW - 1, F.Y, 1, F.H, Display.invert)
  END;
  Validate(F.text, F.org);
  topY := F.Y + F.H - F.top; botY := F.Y + F.bot; L := F.trailer;
  IF topY > botY THEN curY := topY + F.voff;
    Texts.OpenReader(R, F.text, F.org); Texts.Read(R, nextCh);
    WHILE ~L.eot & (curY > botY) DO
      NewLine(F, l);
      DisplayLine(F, l, R, F.X + F.left, curY - asr, topY, botY, 0);
      L.next := l; L := l; curY := curY - lsp
    END
  END;
  L.next := F.trailer;
  F.markH := F.org * F.H DIV (F.text.len + 1)
END Restore;

PROCEDURE Suspend* (F: Frame);
BEGIN F.trailer.next := F.trailer
```

```
END Suspend;

PROCEDURE Extend* (F: Frame; newY: INTEGER);
  VAR R: Texts.Reader; L, l: Line;
    org: LONGINT; curY, topY, botY, botY0: INTEGER;
BEGIN Display.ReplConst(F.col, F.X, newY, F.W, F.Y - newY, Display.replace);
  IF F.left >= barW THEN
    Display.ReplConst(Display.white, F.X + barW - 1, newY, 1, F.Y - newY, Display.invert)
  END;
  topY := F.Y + F.H - F.top; botY0 := F.Y + F.bot;
  F.H := F.H + F.Y - newY; F.Y := newY; botY := F.Y + F.bot;
  IF F.trailer.next = F.trailer THEN Validate(F.text, F.org) END;
  L := F.trailer; org := F.org; curY := topY + F.voff;
  WHILE (L.next # F.trailer) & (curY > botY0) DO
    L := L.next; org := org + L.len; curY := curY - lsp
  END;
  IF (L # F.trailer) & (curY < botY0) THEN
    Texts.OpenReader(R, F.text, org - L.len); Texts.Read(R, nextCh);
    DisplayLine(F, L, R, F.X + F.left, curY + dsr, botY0, botY, 0) (*old fractional bottom line*)
  ELSE Texts.OpenReader(R, F.text, org); Texts.Read(R, nextCh)
  END;
  IF topY > botY THEN
    WHILE ~L.eot & (curY > botY) DO
      NewLine(F, l);
      DisplayLine(F, l, R, F.X + F.left, curY - asr, topY, botY, 0);
      L.next := l; L := l; curY := curY - lsp
    END
  END;
  L.next := F.trailer;
  F.markH := F.org * F.H DIV (F.text.len + 1)
END Extend;

PROCEDURE Reduce* (F: Frame; newY: INTEGER);
  VAR L: Line; curY, topY, botY: INTEGER;
BEGIN F.H := F.H + F.Y - newY; F.Y := newY;
  topY := F.Y + F.H - F.top; botY := F.Y + F.bot; L := F.trailer;
  IF topY > botY THEN curY := topY + F.voff;
    WHILE (L.next # F.trailer) & (curY > botY) DO
      L := L.next; curY := curY - lsp
    END
  END;
  LastLine(F, L);
  IF F.H >= F.top + F.bot THEN
    Display.ReplConst(F.col, F.X + F.left, F.Y, F.W - F.left, F.bot, Display.replace)
  ELSIF F.H > F.top THEN
    Display.ReplConst(F.col, F.X + F.left, F.Y, F.W - F.left, F.H - F.top, Display.replace)
  END;
  F.markH := F.org * F.H DIV (F.text.len + 1); Mark(F, TRUE)
END Reduce;

(*------------------fractional line scroll support-----------------------*)

PROCEDURE ScrollDown (F: Frame; org: LONGINT; voff, dY: INTEGER);
  VAR R: Texts.Reader; L, L0, l: Line;
    curY, topY, botY, Y0, Y1, y, h: INTEGER;
BEGIN topY := F.Y + F.H - F.top; botY := F.Y + F.bot;
  curY := topY + F.voff - dY; Y0 := Max(curY, botY); Y1 := Max(topY - dY, botY);
  F.org := org; F.voff := voff; L := F.trailer;
  WHILE (L.next # F.trailer) & (curY > botY) DO
    L := L.next; curY := curY - lsp
  END;
  LastLine(F, L);
  y := Max(curY, botY); h := Y1 - y;
  IF h > 0 THEN Display.CopyBlock(F.X + F.left, topY - h, F.W - F.left, h, F.X + F.left, y, 0) END;
  ReplConst(F.col, F, F.X + F.left, Y1, F.W - F.left, topY - Y1, Display.replace);
  L := F.trailer; L0 := L.next; curY := topY + voff;
  Texts.OpenReader(R, F.text, org); Texts.Read(R, nextCh);
  WHILE ~L.eot & (curY > Y0) DO
    NewLine(F, l);
    DisplayLine(F, l, R, F.X + F.left, curY - asr, topY, Y0, 0);
    L.next := l; L := l; curY := curY - lsp
  END;
  L.next := L0;
  IF (L0 # F.trailer) & (curY > Y1) THEN
    DisplayLine(F, L0, R, F.X + F.left, curY - asr, topY, Y1, 0) (*old fractional top line*)
  END;
  UpdateMark(F)
END ScrollDown;

PROCEDURE ScrollUp (F: Frame; org: LONGINT; voff, dY, curY: INTEGER);
  VAR R: Texts.Reader; L, l: Line;
    topY, botY, Y0, Y1, y, h: INTEGER;
```

```
BEGIN topY := F.Y + F.H - F.top; botY := F.Y + F.bot;
  F.org := org; F.voff := voff; L := F.trailer.next; Y0 := curY;
  WHILE L.next # F.trailer DO
    org := org + L.len; curY := curY - lsp; L := L.next
  END;
  y := Max(curY - lsp, botY); Y1 := Min(y + dY, topY); h := Y0 - y - F.voff;
  IF h > 0 THEN Display.CopyBlock(F.X + F.left, y, F.W - F.left, h, F.X + F.left, Y1, 0) END;
  ReplConst(F.col, F, F.X + F.left, y, F.W - F.left, Y1 - y, Display.replace);
  org := org + L.len; curY := curY - lsp + dY;
  IF curY - dY < botY THEN Texts.OpenReader(R, F.text, org - L.len); Texts.Read(R, nextCh);
    DisplayLine(F, L, R, F.X + F.left, curY + dsr, Y1, botY, 0) (*old fractional bottom line*)
  ELSE Texts.OpenReader(R, F.text, org); Texts.Read(R, nextCh)
  END;
  WHILE ~L.eot & (curY > botY) DO
    NewLine(F, l);
    DisplayLine(F, l, R, F.X + F.left, curY - asr, topY, botY, 0);
    L.next := l; L := l; curY := curY - lsp
  END;
  L.next := F.trailer; UpdateMark(F)
END ScrollUp;

PROCEDURE Scroll (F: Frame; dY: INTEGER); (*scroll displayed text dY pixels up or down*)
  CONST P = 100; len = 75; (*assumed average line length*)
  VAR R: Texts.Reader; L, L0: Line; done: BOOLEAN;
    org, q: LONGINT; dy, k: INTEGER; p: ARRAY P OF LONGINT;
BEGIN (*dY # 0*)
  IF F.trailer.next # F.trailer THEN
    IF dY < 0 THEN dY := -dY;
      IF dY <= F.voff THEN ScrollDown(F, F.org, F.voff - dY, dY)
      ELSE done := FALSE;
        q := Max(F.org - len*((dY + lsp - F.voff - 1) DIV lsp), 0); (*first guess*)
        REPEAT org := q; Validate(F.text, org); k := 0; dy := 0;
          Texts.OpenReader(R, F.text, org);
          WHILE (org < F.org) & (k < P) DO dy := dy + lsp; p[k] := org; INC(k);
            REPEAT Texts.Read(R, nextCh); INC(org) UNTIL R.eot OR (nextCh = CR)
          END;
          IF org < F.org THEN q := p[1] (*next guess forward*)
          ELSIF q = 0 THEN (*reached beginning of text*) done := TRUE;
            IF dy + F.voff > 0 THEN ScrollDown(F, 0, 0, dy + F.voff) END
          ELSIF (k = 0) OR (dy + F.voff < dY) THEN q := Max(q - len, 0) (*next guess backward*)
          ELSE (*found, now scroll*) k := 0; done := TRUE;
            WHILE dy - lsp + F.voff >= dY DO dy := dy - lsp; INC(k) END;
            ScrollDown(F, p[k], F.voff + dy - dY, dY)
          END
        UNTIL done
      END
    ELSIF dY > 0 THEN
      org := F.org; L0 := F.trailer; L := F.trailer.next; dy := 0;
      WHILE (L # F.trailer) & (dy + lsp <= dY + F.voff) DO
        org := org + L.len; dy := dy + lsp; L0 := L; L := L.next
      END;
      IF L # F.trailer THEN
        IF L0 # F.trailer THEN L0.next := F.pool; F.pool := F.trailer.next; F.trailer.next := L END;
        ScrollUp(F, org, F.voff + dY - dy, dY, F.Y + F.H - F.top + F.voff - dy)
      ELSIF org < F.text.len THEN Texts.OpenReader(R, F.text, org);
        WHILE (org < F.text.len) & (dy + lsp <= dY + F.voff) DO dy := dy + lsp;
          REPEAT Texts.Read(R, nextCh); INC(org) UNTIL R.eot OR (nextCh = CR)
        END;
        IF (org < F.text.len) & (dy + lsp > dY + F.voff) THEN Mark(F, FALSE);
          F.org := org; F.voff := F.voff + dY - dy; LastLine(F, F.trailer); Restore(F); Mark(F, TRUE)
        END
      END
    END;
    SetChangeMark(F, F.text.changed)
  END
END Scroll;

PROCEDURE Show* (F: Frame; pos: LONGINT); (*scroll specified text position to the top*)
  VAR R: Texts.Reader; L, L0: Line;
    org: LONGINT; dy, k, m: INTEGER;
BEGIN
  IF F.trailer.next # F.trailer THEN
    Validate(F.text, pos);
    IF pos < F.org THEN
      org := pos; k := 0; dy := 0; m := F.H - F.top - F.bot + F.voff;
      Texts.OpenReader(R, F.text, org); Texts.Read(R, nextCh);
      WHILE (org < F.org) & (dy < m) DO dy := dy + lsp; INC(k);
        REPEAT Texts.Read(R, nextCh); INC(org) UNTIL R.eot OR (nextCh = CR)
      END;
      IF (org = F.org) & (k > 0) & (dy < m) THEN ScrollDown(F, pos, 0, dy + F.voff)
      ELSE Mark(F, FALSE); F.org := pos; F.voff := 0; LastLine(F, F.trailer); Restore(F); Mark(F, TRUE)
      END
```

```
    ELSIF pos > F.org THEN
      org := F.org; L0 := F.trailer; L := F.trailer.next; dy := 0;
      WHILE (L # F.trailer) & (org < pos) DO
        org := org + L.len; dy := dy + lsp; L0 := L; L := L.next
      END;
      IF L # F.trailer THEN
        IF L0 # F.trailer THEN L0.next := F.pool; F.pool := F.trailer.next; F.trailer.next := L END;
        ScrollUp(F, org, 0, dy - F.voff, F.Y + F.H - F.top + F.voff - dy)
      ELSIF pos < F.text.len THEN Mark(F, FALSE);
        F.org := pos; F.voff := 0; LastLine(F, F.trailer); Restore(F); Mark(F, TRUE)
      END
    ELSIF F.voff > 0 THEN ScrollDown(F, pos, 0, F.voff)
    END;
    SetChangeMark(F, F.text.changed)
  END
END Show;

(*------------------locators-------------------------*)

PROCEDURE LocateLine (F: Frame; y: INTEGER; VAR loc: Location);
  VAR L: Line; org: LONGINT; cury: INTEGER;
BEGIN org := F.org; L := F.trailer.next; cury := F.H - F.top - asr + F.voff;
  WHILE (L.next # F.trailer) & (cury > y + dsr) DO
    org := org + L.len; L := L.next; cury := cury - lsp
  END;
  loc.org := org; loc.lin := L; loc.y := cury
END LocateLine;

PROCEDURE LocateString (F: Frame; x, y: INTEGER; VAR loc: Location);
  VAR R: Texts.Reader;
    patadr, bpos, pos, lim: LONGINT;
    bx, ex, ox, dx, u, v, w, h: INTEGER;
BEGIN LocateLine(F, y, loc);
  lim := loc.org + loc.lin.len - 1;
  bpos := loc.org; bx := F.left;
  pos := loc.org; ox := F.left;
  Texts.OpenReader(R, F.text, loc.org); Texts.Read(R, nextCh);
  REPEAT
    WHILE (pos # lim) & (nextCh > " ") DO (*scan string*)
      Fonts.GetPat(R.fnt, nextCh, dx, u, v, w, h, patadr);
      INC(pos); ox := ox + dx; Texts.Read(R, nextCh)
    END;
    ex := ox;
    WHILE (pos # lim) & (nextCh <= " ") DO (*scan gap*)
      Fonts.GetPat(R.fnt, nextCh, dx, u, v, w, h, patadr);
      INC(pos); ox := ox + dx; Texts.Read(R, nextCh)
    END;
    IF (pos # lim) & (ox <= x) THEN
      Fonts.GetPat(R.fnt, nextCh, dx, u, v, w, h, patadr);
      bpos := pos; bx := ox;
      INC(pos); ox := ox + dx; Texts.Read(R, nextCh)
    ELSE pos := lim
    END
  UNTIL pos = lim;
  loc.pos := bpos; loc.dx := ex - bx; loc.x := bx
END LocateString;

PROCEDURE LocateChar (F: Frame; x, y: INTEGER; VAR loc: Location);
  VAR R: Texts.Reader;
    patadr, pos, lim: LONGINT;
    ox, dx, u, v, w, h: INTEGER;
BEGIN LocateLine(F, y, loc);
  lim := loc.org + loc.lin.len - 1;
  pos := loc.org; ox := F.left; dx := eolW;
  Texts.OpenReader(R, F.text, loc.org);
  WHILE pos # lim DO
    Texts.Read(R, nextCh);
    Fonts.GetPat(R.fnt, nextCh, dx, u, v, w, h, patadr);
    IF ox + dx <= x THEN
      INC(pos); ox := ox + dx;
      IF pos = lim THEN dx := eolW END
    ELSE lim := pos
    END
  END ;
  loc.pos := pos; loc.dx := dx; loc.x := ox
END LocateChar;

PROCEDURE LocatePos (F: Frame; pos: LONGINT; VAR loc: Location);
  VAR T: Texts.Text; R: Texts.Reader; L: Line;
    org: LONGINT; cury: INTEGER;
BEGIN T := F.text;
  org := F.org; L := F.trailer.next; cury := F.H - F.top - asr + F.voff;
```

```
  IF pos < org THEN pos := org END;
  WHILE (L.next # F.trailer) & (pos >= org + L.len) DO
    org := org + L.len; L := L.next; cury := cury - lsp
  END;
  IF pos >= org + L.len THEN pos := org + L.len - 1 END;
  Texts.OpenReader(R, T, org); Texts.Read(R, nextCh);
  loc.org := org; loc.pos := pos; loc.lin := L;
  loc.x := F.left + Width(R, pos - org); loc.y := cury
END LocatePos;

PROCEDURE Pos* (F: Frame; X, Y: INTEGER): LONGINT;
  VAR loc: Location;
BEGIN LocateChar(F, X - F.X, Y - F.Y, loc); RETURN loc.pos
END Pos;

(*------------------caret and selection, mouse tracking-----------------------*)

PROCEDURE FlipCaret (F: Frame);
BEGIN
  IF (F.carloc.x < F.W) & (F.carloc.y >= 10) & (F.carloc.x + 12 < F.W) THEN
    Display.CopyPattern(Display.white, Display.hook,
        F.X + F.carloc.x, F.Y + F.carloc.y - 10, Display.invert)
  END
END FlipCaret;

PROCEDURE SetCaret* (F: Frame; pos: LONGINT);
BEGIN LocatePos(F, pos, F.carloc); FlipCaret(F); F.hasCar := TRUE
END SetCaret;

PROCEDURE TrackCaret* (F: Frame; X, Y: INTEGER; VAR keysum: SET);
  VAR loc: Location; keys: SET;
BEGIN
  IF F.trailer.next # F.trailer THEN
    LocateChar(F, X - F.X, Y - F.Y, F.carloc);
    FlipCaret(F);
    keysum := {};
    REPEAT Input.Mouse(keys, X, Y); keysum := keysum + keys;
      Oberon.DrawMouseArrow(X, Y); LocateChar(F, X - F.X, Y - F.Y, loc);
      IF loc.pos # F.carloc.pos THEN FlipCaret(F); F.carloc := loc; FlipCaret(F) END
    UNTIL keys = {};
    F.hasCar := TRUE
  END
END TrackCaret;

PROCEDURE RemoveCaret* (F: Frame);
BEGIN IF F.hasCar THEN FlipCaret(F); F.hasCar := FALSE END
END RemoveCaret;

PROCEDURE FlipSelection (F: Frame; VAR beg, end: Location);
  VAR L: Line; Y: INTEGER; org: LONGINT; beg0: Location;
BEGIN
  IF end.org >= F.org THEN
    org := F.org; L := F.trailer.next; Y := F.Y + F.H - F.top - lsp + F.voff;
    WHILE (L # F.trailer) & (org < beg.org) DO
      org := org + L.len; L := L.next; Y := Y - lsp
    END;
    IF L # F.trailer THEN
      IF beg.org < F.org THEN beg0.org := F.org; beg0.x := F.left ELSE beg0 := beg END;
      IF beg0.org = end.org THEN
        ReplConst(Display.white, F, F.X + beg0.x, Y, end.x - beg0.x, selH, Display.invert)
      ELSE
        ReplConst(Display.white, F, F.X + beg0.x, Y, F.left + L.wid - beg0.x, selH, Display.invert);
        org := org + L.len; L := L.next; Y := Y - lsp;
        WHILE (L # F.trailer) & (org < end.org) DO
          ReplConst(Display.white, F, F.X + F.left, Y, L.wid, selH, Display.invert);
          org := org + L.len; L := L.next; Y := Y - lsp
        END;
        IF (L # F.trailer) & (org = end.org) THEN
          ReplConst(Display.white, F, F.X + F.left, Y, end.x - F.left, selH, Display.invert)
        END
      END
    END
  END
END FlipSelection;

PROCEDURE SetSelection* (F: Frame; beg, end: LONGINT);
BEGIN
  IF F.hasSel THEN FlipSelection(F, F.selbeg, F.selend) END;
  LocatePos(F, beg, F.selbeg); LocatePos(F, end, F.selend);
  IF F.selbeg.pos < F.selend.pos THEN
    FlipSelection(F, F.selbeg, F.selend); F.time := Oberon.Time(); F.hasSel := TRUE
  END
```

```
END SetSelection;

PROCEDURE TrackSelection* (F: Frame; X, Y: INTEGER; VAR keysum: SET);
  VAR loc, L, R: Location; keys: SET; start, prev: LONGINT; topY, botY: INTEGER;
BEGIN
  IF F.trailer.next # F.trailer THEN topY := F.Y + F.H - F.top; botY := F.Y + F.bot;
    IF F.hasSel THEN FlipSelection(F, F.selbeg, F.selend) END;
    LocateChar(F, X - F.X, Y - F.Y, loc);
    F.selbeg := loc; start := loc.pos;
    INC(loc.pos); loc.x := loc.x + loc.dx; F.selend := loc;
    FlipSelection(F, F.selbeg, F.selend); keysum := {};
    REPEAT prev := loc.pos;
      Input.Mouse(keys, X, Y);
      keysum := keysum + keys;
      IF Y < botY THEN (*scroll text up*)
        IF (F.selbeg.org = F.selend.org) OR (prev <= start) THEN L := F.selbeg; R := F.selend
        ELSE R := F.selend; L.org := F.selend.org; L.pos := L.org; L.x := F.left
        END;
        FlipSelection(F, L, R); Oberon.FadeMouse;
        Scroll(F, botY - Y); Y := botY;
        FlipSelection(F, L, R)
      ELSIF (Y > TopY - 1) & ((F.org > 0) OR (F.voff > 0)) THEN (*scroll text down*)
        IF (F.selbeg.org = F.selend.org) OR (prev > start) THEN L := F.selbeg; R := F.selend
        ELSE L := F.selbeg; R.org := F.selbeg.org; R.pos := F.selbeg.lin.len; R.x := F.left + F.selbeg.lin.wid
        END;
        FlipSelection(F, L, R); Oberon.FadeMouse;
        Scroll(F, TopY - 1 - Y); Y := TopY - 1;
        FlipSelection(F, L, R)
      END;
      Oberon.DrawMouseArrow(X, Y);
      LocateChar(F, X - F.X, Y - F.Y, loc);
      IF prev > start THEN INC(loc.pos); loc.x := loc.x + loc.dx;
        IF loc.pos > start THEN
          IF loc.pos < F.selend.pos THEN FlipSelection(F, loc, F.selend); F.selend := loc
          ELSIF loc.pos > F.selend.pos THEN FlipSelection(F, F.selend, loc); F.selend := loc
          END
        ELSE (*switch to left of start*) FlipSelection(F, F.selbeg, F.selend);
          F.selend := F.selbeg; F.selbeg := loc; FlipSelection(F, F.selbeg, F.selend)
        END
      ELSIF loc.pos > start THEN (*switch to right of start*) FlipSelection(F, F.selbeg, F.selend);
          F.selbeg := F.selend; F.selend := loc; FlipSelection(F, F.selbeg, F.selend)
      ELSIF loc.pos > F.selbeg.pos THEN FlipSelection(F, F.selbeg, loc); F.selbeg := loc
      ELSIF loc.pos < F.selbeg.pos THEN FlipSelection(F, loc, F.selbeg); F.selbeg := loc
      END
    UNTIL keys = {};
    F.time := Oberon.Time(); F.hasSel := TRUE
  END
END TrackSelection;

PROCEDURE RemoveSelection* (F: Frame);
BEGIN IF F.hasSel THEN FlipSelection(F, F.selbeg, F.selend); F.hasSel := FALSE END
END RemoveSelection;

PROCEDURE TrackLine* (F: Frame; X, Y: INTEGER; VAR org: LONGINT; VAR keysum: SET);
  VAR old, new: Location; keys: SET;
BEGIN
  IF F.trailer.next # F.trailer THEN
    LocateLine(F, Y - F.Y, old);
    ReplConst(Display.white, F, F.X + F.left, F.Y + old.y - dsr, old.lin.wid, 2, Display.invert);
    keysum := {};
    REPEAT Input.Mouse(keys, X, Y);
      keysum := keysum + keys;
      Oberon.DrawMouse(ScrollMarker, X, Y);
      LocateLine(F, Y - F.Y, new);
      IF new.org # old.org THEN
        ReplConst(Display.white, F, F.X + F.left, F.Y + old.y - dsr, old.lin.wid, 2, Display.invert);
        ReplConst(Display.white, F, F.X + F.left, F.Y + new.y - dsr, new.lin.wid, 2, Display.invert);
        old := new
      END
     UNTIL keys = {};
     ReplConst(Display.white, F, F.X + F.left, F.Y + new.y - dsr, new.lin.wid, 2, Display.invert);
     org := new.org
  ELSE org := 0
  END
END TrackLine;

PROCEDURE TrackWord* (F: Frame; X, Y: INTEGER; VAR pos: LONGINT; VAR keysum: SET);
  VAR old, new: Location; keys: SET;
BEGIN
  IF F.trailer.next # F.trailer THEN
    LocateString(F, X - F.X, Y - F.Y, old);
    ReplConst(Display.white, F, F.X + old.x, F.Y + old.y - dsr, old.dx, 2, Display.invert);
```

```
    keysum := {};
    REPEAT
      Input.Mouse(keys, X, Y); keysum := keysum + keys;
      Oberon.DrawMouseArrow(X, Y);
      LocateString(F, X - F.X, Y - F.Y, new);
      IF new.pos # old.pos THEN
        ReplConst(Display.white, F, F.X + old.x, F.Y + old.y - dsr, old.dx, 2, Display.invert);
        ReplConst(Display.white, F, F.X + new.x, F.Y + new.y - dsr, new.dx, 2, Display.invert);
        old := new
      END
    UNTIL keys = {};
    ReplConst(Display.white, F, F.X + new.x, F.Y + new.y - dsr, new.dx, 2, Display.invert);
    pos := new.pos
  ELSE pos := 0
  END
END TrackWord;

(*------------------text modifiers-------------------------*)

PROCEDURE Replace* (F: Frame; beg, end: LONGINT);
  VAR R: Texts.Reader; L: Line;
    org, len: LONGINT; curY, topY, botY, wid: INTEGER;
BEGIN
  IF end > F.org THEN topY := F.Y + F.H - F.top;
    IF beg < F.org THEN beg := F.org END;
    org := F.org; L := F.trailer.next; curY := topY + F.voff;
    WHILE (L # F.trailer) & (org + L.len <= beg) DO
      org := org + L.len; L := L.next; curY := curY - lsp
    END;
    IF L # F.trailer THEN botY := F.Y + F.bot;
      Texts.OpenReader(R, F.text, org); Texts.Read(R, nextCh);
      len := beg - org; wid := Width(R, len);
      ReplConst(F.col, F, F.X + F.left + wid, curY - lsp, L.wid - wid, lsp, Display.replace);
      DisplayLine(F, L, R, F.X + F.left + wid, curY - asr, topY, botY, len);
      org := org + L.len; L := L.next; curY := curY - lsp;
      WHILE (L # F.trailer) & (org <= end) DO
        ReplConst(F.col, F, F.X + F.left, curY - lsp, F.W - F.left, lsp, Display.replace);
        DisplayLine(F, L, R, F.X + F.left, curY - asr, topY, botY, 0);
        org := org + L.len; L := L.next; curY := curY - lsp
      END
    END
  END;
  UpdateMark(F)
END Replace;

PROCEDURE Insert* (F: Frame; beg, end: LONGINT);
  VAR R: Texts.Reader; L, L0, l: Line;
    org, len: LONGINT; curY, topY, botY, Y0, Y1, Y2, dY, wid, bhid: INTEGER;
BEGIN
  IF beg < F.org THEN F.org := F.org + (end - beg)
  ELSE topY := F.Y + F.H - F.top;
    org := F.org; L := F.trailer.next; curY := topY + F.voff;
    WHILE (L # F.trailer) & (org + L.len <= beg) DO
      org := org + L.len; L := L.next; curY := curY - lsp
    END;
    IF L # F.trailer THEN botY := F.Y + F.bot;
      Texts.OpenReader(R, F.text, org); Texts.Read(R, nextCh);
      len := beg - org; wid := Width(R, len);
      ReplConst(F.col, F, F.X + F.left + wid, curY - lsp, L.wid - wid, lsp, Display.replace);
      DisplayLine(F, L, R, F.X + F.left + wid, curY - asr, topY, botY, len);
      org := org + L.len; curY := curY - lsp;
      Y0 := curY; L0 := L.next;
      WHILE (org <= end) & (curY > botY) DO
        NewLine(F, l);
        ReplConst(F.col, F, F.X + F.left, curY - lsp, F.W - F.left, lsp, Display.replace);
        DisplayLine(F, l, R, F.X + F.left, curY - asr, topY, botY, 0);
        L.next := l; L := l; org := org + L.len; curY := curY - lsp
      END;
      IF L0 # L.next THEN Y1 := curY;
        L.next := L0;
        WHILE (L.next # F.trailer) & (curY > botY) DO
          L := L.next; curY := curY - lsp
        END;
        LastLine(F, L);
        dY := Y0 - Y1;
        IF Y1 > curY + dY THEN
          IF curY >= botY THEN bhid := 0 ELSE bhid := lsp - (curY + lsp - botY) MOD lsp END;
          Display.CopyBlock(F.X + F.left, curY + dY + bhid,
              F.W - F.left, Y1 - curY - dY - bhid, F.X + F.left, curY + bhid, 0);
          Y2 := Y1 - dY
        ELSE Y2 := curY
        END;
```

```
        curY := Y1; L := L0;
        WHILE curY # Y2 DO
          ReplConst(F.col, F, F.X + F.left, curY - lsp, F.W - F.left, lsp, Display.replace);
          DisplayLine(F, L, R, F.X + F.left, curY - asr, topY, botY, 0);
          L := L.next; curY := curY - lsp
        END
      END
    END
  END;
  UpdateMark(F)
END Insert;

PROCEDURE Delete* (F: Frame; beg, end: LONGINT);
  VAR R: Texts.Reader; L, L0, l: Line;
    org, org0, len: LONGINT; curY, topY, botY, Y0, Y1, wid, bvis: INTEGER;
BEGIN
  IF end <= F.org THEN F.org := F.org - (end - beg)
  ELSE topY := F.Y + F.H - F.top;
    IF beg < F.org THEN
      F.trailer.next.len := F.trailer.next.len + (F.org - beg);
      F.org := beg
    END;
    org := F.org; L := F.trailer.next; curY := topY + F.voff;
    WHILE (L # F.trailer) & (org + L.len <= beg) DO
      org := org + L.len; L := L.next; curY := curY - lsp
    END;
    IF L # F.trailer THEN botY := F.Y + F.bot;
      org0 := org; L0 := L; l := L; Y0 := curY;
      WHILE (L # F.trailer) & (org <= end) DO
        org := org + L.len; l := L; L := L.next; curY := curY - lsp
      END;
      Y1 := curY;
      Texts.OpenReader(R, F.text, org0); Texts.Read(R, nextCh);
      len := beg - org0; wid := Width(R, len);
      ReplConst(F.col, F, F.X + F.left + wid, Y0 - lsp, L0.wid - wid, lsp, Display.replace);
      DisplayLine(F, L0, R, F.X + F.left + wid, Y0 - asr, topY, botY, len);
      Y0 := Y0 - lsp;
      IF L # L0.next THEN
        IF l.next = L THEN l.next := F.pool; F.pool := L0.next; L0.next := L END;
        L := L0; org := org0 + L0.len;
        WHILE L.next # F.trailer DO
          L := L.next; org := org + L.len; curY := curY - lsp
        END;
        IF curY < botY THEN bvis := lsp - (botY - curY);
          IF Y1 > botY THEN
            Display.CopyBlock(F.X + F.left, botY, F.W - F.left, Y1 - botY, F.X + F.left, botY + Y0 - Y1, 0)
          END;
          curY := curY + (Y0 - Y1) + lsp;
          ReplConst(F.col, F, F.X + F.left, botY, F.W - F.left, curY - bvis - botY, Display.replace);
          Texts.OpenReader(R, F.text, org - L.len); Texts.Read(R, nextCh);
          DisplayLine(F, L, R, F.X + F.left, curY - asr, curY - bvis, botY, 0); (*old fractional bottom line*)
          curY := curY - lsp
        ELSE Display.CopyBlock(F.X + F.left, curY, F.W - F.left, Y1 - curY, F.X + F.left, curY + Y0 - Y1, 0);
          curY := curY + (Y0 - Y1);
          ReplConst(F.col, F, F.X + F.left, botY, F.W - F.left, curY - botY, Display.replace)
        END;
        Texts.OpenReader(R, F.text, org); Texts.Read(R, nextCh);
        WHILE ~L.eot & (curY > botY) DO
          NewLine(F, l);
          DisplayLine(F, l, R, F.X + F.left, curY - asr, topY, botY, 0);
          L.next := l; L := l; curY := curY - lsp
        END;
        L.next := F.trailer
      END
    END
  END;
  UpdateMark(F)
END Delete;

PROCEDURE Recall*(VAR B: Texts.Buffer);
BEGIN B := TBuf; NEW(TBuf); Texts.OpenBuf(TBuf)
END Recall;

(*------------------message handling-------------------------*)

PROCEDURE RemoveMarks (F: Frame);
BEGIN RemoveCaret(F); RemoveSelection(F)
END RemoveMarks;

PROCEDURE NotifyDisplay* (T: Texts.Text; op: INTEGER; beg, end: LONGINT);
  VAR M: UpdateMsg;
BEGIN M.id := op; M.text := T; M.beg := beg; M.end := end; Viewers.Broadcast(M)
```

```
END NotifyDisplay;

PROCEDURE Call* (F: Frame; pos: LONGINT; new: BOOLEAN);
  VAR S: Texts.Scanner; res: INTEGER;
BEGIN
  Texts.OpenScanner(S, F.text, pos); Texts.Scan(S);
  IF (S.class = Texts.Name) & (S.line = 0) THEN
    Oberon.SetPar(F, F.text, pos + S.len); Oberon.Call(S.s, res);
    IF res > 0 THEN
      Texts.WriteString(W, "Call error: "); Texts.WriteString(W, Modules.importing);
      IF res = 1 THEN Texts.WriteString(W, " module not found")
      ELSIF res = 2 THEN  Texts.WriteString(W, " bad version")
      ELSIF res = 3 THEN Texts.WriteString(W, " imports ");
        Texts.WriteString(W, Modules.imported); Texts.WriteString(W, " with bad key");
      ELSIF res = 4 THEN Texts.WriteString(W, " corrupted obj file")
      ELSIF res = 5 THEN Texts.WriteString(W, " command not found")
      ELSIF res = 7 THEN Texts.WriteString(W, " insufficient space")
      END;
      Texts.WriteLn(W); Texts.Append(Oberon.Log, W.buf)
    END
  END
END Call;

PROCEDURE Write* (F: Frame; ch: CHAR; fnt: Fonts.Font; col, voff: INTEGER);
  VAR buf: Texts.Buffer;
BEGIN (*F.hasCar*)
  IF ch = BS THEN (*backspace*)
    IF F.carloc.pos > F.org THEN
      Texts.Delete(F.text, F.carloc.pos - 1, F.carloc.pos, DelBuf); SetCaret(F, F.carloc.pos - 1)
    END
  ELSIF ch = 1X THEN (*ctrl-a select-all*)
    IF F.hasSel THEN FlipSelection(F, F.selbeg, F.selend) END;
    F.selbeg.org := 0; F.selbeg.pos := 0; F.selbeg.x := F.left;
    F.selend.org := F.text.len; F.selend.pos := F.text.len; F.selend.x := F.left;
    FlipSelection(F, F.selbeg, F.selend); F.time := Oberon.Time(); F.hasSel := TRUE
  ELSIF ch = 3X THEN (*ctrl-c copy*)
    IF F.hasSel THEN
      NEW(TBuf); Texts.OpenBuf(TBuf); Texts.Save(F.text, F.selbeg.pos, F.selend.pos, TBuf)
    END
  ELSIF ch = 16X THEN (*ctrl-v paste*)
    NEW(buf); Texts.OpenBuf(buf); Texts.Copy(TBuf, buf); Texts.Insert(F.text, F.carloc.pos, buf);
    SetCaret(F, F.carloc.pos + TBuf.len)
  ELSIF ch = 18X THEN (*ctrl-x cut*)
    IF F.hasSel THEN
      NEW(TBuf); Texts.OpenBuf(TBuf); Texts.Delete(F.text, F.selbeg.pos, F.selend.pos, TBuf)
    END
  ELSIF (20X <= ch) & (ch <= DEL) OR (ch = CR) OR (ch = TAB) THEN
    KW.fnt := fnt; KW.col := col; KW.voff := voff; Texts.Write(KW, ch);
    Texts.Insert(F.text, F.carloc.pos, KW.buf);
    SetCaret(F, F.carloc.pos + 1)
  END
END Write;

PROCEDURE Defocus* (F: Frame);
BEGIN RemoveCaret(F)
END Defocus;

PROCEDURE Neutralize* (F: Frame);
BEGIN RemoveMarks(F)
END Neutralize;

PROCEDURE Modify* (F: Frame; id, dY, Y, H: INTEGER);
BEGIN
  Mark(F, FALSE); RemoveMarks(F); SetChangeMark(F,  FALSE);
  IF id = MenuViewers.extend THEN
    IF dY > 0 THEN Display.CopyBlock(F.X, F.Y, F.W, F.H, F.X, F.Y + dY, 0); F.Y := F.Y + dY END;
    Extend(F, Y)
  ELSIF id = MenuViewers.reduce THEN
    Reduce(F, Y + dY);
    IF dY > 0 THEN Display.CopyBlock(F.X, F.Y, F.W, F.H, F.X, Y, 0); F.Y := Y END
  END;
  IF F.H > 0 THEN Mark(F, TRUE); SetChangeMark(F,  F.text.changed) END
END Modify;

PROCEDURE Open* (F: Frame; H: Display.Handler; T: Texts.Text; org: LONGINT;
      col, left, right, top, bot, lsp: INTEGER);
  VAR L: Line;
BEGIN NEW(L); F.pool := NIL; F.voff := 0;
  L.len := 0; L.wid := 0; L.eot := FALSE; L.next := L;
  F.handle := H; F.text := T; F.org := org; F.trailer := L;
  F.left := left; F.right := right; F.top := top; F.bot := bot;
  F.lsp := lsp; F.col := col; F.hasMark := FALSE; F.hasCar := FALSE; F.hasSel := FALSE
```

```
END Open;

PROCEDURE Copy* (F: Frame; VAR F1: Frame);
BEGIN NEW(F1);
  Open(F1, F.handle, F.text, F.org, F.col, F.left, F.right, F.top, F.bot, F.lsp)
END Copy;

PROCEDURE CopyOver(F: Frame; text: Texts.Text; beg, end: LONGINT);
  VAR buf: Texts.Buffer;
BEGIN
  IF F.hasCar THEN
    NEW(buf); Texts.OpenBuf(buf);
    Texts.Save(text, beg, end, buf); Texts.Insert(F.text, F.carloc.pos, buf);
    SetCaret(F, F.carloc.pos + (end - beg))
  END
END CopyOver;

PROCEDURE GetSelection* (F: Frame; VAR text: Texts.Text; VAR beg, end, time: LONGINT);
BEGIN
  IF F.hasSel THEN
    IF F.time > time THEN
      text := F.text; beg := F.selbeg.pos; end := F.selend.pos; time := F.time
    ELSIF F.text = text THEN
      IF (F.time < time) & (F.selbeg.pos < beg) THEN beg := F.selbeg.pos
        ELSIF (F.time > time) & (F.selend.pos > end) THEN end := F.selend.pos; time := F.time
      END
    END
  END
END GetSelection;

PROCEDURE Update* (F: Frame; VAR M: UpdateMsg);
BEGIN (*F.text = M.text*) SetChangeMark(F, FALSE);
  RemoveMarks(F); Oberon.RemoveMarks(F.X, F.Y, F.W, F.H);
  IF M.id = replace THEN Replace(F, M.beg, M.end)
  ELSIF M.id = insert THEN Insert(F, M.beg, M.end)
  ELSIF M.id = delete THEN Delete(F, M.beg, M.end)
  END ;
  SetChangeMark(F,  F.text.changed)
END Update;

PROCEDURE Edit* (F: Frame; X, Y: INTEGER; Keys: SET);
  VAR M: CopyOverMsg;
    text: Texts.Text;
    buf: Texts.Buffer;
    v: Viewers.Viewer;
    beg, end, time, pos: LONGINT;
    keysum: SET;
    fnt: Fonts.Font;
    col, voff, Y0, SL, SR: INTEGER;
BEGIN SL := F.X + Min(F.left, barW);
  IF X < SL THEN  (*cursor starts in scroll bar*)
    Oberon.DrawMouse(ScrollMarker, X, Y); keysum := Keys;
    IF Keys = {2} THEN  (*ML: continuous scroll*)
      Y0 := F.Y + F.H - 1 - F.markH; SR := SL + Min(100, F.W DIV 2);
      WHILE Keys # {} DO Oberon.DrawMouse(ScrollMarker, X, Y);
        IF Y # Y0 THEN
          SetChangeMark(F, FALSE);
          RemoveMarks(F); Oberon.RemoveMarks(F.X, F.Y, F.W, F.H);
          IF X < SR THEN (*cursor stays near scroll bar*)
            Show(F, (F.Y + F.H - Y) * (F.text.len) DIV F.H)
          ELSE (*cursor moves into text area*)
            Scroll(F, Y0 - Y)
          END ;
          Y0 := Y
        END ;
        Input.Mouse(Keys, X, Y)
      END
    ELSIF Keys = {1} THEN  (*MM: positional scrolling*)   keysum := Keys;
      REPEAT Input.Mouse(Keys, X, Y); keysum := keysum + Keys;
        Oberon.DrawMouse(ScrollMarker, X, Y)
      UNTIL Keys = {};
      IF keysum # {0, 1, 2} THEN
        IF 0 IN keysum THEN pos := 0
        ELSIF 2 IN keysum THEN pos := Max(F.text.len - 40, 0)
        ELSE pos := (F.Y + F.H - Y) * (F.text.len) DIV F.H
        END ;
        SetChangeMark(F, FALSE);
        RemoveMarks(F); Oberon.RemoveMarks(F.X, F.Y, F.W, F.H);
        Show(F, pos)
      END
    ELSIF Keys = {0} THEN  (*MR: scroll up or down*)
      TrackLine(F, X, Y, pos, keysum);
```

```
      IF keysum # {0, 1, 2} THEN
        IF (pos >= 0) & (keysum = {0}) THEN (*MR, scroll up*)
          SetChangeMark(F, FALSE);
          RemoveMarks(F); Oberon.RemoveMarks(F.X, F.Y, F.W, F.H);
          Show(F, pos)
        ELSIF (keysum = {0,1}) THEN (*MR and MM, scroll down*)
          SetChangeMark(F, FALSE);
          RemoveMarks(F); Oberon.RemoveMarks(F.X, F.Y, F.W, F.H);
          Show(F, F.org*2 - pos - 100)
        END
      END
    END
  ELSE  (*cursor is in text area*)
    Oberon.DrawMouseArrow(X, Y);
    IF 0 IN Keys THEN  (*MR: select*)
      TrackSelection(F, X, Y, keysum);
      IF F.hasSel THEN
        IF keysum = {0, 2} THEN (*MR, ML: delete text*)
          Oberon.GetSelection(text, beg, end, time);
          Texts.Delete(text, beg, end, TBuf);
          Oberon.PassFocus(Viewers.This(F.X, F.Y)); SetCaret(F, beg)
        ELSIF keysum = {0, 1} THEN  (*MR, MM: copy to caret*)
          Oberon.GetSelection(text, beg, end, time);
          M.text := text; M.beg := beg; M.end := end;
          Oberon.FocusViewer.handle(Oberon.FocusViewer, M)
        END
      END
    ELSIF 1 IN Keys THEN  (*MM: call*)
      TrackWord(F, X, Y, pos, keysum);
      IF (pos >= 0) & ~(0 IN keysum) THEN Call(F, pos, 2 IN keysum) END
    ELSIF 2 IN Keys THEN  (*ML: set caret*)
      Oberon.PassFocus(Viewers.This(F.X, F.Y));
      TrackCaret(F, X, Y, keysum);
      IF keysum = {2, 1} THEN (*ML, MM: copy from selection to caret*)
        Oberon.GetSelection(text, beg, end, time);
        IF time >= 0 THEN
          NEW(TBuf); Texts.OpenBuf(TBuf);
          Texts.Save(text, beg, end, TBuf); Texts.Insert(F.text, F.carloc.pos, TBuf);
          SetSelection(F, F.carloc.pos, F.carloc.pos + (end  - beg));
          SetCaret(F, F.carloc.pos + (end - beg))
        ELSIF TBuf # NIL THEN
          NEW(buf); Texts.OpenBuf(buf);
          Texts.Copy(TBuf, buf); Texts.Insert(F.text, F.carloc.pos, buf);
          SetCaret(F, F.carloc.pos + buf.len)
        END
      ELSIF keysum = {2, 0} THEN (*ML, MR: copy looks*)
        Oberon.GetSelection(text, beg, end, time);
        IF time >= 0 THEN
          Texts.Attributes(F.text, F.carloc.pos, fnt, col, voff);
          IF fnt # NIL THEN Texts.ChangeLooks(text, beg, end, {0,1,2}, fnt, col, voff) END
        END
      END
    END
  END
END Edit;

PROCEDURE Handle* (F: Display.Frame; VAR M: Display.FrameMsg);
  VAR F1: Frame; buf: Texts.Buffer;
BEGIN
  CASE F OF Frame:
    CASE M OF
    Oberon.InputMsg:
      IF M.id = Oberon.track THEN Edit(F, M.X, M.Y, M.keys)
      ELSIF M.id = Oberon.consume THEN
        IF F.hasCar THEN Write(F, M.ch, M.fnt, M.col, M.voff) END
      END |
    Oberon.ControlMsg:
      IF M.id = Oberon.defocus THEN Defocus(F)
      ELSIF M.id = Oberon.neutralize THEN Neutralize(F)
      END |
    Oberon.SelectionMsg:
      GetSelection(F, M.text, M.beg, M.end, M.time) |
    Oberon.CopyMsg: Copy(F, F1); M.F := F1 |
    MenuViewers.ModifyMsg: Modify(F, M.id, M.dY, M.Y, M.H) |
    CopyOverMsg: CopyOver(F, M.text, M.beg, M.end) |
    UpdateMsg: IF F.text = M.text THEN Update(F, M) END
    END
  END
END Handle;

(*creation*)
```

```
  PROCEDURE Menu (name, commands: ARRAY OF CHAR): Texts.Text;
    VAR T: Texts.Text;
  BEGIN NEW(T); T.notify := NotifyDisplay;  Texts.Open(T, "");
    Texts.WriteString(W, name); Texts.WriteString(W, " | ");  Texts.WriteString(W, commands);
    Texts.Append(T, W.buf); RETURN T
  END Menu;

  PROCEDURE Text* (name: ARRAY OF CHAR): Texts.Text;
    VAR T: Texts.Text;
  BEGIN NEW(T); T.notify := NotifyDisplay; Texts.Open(T, name); RETURN T
  END Text;

  PROCEDURE NewMenu* (name, commands: ARRAY OF CHAR): Frame;
    VAR F: Frame; T: Texts.Text;
  BEGIN NEW(F); T := Menu(name, commands);
    Open(F, Handle, T, 0, Display.white, left DIV 4, 0, 0, 0, lsp); RETURN F
  END NewMenu;

  PROCEDURE NewText* (text: Texts.Text; pos: LONGINT): Frame;
    VAR F: Frame;
  BEGIN NEW(F);
    Open(F, Handle, text, pos, Display.black, left, right, top, bot, lsp); RETURN F
  END NewText;

BEGIN NEW(TBuf); NEW(DelBuf);
  Texts.OpenBuf(TBuf); Texts.OpenBuf(DelBuf);
  lsp := Fonts.Default.height; menuH := lsp + 2; barW := menuH;
  left := barW + lsp DIV 2;
  right := lsp DIV 2;
  top := lsp DIV 2; bot := lsp DIV 2;
  asr := Fonts.Default.maxY;
  dsr := -Fonts.Default.minY;
  selH := lsp; markW := lsp DIV 2;
  eolW := 0;   (*!*)
  ScrollMarker.Fade := FlipSM; ScrollMarker.Draw := FlipSM;
  Texts.OpenWriter(W); Texts.OpenWriter(KW)
END TextFrames.


(*---------------------------- Modified: MenuViewers.Mod ----------------------------------*)

MODULE MenuViewers; (*JG 26.8.90 / 16.9.93 / NW 10.3.2013 / AP 1.12.15 continuous refresh*)
  IMPORT Input, Display, Viewers, Oberon;

  CONST extend* = 0; reduce* = 1; Display.White = Display.white;

  TYPE Viewer* = POINTER TO ViewerDesc;

    ViewerDesc* = RECORD (Viewers.ViewerDesc)
      menuH*: INTEGER
    END;

    ModifyMsg* = RECORD (Display.FrameMsg)
      id*: INTEGER;
      dY*, Y*, H*: INTEGER
    END;

  PROCEDURE Copy (V: Viewer; VAR V1: Viewer);
    VAR Menu, Main: Display.Frame; M: Oberon.CopyMsg;
  BEGIN Menu := V.dsc; Main := V.dsc.next;
    NEW(V1); V1^ := V^; V1.state := 0;
    M.F := NIL; Menu.handle(Menu, M); V1.dsc := M.F;
    M.F := NIL; Main.handle(Main, M); V1.dsc.next := M.F
  END Copy;

  PROCEDURE Draw (V: Viewers.Viewer);
  BEGIN
    Display.ReplConst(Display.White, V.X, V.Y, 1, V.H, Display.replace);
    Display.ReplConst(Display.White, V.X + V.W - 1, V.Y, 1, V.H, Display.replace);
    Display.ReplConst(Display.White, V.X + 1, V.Y, V.W - 2, 1, Display.replace);
    Display.ReplConst(Display.White, V.X + 1, V.Y + V.H - 1, V.W - 2, 1, Display.replace)
  END Draw;

  PROCEDURE Extend (V: Viewer; newY: INTEGER);
    VAR dH: INTEGER;
  BEGIN dH := V.Y - newY;
    IF dH > 0 THEN
      Display.ReplConst(Display.black, V.X + 1, newY + 1, V.W - 2, dH, Display.replace);
      Display.ReplConst(Display.White, V.X, newY, 1, dH, Display.replace);
      Display.ReplConst(Display.White, V.X + V.W - 1, newY, 1, dH, Display.replace);
      Display.ReplConst(Display.White, V.X + 1, newY, V.W - 2, 1, Display.replace)
    END
```

```
  END Extend;

  PROCEDURE Reduce (V: Viewer; newY: INTEGER);
  BEGIN Display.ReplConst(Display.White, V.X + 1, newY, V.W - 2, 1, Display.replace)
  END Reduce;

  PROCEDURE Grow (V: Viewer; oldH: INTEGER);
    VAR dH: INTEGER;
  BEGIN dH := V.H - oldH;
    IF dH > 0 THEN
      Display.ReplConst(Display.White, V.X, V.Y + oldH, 1, dH, Display.replace);
      Display.ReplConst(Display.White, V.X + V.W - 1, V.Y + oldH, 1, dH, Display.replace);
      Display.ReplConst(Display.White, V.X + 1, V.Y + V.H - 1, V.W - 2, 1, Display.replace)
    END
  END Grow;

  PROCEDURE Shrink (V: Viewer; newH: INTEGER);
  BEGIN Display.ReplConst(Display.White, V.X + 1, V.Y + newH - 1, V.W - 2, 1, Display.replace)
  END Shrink;

  PROCEDURE Adjust (F: Display.Frame; id, dY, Y, H: INTEGER);
    VAR M: ModifyMsg;
  BEGIN M.id := id; M.dY := dY; M.Y := Y; M.H := H; F.handle(F, M); F.Y := Y; F.H := H
  END Adjust;

  PROCEDURE Restore (V: Viewer);
    VAR Menu, Main: Display.Frame;
  BEGIN Menu := V.dsc; Main := V.dsc.next;
    Oberon.RemoveMarks(V.X, V.Y, V.W, V.H);
    Draw(V);
    Menu.X := V.X + 1; Menu.Y := V.Y + V.H - 1; Menu.W := V.W - 2; Menu.H := 0;
    Main.X := V.X + 1; Main.Y := V.Y + V.H - V.menuH; Main.W := V.W - 2; Main.H := 0;
    IF V.H > V.menuH + 1 THEN
      Adjust(Menu, extend, 0, V.Y + V.H - V.menuH, V.menuH - 1);
      Adjust(Main, extend, 0, V.Y + 1, V.H - V.menuH - 1)
    ELSE Adjust(Menu, extend, 0, V.Y + 1, V.H - 2)
    END
  END Restore;

  PROCEDURE Modify (V: Viewer; Y, H: INTEGER);
    VAR Menu, Main: Display.Frame;
  BEGIN Menu := V.dsc; Main := V.dsc.next;
    IF Y < V.Y THEN (*extend*)
      Oberon.RemoveMarks(V.X, Y, V.W, V.Y - Y);
      Extend(V, Y);
      IF H > V.menuH + 1 THEN
        Adjust(Menu, extend, 0, Y + H - V.menuH, V.menuH - 1);
        Adjust(Main, extend, 0, Y + 1, H - V.menuH - 1)
      ELSE Adjust(Menu, extend, 0, Y + 1, H - 2)
      END
    ELSIF Y > V.Y THEN (*reduce*)
      Oberon.RemoveMarks(V.X, V.Y, V.W, V.H);
      IF H > V.menuH + 1 THEN
        Adjust(Main, reduce, 0, Y + 1, H - V.menuH - 1);
        Adjust(Menu, reduce, 0, Y + H - V.menuH, V.menuH - 1)
      ELSE
        Adjust(Main, reduce, 0, Y + H - V.menuH, 0);
        Adjust(Menu, reduce, 0, Y + 1, H - 2)
      END;
      Reduce(V, Y)
    END
  END Modify;

PROCEDURE Change (V: Viewer; X, Y: INTEGER; Keys: SET);
    VAR Menu, Main: Display.Frame;
      V1: Viewers.Viewer;
      keysum: SET; Y0, dY, H: INTEGER; inverted: BOOLEAN;
  BEGIN (*Keys # {}*)
    Menu := V.dsc; Main := V.dsc.next;
    Oberon.DrawMouseArrow(X, Y);
    Display.ReplConst(Display.white, V.X + 1, V.Y + V.H - 1 - V.dsc.H, V.W - 2, V.dsc.H, Display.invert);
    (*Y0 := Y;*) keysum := Keys; Input.Mouse(Keys, X, Y); inverted := TRUE;
    WHILE Keys # {} DO Y0 := Y;
      keysum := keysum + Keys;
      Oberon.DrawMouseArrow(X, Y); Input.Mouse(Keys, X, Y);
      IF Keys = {2} THEN
        IF inverted & (Y # Y0) THEN inverted := FALSE;
          Display.ReplConst(Display.white, V.X + 1, V.Y + V.H - 1 - V.dsc.H, V.W - 2, V.dsc.H, Display.invert);
        END;
        (*the following code was moved to here unchanged from the ELSE clause further below*)
        IF Y > Y0 THEN (*extend*) dY := Y - Y0;
          V1 := Viewers.Next(V);
```

```
        IF V1.state > 1 THEN
          CASE V1 OF
          Viewer:
            IF V1.H < V1.menuH + 2 THEN dY := 0
            ELSIF V1.H < V1.menuH + 2 + dY THEN dY := V1.H - V1.menuH - 2
            END |
          Viewers.Viewer: IF V1.H < 1 + dY THEN dY := V1.H - 1 END
          END
        ELSIF V1.H < dY THEN dY := V1.H
        END;
        Viewers.Change(V, V.Y + V.H + dY);
        Oberon.RemoveMarks(V.X, V.Y, V.W, V.H);
        Grow(V, V.H - dY);
        IF V.H > V.menuH + 1 THEN
          Adjust(Menu, extend, dY, V.Y + V.H - V.menuH, V.menuH - 1);
          Adjust(Main, extend, dY, V.Y + 1, V.H - V.menuH - 1)
        ELSE (*V.H > 1*)
          Adjust(Menu, extend, dY, V.Y + 1, V.H - 2);
          Adjust(Main, extend, dY, V.Y + V.H - V.menuH, 0)
        END;
      ELSIF Y < Y0 THEN (*reduce*) dY := Y0 - Y;
        IF V.H >= V.menuH + 2 THEN
          IF V.H < V.menuH + 2 + dY THEN dY := V.H - V.menuH - 2 END;
          Oberon.RemoveMarks(V.X, V.Y, V.W, V.H);
          H := V.H - dY;
          Adjust(Main, reduce, dY, V.Y + 1, H - V.menuH - 1);
          Adjust(Menu, reduce, dY, V.Y + H - V.menuH, V.menuH - 1);
          Shrink(V, H); Viewers.Change(V, V.Y + H)
        END
      END
    END
  END;
  IF inverted
    Display.ReplConst(Display.white, V.X + 1, V.Y + V.H - 1 - V.dsc.H, V.W - 2, V.dsc.H, Display.invert)
  END;
  IF ~(0 IN keysum) THEN
    IF 1 IN keysum THEN V1 := Viewers.This(X, Y);
      IF (V1 IS Viewer) & (Y > V1.Y + V1.H - V1(Viewer).menuH - 2) THEN Y := V1.Y + V1.H END;
      IF Y < V1.Y + V.menuH + 2 THEN Y := V1.Y + V.menuH + 2 END;
      Viewers.Close(V); Viewers.Open(V, X, Y); Restore(V)
    (*ELSE ...*) (*code moved from here unchanged to inside the WHILE Keys # {} loop above*)
    END
  END
END Change;

PROCEDURE Suspend (V: Viewer);
  VAR Menu, Main: Display.Frame;
BEGIN Menu := V.dsc; Main := V.dsc.next;
  Adjust(Main, reduce, 0, V.Y + V.H - V.menuH, 0);
  Adjust(Menu, reduce, 0, V.Y + V.H - 1, 0)
END Suspend;

PROCEDURE Handle* (V: Display.Frame; VAR M: Display.FrameMsg);
  VAR X, Y: INTEGER;
    Menu, Main: Display.Frame; V1: Viewer;
BEGIN Menu := V.dsc; Main := V.dsc.next;
    CASE M OF
    Oberon.InputMsg:
      IF M.id = Oberon.track THEN
        X := M.X; Y := M.Y;
        IF Y < V.Y + 1 THEN Oberon.DrawMouseArrow(X, Y)
        ELSIF Y < V.Y + V.H - V(Viewer).menuH THEN Main.handle(Main, M)
        ELSIF Y < V.Y + V.H - V(Viewer).menuH + 2 THEN Menu.handle(Menu, M)
        ELSIF Y < V.Y + V.H - 1 THEN
          IF 2 IN M.keys THEN Change(V(Viewer), X, Y, M.keys) ELSE Menu.handle(Menu, M) END
        ELSE Oberon.DrawMouseArrow(X, Y)
        END
      ELSE Menu.handle(Menu, M); Main.handle(Main, M)
      END |
    Oberon.ControlMsg:
      IF M.id = Oberon.mark THEN
        X := M.X; Y := M.Y; Oberon.DrawMouseArrow(X, Y); Oberon.DrawPointer(X, Y)
      ELSE Menu.handle(Menu, M); Main.handle(Main, M)
      END |
    Oberon.CopyMsg:
      Copy(V(Viewer), V1); M.F := V1 |
    Viewers.ViewerMsg:
      IF M.id = Viewers.restore THEN Restore(V(Viewer))
      ELSIF M.id = Viewers.modify THEN Modify(V(Viewer), M.Y, M.H)
      ELSIF M.id = Viewers.suspend THEN Suspend(V(Viewer))
      END |
    Display.FrameMsg: Menu.handle(Menu, M); Main.handle(Main, M)
```

```
      END
  END Handle;

  PROCEDURE New* (Menu, Main: Display.Frame; menuH, X, Y: INTEGER): Viewer;
    VAR V: Viewer;
  BEGIN NEW(V);
    V.handle := Handle; V.dsc := Menu; V.dsc.next := Main; V.menuH := menuH;
    Viewers.Open(V, X, Y); Restore(V); RETURN V
  END New;

END MenuViewers.
```