

```

%
% This file is a product of Sun Microsystems, Inc. and is provided for
% unrestricted use provided that this legend is included on all tape
% media and as a part of the software program in whole or part. Users
% may copy or modify this file without charge, but are not authorized to
% license or distribute it to anyone else except as part of a product
% or program developed by the user.
%
% THIS FILE IS PROVIDED AS IS WITH NO WARRANTIES OF ANY KIND INCLUDING THE
% WARRANTIES OF DESIGN, MERCHANTIBILITY AND FITNESS FOR A PARTICULAR
% PURPOSE, OR ARISING FROM A COURSE OF DEALING, USAGE OR TRADE PRACTICE.
%
% This file is provided with no support and without any obligation on the
% part of Sun Microsystems, Inc. to assist in its use, correction,
% modification or enhancement.
%
% SUN MICROSYSTEMS, INC. SHALL HAVE NO LIABILITY WITH RESPECT TO THE
% INFRINGEMENT OF COPYRIGHTS, TRADE SECRETS OR ANY PATENTS BY THIS FILE
% OR ANY PART THEREOF.
%
% In no event will Sun Microsystems, Inc. be liable for any lost revenue
% or profits or other special, indirect and consequential damages, even
% if Sun has been advised of the possibility of such damages.
%
% Sun Microsystems, Inc.
% 2550 Garcia Avenue
% Mountain View, California 94043
%
%
%      textcan.ps 1.5 88/01/14
%
%----- TextCanvas -----
%
%      Copyright (c) 1987 by Sun Microsystems, Inc.
%      Steve Isaac 12/18/87
%
% TextCanvas User's Guide
% -----
% Description:
%      TextCanvas is a NeWS class that provides a generic text storage and
%      display facility NeWS server. It has the following features:
%      . Scrolling
%      . Text caret
%      . Selections (integrated with system selections)
%      . Text may be larger than the display canvas
%      . Display canvas can be positioned anywhere over the stored text
%      . Font and point size may be changed dynamically
%      . Colors may be changed dynamically
%      . No limitations on width of text
%      . Fixed number of lines of text, specified at creation time
%      . Currently limited to fixed-width fonts
%
%      The TextCanvas allows text-based applications (terminal emulators,
%      text editors, text widgets) to be easily written. It is a start at
%      a common platform for NeWS text-oriented applications.
%
%      The following things need to be done to TextCanvas in order to fully
%      achieve this goal:

```

```

%
%      . Support for variable width fonts and simultaneous multiple fonts
%      (maybe a subclass if this causes too big a performance hit)
%      . Text attributes (ie. reversed, blinking, bold, etc).
%      . Separation of Text array and TextCanvas (to allow multiple
%        views onto the same text)
%      . Completion of selection service (multiple clicks, etc)
%      . Dynamic changing of the number of lines of text
%      . The caret should be a separate class
%      . The terminal emulation specifics (coordinate system, scrolling)
%        should be a separate subclass
%      . Continuing performance improvements
%
% Overview:
%   The TextCanvas can be viewed as a NeWS canvas that understands a set
%   of messages relating to text manipulation (eg. inserting a string,
%   deleting some lines, moving the caret). Basic canvas operations
%   (eg. shaping, resizing, moving) are done via standard NeWS primitives.
%   All canvas operations can be freely performed, however, the TextCanvas
%   must be informed if the canvas is resized, damaged, or destroyed.
%
%   A TextCanvas has an underlying fixed-size text array that contains
%   arbitrarily long strings. The canvas is essentially a viewport into this
%   array; this viewport may be moved around to display different parts of
%   the text array. It cannot be moved outside the boundaries of the
%   text array, however.
%
%   An instance of TextCanvas is created by specifying an existing canvas
%   and the number of lines that the Text array will store. This canvas
%   could be the client canvas of a window, which is how the TextCanvas
%   typically interfaces to the LiteWindow package.
%
% Coordinates:
%   The TextCanvas presents an integer coordinate system that addresses each
%   possible character position in the Text array. It is laid out as follows:
%
%
%      1,-N ----- W,-N  <-----+
%      |
%      |
%      |
%      |
%      |
%      1,-1          W,-1      +-- NumLines
%      1,0            W,0
%      1,1 ----- W,1
%      1,2            W,2
%      |
%      |
%      |
%      1,H ----- W,H  <-----+
%
%   where: W is the largest width of any string in the text array,
%          H is the number of lines that fit in the viewport canvas,
%          NumLines is the number of lines in the text array,
%          N is: NumLines - H - 1.
%
%   This means that the lines of the Text array under the original viewport
%   position are addressed starting at 1, with increasing line numbers going

```

```

%   down the viewport.
%
%   Note that W is always changing as new lines of text are written
%   into the Text canvas. The coordinate system will change if the
%   viewport canvas is resized.
%
% Scrolling:
%   Text which is written at line number H+1 or greater will cause
%   the entire Text array to scroll up so that the last line written
%   is at line H. Lines 0 through -N are therefore a transcript of
%   what was displayed in the viewport. Upwards scrolling can also be caused
%   by cursor movement (movecursordelta).
%
%   Downwards scrolling is limited to the original viewport area only. It is
%   caused by cursor movement (movecursordelta).
%
%   Scrolling can be limited to a portion of the text array (via
%   setscrollinglimits).
%
% Moving the viewport:
%   The viewport can be moved to display non-visible portions of the text
%   array. Note that the coordinate system does not change if the viewport
%   is moved. It remains relative to the original viewport position. Movement
%   is requested via the moveviewport message.
%
% Caret:
%   The TextCanvas contains a built-in text caret that can be turned
%   on or off, be any color, be any shape that can be expressed as a
%   PostScript path, made to blink at a user-defined speed and duty
%   cycle, and moved to any coordinate location. The caret is positioned
%   just to the left and below the location it is on.
%
% Writing text:
%   Text is put into the text array via the writelines, and writeatcaret
%   messages. They allow large blocks of text to be written in a single
%   message. There is a big performance win for writing as large
%   a block as possible. Text must consist of printable characters (ASCII
%   codes 0x20-0x7E) only. Non-printable characters will cause erroneous
%   selections. There is no special interpretation for control characters.
%
% Selections:
%   The TextCanvas provides a MacIntosh-like selection mechanism.
%   Left mouse button down sets the selection start point. Dragging the
%   mouse with the left button down drags out a selection. Left mouse
%   button up completes the selection. The middle mouse button allows
%   the current selection to be extended, either by clicking or by dragging.
%
%   The selected text is made the PrimarySelection in the system selection
%   dictionary. Any previous primary selection is cleared. The selected text
%   can be accessed via the standard system selection call (getselection).
%   If LiteUI is running then the "Put" function key will put the selection
%   on the shelf.
%
% Input Handling and Callback Routines:
%   The TextCanvas expresses interest in keyboard events. An optional
%   user-supplied callback routine (KeyHitCallback) is called whenever a
%   keystroke is detected. Similarly, another callback routine
%   (InsertValueCallback) is called when a selection /InsertValue

```

```

% event is detected.
%
% A callback routine for when the number of rows and columns in the
% viewport changes (ResizeCallback) is provided. Callbacks for mouse button
% events (LeftMouseDownCallback, LeftMouseUpCallback,
% MiddleMouseDownCallback, MiddleMouseUpCallback) are also provided. Other
% callbacks may be added by subclassing the text canvas.
%
% These callback routines make it very easy to write short routines
% do things like echo what the user has typed or move the caret to where
% the mouse was clicked.
%
% The callback routine will be executed within the context of the the
% TextCanvas instance. You may access all TextCanvas instance or class
% variables, or invoke methods by simply calling them as procedures.
%
% TextCanvas Interface Definition
% -----
% The messages and callback routines that class TextCanvas provides are:
%
% /changeFont { % fname fheight - => -
% --- Change the font and point size for all text. Either fname or fheight
% may be null, in which case they are ignored.
%
% /writeln { % arrayofstrings col row => -
% --- Write an array of strings, starting the first string at col,row
% with subsequent strings going at 1,row+1 1,row+2 etc.
%
% /writeatCaret { % arrayofstrings => -
% --- Similar to writeln, except start at the current caret location. The
% caret is moved to the next available character position when the
% write is done.
%
% /deleteString { % length col row => -
% --- Delete a string starting at col,row for length characters.
% length must be 0 or a positive integer.
%
% /insertLine { % numlines row => -
% --- Insert numlines blank lines, starting at line row.
% numlines must be 0 or a positive integer
%
% /deleteLine { % numlines row => -
% --- Delete numlines lines, starting at line row.
% numlines must be 0 or a positive integer
%
% /setScrollingLimits { % toprow bottomrow => -
% --- Sets the scrolling limits for the TextCanvas. All up or down scrolling
% will be limited to this region instead of affecting the entire Text
% canvas. When scrolling limits are set, those methods which can
% trigger scrolling (writeln, writeatCaret, moveCaretDelta) will
% only cause scrolling if they affect lines within the scrolling region.
% Any scrolling that is initiated will only move lines within the
% scrolling limits.
%
% /removeScrollingLimits { % - => -
% --- Removes any scrolling bounds set by setScrollingLimits.
%
% /clearViewport { % - => -

```

```

%    --- Clear the text and screen area of the viewport
%
% /flashviewport { % - => -
%    --- Flash the contents of the viewport (visible bell)
%
% /moveviewport { % x y => -
%    --- Move the viewport to another part of the underlying Text array.
%    x and y must be between 0 and 1. This represents a percentage of the
%    current total width or height of the Text array. Either argument
%    can be null, in which case it is ignored.
%
% /getviewportsize { % - => col rows xpixels ypixels
%    --- Return the number of columns, rows, pixel height and width of
%    the viewport.
%
% /getlinelength { % row => length
%    --- Return the current length of the line at row.
%
% /calcareas { % pixwidth pixheight => numcols numrows
%    --- Returns the number of rows and columns that will fit into the pixel
%    area specified by pixwidth and pixheight, given the current Font and
%    point size.
%
% /calcpixarea { % numcols numrows => pixwidth pixheight
%    --- Returns the minimum pixel area required to display numrows and numcols,
%    given the current Font and point size.
%
% /oncaret { % - => -
%    --- Turn the caret on.
%
% /offcaret { % - => -
%    --- Turn the caret off.
%
% /movecaret { % col row => -
%    --- Move the caret to an absolute position. col and row are integers. Scrolling
is    never triggered.
%
% /movecaretdelta { % deltax deltay => -
%    --- Move the caret relative to its current position. deltax and deltay must be
%    integers (negatives allowed). Scrolling is triggered if deltay moves the
caret    outside of the scrolling limits. If no scrolling limits are set, scrolling
is    triggered if deltay moves the caret outside of the original viewport
region.
%
% /setcaretblink { % blink-rate duty-cycle => -
%    --- Set the caret blink rate and the blink duty cycle. blink-rate is
%    in seconds and represents a complete on/off cycle. duty-cycle is
%    between 0 and 1, and represents the percentage of on time.
%
% /setcaretcolor { % color => -
%    --- Set the current caret color. color is a color object.
%
% /setcaretshape { % shapename => successful?
%    --- Set the caret shape. shapename is an entry in the CaretShapeDict.
%    Return a boolean that tells whether shapename was found.

```

```

%
% /getcaretpos { % - => col row
% --- Return the current caret position
%
% /setbgcolor { % color => -
% --- Set the current canvas background color
%
% /setfgcolor { % color => -
% --- Set the current text color
%
% /fixdamage { % - => -
% --- Damage handler; goes in the PaintClient window callback routine
%
% /new { % numrows can => object
% --- Create a new instance of the TextCanvas. numrows is the number
% of lines to be allocated in the Text array; it is fixed for
% the life of the instance. can is the viewport canvas.
%
% /reshape { % - => -
% --- This method must be called whenever the viewport canvas has changed
% size. It updates the number of rows and columns in the TextCanvas,
% repositions the caret to be as close to its old position as possible,
% resets the scrolling region, and moves the viewport to its original
% position.
%
% /destroy { % - => -
% --- Destroy the TextCanvas. This must be called if the viewport canvas
% is ever destroyed.
%
% --- Client callback routines.
% /ResizeCallback nullproc def % { - => -
% --- ResizeCallback is called whenever the number of rows and columns
% changes.
% /KeyHitCallback nullproc def % { keyvalue - => -
% --- KeyHitCallback is called whenever a keyboard input event happens
% /InsertValueCallback nullproc def % { insertstring => -
% --- InsertValueCallback is called whenever an InsertValue event happens
% /LeftMouseDownCallback nullproc def % { col row => -
% --- LeftMouseDownCallback is called when the left mouse button goes down.
% /LeftMouseUpCallback nullproc def % { col row => -
% --- LeftMouseUpCallback is called when the left mouse button goes up.
% /MiddleMouseDownCallback nullproc def % { col row => -
% --- MiddleMouseDownCallback is called when the middle button goes down.
% /MiddleMouseUpCallback nullproc def % { col row => -
% --- MiddleMouseUpCallback is called when the middle button goes up.
%
% Implementation Details
% -----
% Coordinates:
% The text canvas has an internal coordinate system as follows:
%
% 1,1 ----- TextWidth,1
% |
% |
% |
% 1,TextHeight TextWidth,TextHeight
%
% The Can canvas is essentially a viewport on this coordinate system. The
% TM tranformation matrix reflects this coordinate system, and is used for

```

```

%      computing caret or text movement on Can. The base window lives in the
%      lower left hand corner of the text canvas, the original location of the
%      viewport. It is the same size as the Can canvas. External caret
%      coordinates are relative to the base window; the internal caret
%      coordinates (CaretX, CaretY) are relative to the internal coordinate
%      system. The text array is laid out in a similar fashion to the internal
%      coordinate system; however, indices are 0 based. The Xindex and Yindex
%      functions map coordinate values to array indices. Text scrolling is
%      handled by manipulating this mapping.
%
systemdict /TextCanvas known not {systemdict begin
/TextCanvas Object
dictbegin
    /DEBUG                false      def % Turn on debugging output
    /Text                  null       def % Main text array
    /TextWidth             0          def % Number of text columns (changes
                                         % dynamically)
    /TextHeight            0          def % Number of text rows (specified
                                         % at initialization)
    /Can                   null       def % The main canvas
    /SelectionCan          null       def % Visible selection feedback canvas
    /SelDragCan            null       def % Transparent selection drag canvas
    /Caret                 null       def % The caret canvas

    /CanPixWidth           0          def % Canvas width (pixels)
    /CanPixHeight          0          def % Canvas height (pixels)
    /CanPixX               0          def % Canvas X origin (pixels)
    /CanPixY               0          def % Canvas Y origin (pixels)
    /CanX                  0          def % Canvas X origin (Text coordinates)
    /CanY                  0          def % Canvas Y origin (Text coordinates)
    /CanWidth              0          def % Number of columns in canvas
    /CanHeight             0          def % Number of rows in canvas

    % --- Caret variables.
    /CaretOn?              false      def % Is the caret on?
    /CaretInactive?        false      def % Is the caret inactive (shaded,
                                         % no blink)?
    /CaretSupressed?       false      def % Is the caret supressed
                                         % (temporarily off)?
    /NextMoveTime          0          def % Time at which to do the next
                                         % caret move
    /DelayedMoveProc        null       def % Delayed caret move timer process
    /CaretX                 1          def % Caret column in canvas
    /CaretY                 1          def % Caret row in canvas
    /CaretShape             /TrianglePlus def % Current caret shape (from
                                         % CaretShapeDict)
    /CaretColor             null       def % Current caret color
    /CaretBlinkEnabled?     true       def % Are we blinking?
    /CaretBlinkTime         1.0       def % Seconds
    /CaretDutyCycle         0.8       def % Percentage on
    /CaretDelayTime         .06       def % Caret move delay time (seconds)

    /EventManager           null       def % The main event manager
    /Interests              null       def % Main event manager interests
    /MouseDownEventManager  null       def % Event manager for mouse dragging
    /DragInterests          null       def % Drag event manager interests
    /KeyboardEventManager   null       def % Keyboard/Insert_Value event mgr

```

```

% --- Selection variables.
/MouseDownX      0      def % Where MouseDown actually happened
/MouseDownY      0      def
/SelectionX      1      def % Current initial selection point
/SelectionY      1      def
/SelectionX1     1      def % Current ending selection point
/SelectionY1     1      def
/SelExtendTop?   false  def % Extend the top of the selection
/SelectionOn?    false  def % Is the selection visible?
/SelectionPath   null   def % Current path of the visible
                        % selection

/SelectionDict 10 dict dup begin      % Dictionary for i/f to system
                                        % selections
    /ContentsAscii null               def
    /SelectionObjSize 1                def
    /SelectionResponder null           def
end def

/ViewportXdelta  0      def % Viewport offset adjustment
/ViewportYdelta  0      def % Viewport offset adjustment
/WriteInProgress? false  def % Is there text output happening?
/BotScrollLimit  0      def % Scrolling limit for bottom of screen
/TopScrollLimit  0      def % Scrolling limit for top of screen
/ScrollRegionLength 0    def % Number of lines in scrolling region
/ScrollLimitOn?  false  def % Are scrolling limits in effect?
/BaseY           0      def % Base window Y position in Text
/PixColWidth     0      def % Row width (pixels)
/PixRowHeight    0      def % Column height (pixels)
/TM              null   def % Position tranform matrix
/Font            null   def % Current font
/FontDescentTM   null   def % TM plus font descent
/MapOffset       0      def % Y array index offset
/InputBuffer     null   def % Input line buffer.
/InputBufferLine 0      def % Line that input buffer is on
/InputBufferLength 0    def % Number of characters in the buffer
/BgColor         1 1 1 rgbcolor def % Current background color
/FgColor         0 0 0 rgbcolor def % Current foreground color
/KeyboardInterest null   def % Need to keep this so we can revoke
                        % it at destroy time to free memory

% --- Client callback routines.
% --- ResizeCallback is called whenever the number of rows and columns
% changes.
/ResizeCallback  nullproc def % { - => -
% --- KeyHitCallback is called whenever a keyboard input event happens
/KeyHitCallback  nullproc def % { keyvalue - => -
% --- InsertValueCallback is called whenever an InsertValue event happens
/InsertValueCallback nullproc def % { insertstring => -
% --- LeftMouseDownCallback is called when the left mouse button goes down.
/LeftMouseDownCallback nullproc def % { col row => -
% --- LeftMouseUpCallback is called when the left mouse button goes up.
/LeftMouseUpCallback nullproc def % { col row => -
% --- MiddleMouseDownCallback is called when the middle button goes down.
/MiddleMouseDownCallback nullproc def % { col row => -
% --- MiddleMouseUpCallback is called when the middle button goes up.
/MiddleMouseUpCallback nullproc def % { col row => -

/FontName      /Screen      def

```



```

    /FontHeight      14          def
dictend
classbegin
    /LF              10          def
    /CR              13          def
    /BLANK           32          def

```

```

/DefaultColorCaret  1 0 0 rgbcolor  def
/DefaultMonoCaret   0 0 0 rgbcolor  def
/DefaultInactiveColor ColorDisplay?
                    {.75 .75 .75 rgbcolor}
                    {.5 .5 .5 rgbcolor}
                    ifelse          def

```

```

%----- Utilities -----

```

```

/?def {
    currentdict 2 index known {
        pop pop
    }{
        def
    } ifelse
} def

```

```

/LoadCaretShapeDict {
systemdict /CaretShapeDict known not {
    systemdict begin /CaretShapeDict dictbegin dictend def end
} if
CaretShapeDict begin % --- Caret Shape dictionary

```

```

    /HLine { % xscale yscale => {path}
        gsave
            dup scale
            pop
            0 0 moveto
            0 .8 transform round exch round exch itransform rlineto
            -0.3 0 transform round exch round exch itransform rlineto
            0 -1 transform round exch round exch itransform rlineto
            currentpath
        grestore
        setpath
    } ?def

```

```

/Diamond { % xscale yscale => {path}
    gsave
        dup scale
        pop
        0 0 moveto 0.25 0 rmoveto 0.25 0.25 rlineto
        -0.25 0.25 rlineto -0.25 -0.25 rlineto closepath
        currentpath
    grestore
    setpath
} ?def

```

```

/TrianglePlus { % xscale yscale => {path}
    gsave
        dup scale
        pop
        0 0 moveto

```

```

    0 .8 transform round exch round exch itransform rlineto
    -0.1 0 transform round exch round exch itransform rlineto
    0 -.8 transform round exch round exch itransform rlineto
    -0.35 -0.4 transform round exch round exch itransform rlineto
    .35 0 transform round exch round exch itransform rlineto
    .1 0 transform round exch round exch itransform rlineto
    .35 0 transform round exch round exch itransform rlineto
    closepath
    currentpath
  grestore
  setpath
} ?def

/Triangle { % xscale yscale => {path}
  gsave
    dup scale
    pop
    0 0 moveto
    -0.3 -0.6 transform round exch round exch itransform rlineto
    .6 0 transform round exch round exch itransform rlineto
    -0.3 0.6 transform round exch round exch itransform rlineto
    currentpath
  grestore
  setpath
} ?def

/Box { % xscale yscale => {path}
  gsave
    scale
    0 0 moveto
    0 1 rlineto
    1 0 rlineto
    0 -1 rlineto
    -1 0 rlineto
    -.1 -.1 rmoveto
    0 1.2 rlineto
    1.2 0 rlineto
    0 -1.2 rlineto
    closepath
    currentpath
  grestore
  setpath
} ?def
end
} def

/Xindex { % col => x-index
% --- Convert a column coordinate into a Text array index
  1 sub
} def

/Yindex { % row => y-index
% --- Convert a row coordinate into a Text array index
% Text array
  1 sub MapOffset add TextHeight mod
} def

/CreateInterests { % - => -

```

```

% --- Main event handler interests
/Interests dictbegin

/CaretDamageEvent
/Damaged
{pop gsave
  Caret setcanvas
  CaretInactive? {
    DefaultInactiveColor fillcanvas
  }{
    CaretColor fillcanvas
  } ifelse
  grestore }
null Caret eventmgrinterest
def

/CaretTimerEvent
% --- Caret blink events. Send this event out again with the time
%   of the next blink
/CaretTimer
{/e exch def
  e begin
  Caret /Mapped get {
    CaretBlinkEnabled? CaretOn? CaretInactive? not CaretSupressed? not
    and and and {
      UnMapCaret
      /TimeStamp
      % --- When to turn caret back on
      CaretBlinkTime 60 div 1 CaretDutyCycle
      sub mul currenttime add
      def
    }{
      % --- If the caret is disabled, keep the timer event
      %   circulating at a 2 second rate
      /TimeStamp currenttime 1 30 div add def
    } ifelse
  }{
    CaretBlinkEnabled? CaretOn? CaretInactive? not CaretSupressed? not
    and and and {
      MapCaret
      /TimeStamp
      % --- When to turn caret back off
      CaretBlinkTime 60 div CaretDutyCycle mul
      currenttime add
      def
    }{
      % --- If the caret is disabled, keep the timer event
      %   circulating at a 2 second rate
      /TimeStamp currenttime 1 30 div add def
    } ifelse
  } ifelse
  e sendevent
end}
null Caret eventmgrinterest
def

/LeftMouseDownEvent
/LeftMouseButton

```

```

{begin
  InactivateCaret
  % --- Clear anyone else's primary selection
  SendClearSelection
  % --- Synchronously clear my primary selection
  ClearMySelection
  Canvas setcanvas
  TM setmatrix
  /MouseDownX XLocation 1 max round store
  /SelectionX MouseDownX store
  /SelectionX1 SelectionX store
  /MouseDownY YLocation TextHeight min CanY max round store
  /SelectionY MouseDownY store
  /SelectionY1 SelectionY store
  /len Text SelectionY Yindex get length store
  SelectionX len 2 add gt {
    /SelectionX len 2 add store
  } if
  /SelExtendTop? false def
  /MouseDownX MouseDownY BaseY sub LeftMouseDownCallback
end }
/DownTransition Can eventmgrinterest
def

/MiddleMouseDownEvent
/MiddleMouseButton
{begin
  InactivateCaret
  % --- Clear anyone else's primary selection
  SendClearSelection
  % --- Remove any visual feedback for my selection, but leave
  %       the selection path intact so we can extend it.
  false DrawSelection
  SelDragCan setcanvas
  TM setmatrix
  YLocation SelectionY sub abs dup mul
    XLocation SelectionX sub abs dup mul add
    YLocation SelectionY1 sub abs dup mul
    XLocation SelectionX1 sub abs dup mul add lt {
      /SelectionX XLocation 1 max round store
      /SelectionY YLocation TextHeight min CanY max round store
      /SelExtendTop? true store
    }{
      /SelectionX1 XLocation 1 max round store
      /SelectionY1 YLocation TextHeight min CanY max round store
      /SelExtendTop? false store
    } ifelse
  ExtendSelection
  /MouseDownX MouseDownY BaseY sub LeftMouseDownCallback
end }
  DragInterests forkeventmgr
store
XLocation 1 max round YLocation TextHeight min CanY max round BaseY
sub
  MiddleMouseDownCallback
end}

```

```

        /DownTransition Can eventmgrinterest
def

    dictend store % Interests
} def

/CreateDragInterests { % - => -
% --- Interests for mouse drag event manager
    /DragInterests dictbegin
        /MouseDownEvent
        /MouseDragged
        { begin
            SelDragCan setcanvas
            TM setmatrix
            SelExtendTop? {
                /SelectionX XLocation 1 max round store
                /SelectionY YLocation TextHeight min CanY max round store
            }{
                /SelectionX1 XLocation 1 max round store
                /SelectionY1 YLocation TextHeight min CanY max round store
            } ifelse
            erasepage
            ExtendSelection
        end}
    null Can eventmgrinterest
def

/LeftMouseUpEvent
/LeftMouseButton
{begin
    SelDragCan setcanvas
    erasepage
    Can setcanvas
    TM setmatrix
    % --- If we are at the same location as LeftButton down, then
    % remove any selection on our canvas. Otherwise, make the selected
    % area the primary selection.
    MouseDownX XLocation 1 max round eq
    MouseDownY YLocation TextHeight min CanY max round eq and {
        false DrawSelection
        /SelectionPath null store
    }{
        % --- SelectionX,Y must always be lower than SelectionX1,Y1
        SelectionY1 SelectionY lt SelectionY1 SelectionY eq
        SelectionX1 SelectionX lt and or {
            SelectionX SelectionY
            /SelectionX SelectionX1 store
            /SelectionY SelectionY1 store
            /SelectionY1 exch store
            /SelectionX1 exch store
        } if
        SelectionDict /ContentsAscii GetSelection put
        SelectionDict /Canvas Can put
        SelectionDict /SelectionHolder KeyboardEventMgr put
        SelectionDict /PrimarySelection setselection
        true DrawSelection
    } ifelse
    ReactivateCaret
}

```

```

        XLocation 1 max round YLocation TextHeight min CanY max round BaseY
sub
LeftMouseUpCallback
    MouseDragEventMgr killprocess
end}
/UpTransition null eventmgrinterest
def

/MiddleMouseUpEvent
/MiddleMouseButton
{begin
    SelDragCan setcanvas
    TM setmatrix
    erasepage
    % --- SelectionX,Y must always be lower than SelectionX1,Y1
    SelectionY1 SelectionY lt SelectionY1 SelectionY eq
        SelectionX1 SelectionX lt and or {
            SelectionX SelectionY
            /SelectionX SelectionX1 store
            /SelectionY SelectionY1 store
            /SelectionY1 exch store
            /SelectionX1 exch store
        } if
    SelectionDict /ContentsAscii GetSelection put
    SelectionDict /Canvas Can put
    SelectionDict /SelectionHolder KeyboardEventMgr put
    SelectionDict /PrimarySelection setselection
    true DrawSelection
    ReactivateCaret
    XLocation round YLocation TextHeight min CanY max round BaseY sub
MiddleMouseUpCallback
    MouseDragEventMgr killprocess
end}
/UpTransition null eventmgrinterest
def
    dictend store
} def

/KeyboardHandler { % - => -
    % --- Handler for keyboard, InsertValue, and Deselect events
/KeyboardInterest [
    Can addkbdinterests aload pop
    Can addselectioninterests aload pop
    % Get rid of LiteUI's mouse interests
    revokeinterest
    Can addfunctionstringsinterest
] def
{ awaitevent begin
    Name type /integertype eq {
        Name /KeyHitCallback self send
    } if
    Name /DeSelect eq {
        Action /PrimarySelection eq {
            false DrawSelection
            /SelectionPath null store
        } if
        Action /InputFocus eq {
            InactivateCaret

```

```

        } if
    } if
    Name /RestoreFocus eq {
        ReactivateCaret
    } if
    Name /InsertValue eq {
        Action /InsertValueCallback self send
    } if
    Name /Ignore eq {
    } if
end
} loop
} def

/InitFont { % - => -
% --- Initialize the current font and font metrics
10 dict begin
    /Font FontName findfont FontHeight scalefont store
    gsave
        false setprintermatch
        Font setfont (m) stringwidth pop /PixColWidth exch store
    grestore
    /PixRowHeight Font fontheight store
end
} def

/Reshape { % firsttime? => -
% --- Reshape the TextCanvas. This is where all initialization happens.
%     firsttime is true the first time the TextCanvas is reshaped;
%     false otherwise.
%
% --- Note: we are not enclosing this proc in a '10 dict begin end'
%     because the event handlers must be started with the class dict
%     being first on the dictionary stack. This results in firsttime?
%     being put into the instance dictionary.
/firsttime? exch def
% --- Take down the caret and clear any selection that is up
firsttime? {
    LoadCaretShapeDict
    InactivateCaret
    /InputBuffer 1024 string def      % Set input line buffer string to a
                                      % reasonable size. The buffer will
                                      % be grown dynamically if needed
}
    SupressCaret
} ifelse
ClearMySelection
gsave
    Can setcanvas
    6 array identmatrix setmatrix
    % --- Set up transformation matrix with font descent at the baseline
    0 TextHeight PixRowHeight mul
        Font fontascent add Font fontdescent sub
    translate
    PixColWidth PixRowHeight neg scale
    /FontDescentTM 6 array currentmatrix store
    % --- Set up transformation matrix for direct mapping of Text coords
    6 array identmatrix setmatrix

```

```

    0 TextHeight PixRowHeight mul
      Font fontascent add
      translate
    PixColWidth PixRowHeight neg scale
    /TM 6 array currentmatrix store
grestore
% --- Initialize the viewport and caret positions. Set the caret to
%   1,1 the first time around, try to maintain previous caret
%   position subsequently
firsttime? {
    % --- Determine the number of rows and columns in this canvas
    /CanWidth CanPixWidth PixColWidth idiv 1 sub store
    /CanHeight CanPixHeight PixRowHeight idiv 1 sub store
    % --- Initialize the position of the canvas viewport and the caret
    /CanX 1 store
    /CanY TextHeight CanHeight sub 1 add store
    /BaseY CanY 1 sub store
    /CaretX CanX store
    /CaretY CanY store
}
    /CaretX CaretX ViewportXdelta add store
    /CaretY CaretY ViewportYdelta add store
    /ViewportXdelta 0 store
    /ViewportYdelta 0 store
    % --- Remove any scrolling offset from caret position
    /CaretX CanX 1 sub CaretX add store
    /CaretY CanY TextHeight CanHeight sub 1 add sub CaretY add store
    % --- Determine the number of rows and columns in this canvas
    /CanWidth CanPixWidth PixColWidth idiv 1 sub store
    /CanHeight CanPixHeight PixRowHeight idiv 1 sub store
    % --- Initialize the position of the canvas viewport and the caret
    /CanX 1 store
    /CanY TextHeight CanHeight sub 1 add store
    /BaseY CanY 1 sub store
    % --- Check if the caret is out of bounds
    CaretY CanY lt {
        /CaretY CanY store
    } if
    CaretX CanWidth gt {
        /CaretX CanWidth store
    } if
} ifelse
% --- Reset scrolling limits
/BotScrollLimit TextHeight store
/TopScrollLimit CanY store
/ScrollRegionLength BotScrollLimit TopScrollLimit sub 1 add def
/ScrollLimitOn? false store
% --- Set up the text arrays
firsttime? {
    /Text TextHeight array store
    % --- Initialize Text array to empty strings
    0 1 TextHeight 1 sub {
        Text exch () put
    } for
    % --- Initialize the input buffer to blanks
    0 1 InputBuffer length 1 sub {
        InputBuffer exch BLANK put
    } for

```



```

} if
% --- Create the caret if needed
Caret null eq {
  CreateCaret
  % --- Kick off first blink event
  createevent begin
    /Canvas      Caret      def
    /Name        /CaretTimer def
    /Action      null       def
    /TimeStamp
      CaretBlinkTime 60 div CaretDutyCycle mul currenttime add
    def
    currentdict
  end
  sendevent
} if
% --- Create interests and event managers if they aren't running.
%   Note: this must be done with the class instance variable being
%   the first thing on the dictionary stack; otherwise the event
%   managers won't share the class' instance variables!
EventManager null eq {
  CreateInterests
  CreateDragInterests
  /EventManager Interests forkeventmgr def
  /KeyboardEventManager {KeyboardHandler} fork def
} if
%SelectionCan null eq {
  % --- Create the selection feedback canvas
  %   XXX - Not using the selection canvas yet; still doing xor
  %/SelectionCan Can newcanvas store
  %SelectionCan begin
  %   /Transparent false def
  %   /EventsConsumed /NoEvents def
  %end
%} if
% --- Shape the viewport canvas
gsave
Can setcanvas
% --- Clear the canvas
BgColor fillcanvas
% --- Make the canvas size be an even number of rows and cols
CanPixX CanPixY CanWidth PixColWidth mul PixColWidth add
CanHeight PixRowHeight mul PixRowHeight add Font fontdescent 2 mul sub
rectpath
% --- Set the default matrix for the canvas to the identity matrix
6 array identmatrix setmatrix
Can reshapecanvas
% --- Create the outline selection drag canvas
/SelDragCan Can createoverlay store
grestore
/SelectionOn? false def
/SelectionPath null def
UnSuppressCaret
MoveCaret
} def

/ClearScreenArea { % x y width height => -
% --- Fill the designated area with the background color

```

```

gsave
  Can setcanvas
  FontDescentTM setmatrix
  4 -2 roll          % width height x y
  1 sub              % width height x y-1
  4 2 roll           % x y-1 width height
  rectpath
  BgColor setcolor fill
grestore
} def

/MoveScreenArea { % numlines x y width height => -
% --- Move a given area of the screen numlines
%   numlines < 0 - move down
%   "           > 0 - move up
  10 dict begin
    /height exch def
    /width exch def
    /y exch def
    /x exch def
    /numlines exch def
    gsave
      Can setcanvas
      FontDescentTM setmatrix
      x y 1 sub width height rectpath
      0 numlines neg copyarea
    grestore
  end
} def

/FlushInputBuffer { % - => -
% --- Flush the InputBuffer to Text array if it is in use
InputBufferLine 0 ne {
  % --- Create a standalone string for this line in the Text array
  Text InputBufferLine Yindex
  Text InputBufferLine Yindex get InputBufferLength string copy
  put
  % --- Blank fill the previously used part of InputBuffer
  0 1 InputBufferLength {
    InputBuffer exch BLANK put
  } for
  /InputBufferLine 0 store
  /InputBufferLength 0 store
} if
} def

/ScrollUp { % numlines beginrow endrow => -
% --- Scroll up numlines from a given line to another line
  10 dict begin
    /endrow exch def
    /beginrow exch def
    /numlines exch def
    FlushInputBuffer
    ClearMySelection
    /len endrow beginrow sub 1 add def
    /numlines numlines len min def
    /inset beginrow numlines add def
    % --- Move the text

```

```

beginrow Yindex endrow Yindex le {
  % --- We can do a fast move if the scroll region doesn't
  %       wrap around the end of the physical array
  Text
    beginrow Yindex Text inset Yindex endrow inset sub 1 add
    getinterval putinterval
} {
  % XXX This should be done as two getinterval/putintervals
  beginrow numlines add 1 endrow {
    /i exch def
    Text i numlines sub Yindex Text i Yindex get put
  } for
} ifelse
endrow numlines sub 1 add 1 endrow {
  Text exch Yindex () put
} for
numlines 1 beginrow numlines add TextWidth len numlines sub MoveScreenArea
1 endrow numlines sub 1 add TextWidth numlines ClearScreenArea
end
} def

/ScrollDown { % numlines beginrow endrow => -
% --- Scroll down numlines from a given line to another line
10 dict begin
  /endrow exch def
  /beginrow exch def
  /numlines exch def
  FlushInputBuffer
  ClearMySelection
  /len endrow beginrow sub 1 add def
  /numlines numlines len min def
  /inset endrow numlines sub def
  beginrow Yindex endrow Yindex le {
    % --- We can do a fast move if the scroll region doesn't
    %       wrap around the end of the physical array
    Text
      beginrow numlines add Yindex Text beginrow Yindex
      inset beginrow sub 1 add getinterval putinterval
  } {
    % XXX This should be done as two getinterval/putintervals
    endrow -1 beginrow numlines add {
      /i exch def
      Text i Yindex Text i numlines sub Yindex get put
    } for
  } ifelse
  beginrow 1 beginrow numlines add 1 sub {
    Text exch Yindex () put
  } for
  numlines neg 1 beginrow TextWidth len numlines sub MoveScreenArea
  1 beginrow TextWidth numlines ClearScreenArea
end
} def

/RollAllTextUp { % numlines => -
% --- Scroll the entire Text Array up numlines, adding blank lines at the
%       bottom
10 dict begin
  /numlines exch def

```

```

    FlushInputBuffer
    ClearMySelection
    1 1 numlines { pop
        /MapOffset MapOffset 1 add TextHeight mod store
        Text MapOffset () put
    } for
end
} def

/DrawText { % x y w h => -
% --- Draw the text within the specified rectangle
10 dict begin
    /h exch def
    /w exch def
    /y exch def
    /x exch def
    gsave
        false setprintermatch
        Can setcanvas
        TM setmatrix
        Font setfont
        FgColor setcolor
        % --- Use the clip path to get x clipping
        x CanY 1 sub w CanHeight 1 add rectpath clip newpath
        y 1 y h add 1 sub {
            /i exch def
            1 i moveto
            6 array identmatrix setmatrix
            Text i Yindex get show
            TM setmatrix
        } for
    grestore
end
} def

/WriteLines { % stringarray insertmode? col row => - newcol newrow
% Put lines into the text buffer and display them on screen.
% stringarray is an array of lines to be displayed. Lines must
% contain printable characters only. col,row specify the starting point
% of the lines. The col,row of the next available text position are
% returned. insertmode? specifies whether to overwrite existing text
% or to insert each new line into the existing text.
10 dict begin
    /row exch def
    /col exch def
    /insertmode? exch def
    /lines exch def

    DEBUG {
        console (WriteLines: row: % col: % numlines: %\n) [row col lines length]
fprintf
        0 1 lines length 1 sub { /i exch def
            console (%\n) [lines i get] fprintf
        } for
        console flushfile
    } if
    SupressCaret
    /WriteInProgress? true store

```

```

/numlines lines length def
% --- Clear the current selection
% XXX This should be more selective; only clear if overwriting
ClearMySelection
% --- Do one line case as fast as possible
numlines 1 eq {
  lines col row WriteOneLine
}
% --- Do any text throw-away required, either due to scrolling
%   region or exceeding the basic capacity of the Text array.
ScrollLimitOn? {
  /numscroll 0 def
  % --- Note: No need to do anything if we are completely above
  %   the scrolling region.

  % --- Are we starting within the scrolling region?
  row TopScrollLimit ge row BotScrollLimit le and {
    % --- Get rid of everything that won't fit in the scroll region
    numlines ScrollRegionLength gt {
      /lines lines numlines ScrollRegionLength sub
      ScrollRegionLength getinterval def
      /numlines ScrollRegionLength def
      /col 1 def
    } if
    % --- Determine number of lines to scroll
    /numscroll numlines BotScrollLimit row sub 1 sub def
    % --- Adjust the starting row, if needed
    row numlines add 1 sub BotScrollLimit gt {
      /row BotScrollLimit numlines 1 sub sub def
    } if
  }
% --- Are we starting above the scrolling region and extending
%   into it?
row TopScrollLimit lt
  row numlines add TopScrollLimit gt and {
    % --- Write out the portion of the text that is above the
    %   scrolling region by recursively calling WriteLines.
    /abovescroll TopScrollLimit row sub def
    lines 0 abovescroll getinterval insertmode? col row WriteLines
    % --- Get rid of what we just wrote out. This will make us
    %   eligible for the "starting within the scrolling region"
    %   case.
    /lines lines abovescroll numlines abovescroll sub
    getinterval def
    /row TopScrollLimit def
    /col 1 def
    /numlines lines length def
  } if
  % --- Are we starting below the scrolling region?
  row BotScrollLimit gt {
    % --- Get rid of everything but the last line
    /lines lines numlines 1 sub 1 getinterval def
    numlines 1 ne {
      /col 1 def
    } if
    /numscroll numlines 1 sub def
    /numlines 1 def
  } if
} if

```

```

    } ifelse
    % --- Move existing text up, if necessary
    numscroll 0 gt {
        numscroll TopScrollLimit BotScrollLimit ScrollUp
    } if
}{ % --- No scrolling limits set
% --- We can handle a max of TextHeight lines. Throw everything
%     else away.
numlines TextHeight gt {
    /lines lines numlines TextHeight sub
    TextHeight getinterval def
    /numlines TextHeight def
} if
% --- Scroll up the text array, if needed
/numscroll numlines TextHeight row sub sub 1 sub def
numscroll 0 gt {
    numscroll RollAllTextUp
    /row row numscroll sub def
    numscroll 1 1 TextWidth TextHeight MoveScreenArea
    col ViewportXdelta sub row ViewportYdelta sub TextWidth
    numscroll 1 add ClearScreenArea
    % --- If the viewport is off its original position, fill in text
    ViewportYdelta 0 ne {
        col ViewportXdelta sub row ViewportYdelta sub TextWidth
        numscroll 1 add DrawText
        % --- Clear out any partial rows or columns
        1 CanY CanHeight add TextWidth 1 ClearScreenArea
        CanX 1 sub CanY 1 TextHeight ClearScreenArea
    } if
} if
} ifelse
col 1 ne {
    lines 0 1 getinterval col row WriteOneLine /row exch def pop
    lines 1 numlines 1 sub getinterval row 1 add WriteManyLines
}{
    lines row WriteManyLines
} ifelse
} ifelse
/WriteInProgress? false store
% --- Make sure that the newrow return value is within the scrolling
limits,
%     if appropriate
ScrollLimitOn? row BotScrollLimit le and {
    BotScrollLimit min
} if
end
DelayedCaretMove % --- Must be called outside of this proc's temp dict
} def

/WriteOneLine { % stringarray col row => - newcol newrow
% Put one line into the text buffer and display it on screen.
% stringarray is an array containing a single line of text.
% WriteOneLine makes use of the InputBuffer optimization.
% col,row specify the starting point of the lines. The col,row
% of the next available text position are returned.
10 dict begin
    /row exch def
    /col exch def

```

```

/s exch 0 get def

DEBUG {
  console (WriteOneLine row: % col: % slen: %\n) [row col s length]
fprintf
  console (%\n) [s] fprintf
  console flushfile
} if
/slen s length def
% --- Set the input buffer to this line if it isn't already there
row InputBufferLine ne {
  FlushInputBuffer
  % --- Copy the existing Text string into InputBuffer
  /oldline Text row Yindex get def
  oldline InputBuffer copy pop
  /InputBufferLength oldline length store
  /InputBufferLine row store
} if
insertmode? {
  % --- Make sure we aren't exceeding the size of the input buffer
  slen col add slen InputBufferLength add max InputBuffer length gt {
    % --- If we are too big, simply grow the input buffer to accomodate!
    %      Ain't automatic garbage collection wonderful...
    /InputBuffer slen col add string store
    Text row Yindex get InputBuffer copy pop
  } if
  % --- Move old Text over if necessary
  col InputBufferLength le {
    InputBuffer col slen add Xindex
    InputBuffer col Xindex InputBufferLength col sub 1 add getinterval
    putinterval
    /InputBufferLength InputBufferLength slen add store
  }{
    /InputBufferLength col slen add 1 sub store
  } ifelse
  InputBufferLength TextWidth gt {
    /TextWidth InputBufferLength store
  } if
  % --- Put insert string in
  InputBuffer col Xindex s putinterval
}{
  % --- Make sure we aren't exceeding the size of the input buffer
  slen col add InputBuffer length gt {
    % --- If we are too big, simply grow the input buffer to accomodate!
    %      Ain't automatic garbage collection wonderful...
    /InputBuffer slen col add string store
    Text row Yindex get InputBuffer copy pop
  } if
  % --- Slam the new text into the InputBuffer
  InputBuffer col Xindex s putinterval
  /InputBufferLength InputBufferLength col slen add 1 sub max store
} ifelse
% --- Update the text array with a substring of InputBuffer
Text row Yindex InputBuffer 0 InputBufferLength getinterval put
InputBufferLength TextWidth gt {
  /TextWidth InputBufferLength store
} if
% --- Paint the screen

```

```

insertmode? {
    col row InputBufferLength 1 ClearScreenArea
    col row InputBufferLength 1 DrawText
}{
    col row slen 1 ClearScreenArea
    col row slen 1 DrawText
} ifelse
% --- Remove any partial rows or columns
ViewportXdelta 0 ne ViewportYdelta 0 ne or {
    1 CanY CanHeight add TextWidth 1 ClearScreenArea
    CanX 1 sub CanY 1 TextHeight ClearScreenArea
} if
% --- Return new row,col
col slen add row
end
} def

/WriteManyLines { % stringarray row => - newcol newrow
% Put one line into the text buffer and display it on screen.
% stringarray is an array containing a single line of text.
% row specifies the starting point of the lines. The col,row
% of the next available text position are returned.
20 dict begin
    /row exch def
    /lines exch def

    DEBUG {
        console (WriteManyLines: row: % col: % numlines: %\n) [row col lines
length] fprintf
        0 1 lines length 1 sub { /i exch def
            console (%\n) [lines i get] fprintf
        } for
        console flushfile
    } if
    FlushInputBuffer
    /endrow row def
    lines {
        /s exch def
        /slen s length def
        /oldline Text endrow Yindex get def
        /oldlen oldline length def
        slen TextWidth gt {
            /TextWidth slen store
        } if
        insertmode? {
            oldlen 0 eq {
                Text endrow Yindex s put
            }{
                Text endrow Yindex s oldline append put
            } ifelse
        }{
            slen oldlen ge {
                Text endrow Yindex s put
            }{
                oldline 0 s putinterval
            } ifelse
        } ifelse
    } /endrow endrow 1 add def

```



```

        /col slen 1 add def
    } forall
    % --- Clear the changed screen area and draw the new text
    1 row TextWidth endrow row sub 1 add ClearScreenArea
    1 row TextWidth endrow row sub 1 add DrawText
    % --- Remove any partial rows or columns
    ViewportXdelta 0 ne ViewportYdelta 0 ne or {
        1 CanY CanHeight add TextWidth 1 ClearScreenArea
        CanX 1 sub CanY 1 TextHeight ClearScreenArea
    } if
    % --- Return values
    col endrow 1 sub
end
} def

/CreateCaret {    %    - => -
% --- Create, shape, and color the caret canvas
    CaretColor null eq {
        ColorDisplay?
        {/CaretColor DefaultColorCaret store}
        {/CaretColor DefaultMonoCaret store} ifelse
    } if
    gsave
        Can setcanvas
        /Caret Can newcanvas store
        Caret begin
            /Transparent false def
            Can /Retained get {
                /Retained true def
            }{
                /SaveBehind true def
            } ifelse
            /EventsConsumed /MatchedEvents def
        end
        ShapeCaret
    grestore
} def

/ShapeCaret {    %    - => -
% --- Shape the caret canvas from a proc in the CaretShapeDict
    gsave
        % ---Set up x,y arguments to caret shape proc
        PixColWidth PixRowHeight
        % ---Get the caret shape proc and execute it
        CaretShapeDict CaretShape get exec
        TM setmatrix
        0 1 translate
        Caret reshapecanvas
    grestore
} def

/MapCaret {    %    - => -
% --- Make the caret visible and color it
    gsave
        Caret mapcanvas
        Caret setcanvas
        CaretInactive? {
            DefaultInactiveColor fillcanvas

```

```

        }{
            CaretColor fillcanvas
        } ifelse
    grestore
} def

/UnMapCaret { % - => -
% --- Make the caret invisible
    Caret unmapcanvas
} def

/MoveCaret { % - => -
    gsave
        Caret unmapcanvas
        Caret setcanvas
        CaretX ViewportXdelta add CaretY ViewportYdelta add
        movecanvas
        Caret mapcanvas
    grestore
} def

/InactivateCaret { % - => -
% --- Shade the caret with the inactive color and stop any blinking
10 dict begin
    /CaretInactive? true store
    CaretOn? {
        gsave
            Caret mapcanvas
            Caret setcanvas
            DefaultInactiveColor fillcanvas
        grestore
    } if
end
} def

/ReactivateCaret { % - => -
% --- Set caret back to normal
10 dict begin
    /CaretInactive? false store
    CaretOn? {
        gsave
            Caret mapcanvas
            Caret setcanvas
            CaretColor fillcanvas
        grestore
    } if
end
} def

/SupressCaret { % - => -
% --- Temporarily turn the caret off
10 dict begin
    CaretOn? {
        /CaretSupressed? true store
        UnMapCaret
    } if
end
} def

```

```

/UnSupressCaret { % - => -
% --- Turn the caret back on
10 dict begin
  CaretOn? {
    /CaretSupressed? false store
    MapCaret
  } if
end
} def

/DelayedCaretMove { % - => -
% --- Move the caret after waiting CaretDelayTime seconds,
% but only if a write or another delayed caret move is
% not in progress
% Note: This proc must be called with the class instance dictionary
% being the first thing on the dict stack, since it forks a process
% that change instance variables.
CaretOn? {
  % --- Update the time that the caret move will actually be done
  /NextMoveTime currenttime 1 60 div CaretDelayTime mul add store
  % --- Only fork a timer if one isn't already going
  DelayedMoveProc null eq {
    % --- Fork the move timer
    /DelayedMoveProc {
      % Go to sleep, checking NextMoveTime each time we awaken
      {
        NextMoveTime currenttime sub sleep
        NextMoveTime currenttime le {
          exit
        } if
      } loop
      % --- If there is no write in progress then move the caret and
      % turn it on
      WriteInProgress? not {
        /CaretSupressed? false store
        MoveCaret
        MapCaret
      } if
      /DelayedMoveProc null store
    } fork store
  } if
} if
} def

```

```

% /DrawSelectionText { % x y y1 => -
% % --- Draw text onto the selection canvas
% 10 dict begin
% /y1 exch def
% /y exch def
% /x exch def
% gsave
% false setprintermatch
% SelectionCan setcanvas
% FgColor fillcanvas
% FontDescentTM setmatrix
% x 1 sub neg TextHeight y1 sub translate
% /SelectTM 6 array currentmatrix def

```

```

%           Font setfont
%           BgColor setcolor
%           /j 1 def
%           y 1 y1 {
%               /i exch def
%               1 i moveto
%               6 array identmatrix setmatrix
%               Text i Yindex get show
%               SelectTM setmatrix
%               /j j 1 add def
%           } for
%       grestore
%   end
% } def

/DrawSelection { % state => -
% --- Make the selection area visible or invisible, depending on state.
10 dict begin
    /state exch def
    %state {
    %     SelectionPath null ne {
    %         gsave
    %             6 array identmatrix setmatrix
    %             SelectionPath setpath
    %             SelectionCan reshapecanvas
    %             SelectionY SelectionY1 lt {
    %                 1 SelectionY SelectionY1 DrawSelectionText
    %             } if
    %             SelectionY SelectionY1 gt {
    %                 1 SelectionY1 SelectionY DrawSelectionText
    %             } if
    %             SelectionY SelectionY1 eq {
    %                 SelectionX SelectionY1 SelectionY DrawSelectionText
    %             } if
    %             SelectionCan mapcanvas
    %             /SelectionOn? true store
    %         grestore
    %     } if
    % }{ % --- state = off
    %     SelectionCan unmapcanvas
    %     /SelectionOn? false store
    % } ifelse
    % --- XXX Use xor for now; SelectionCanvas in the future...
    gsave
        SelectionPath null ne SelectionOn? state xor and {
            Can setcanvas
            TM setmatrix
            5 setrasteropcode
            SelectionPath setpath
            fill
            /SelectionOn? state store
        } if
    grestore
    end
} def

/ClearMySelection { % - => -
% --- Clear any selection I might have in the system selection mechanism

```

```

%    and on my screen.
10 dict begin
  SelectionOn? {
    false DrawSelection
    /SelectionPath null store
  } if
  /seldict /PrimarySelection getselection def
  seldict null ne {
    seldict /Canvas get Can eq {
      Selections /PrimarySelection null put
    } if
  } if
end
} def

/SendClearSelection { % - => -
% --- Clear anyone else's Primary selection
10 dict begin
  /seldict /PrimarySelection getselection def
  seldict null ne {
    seldict /Canvas get Can ne {
      /PrimarySelection clearselection
    } if
  } if
end
} def

/ExtendSelection { % - => -
% --- Draw the selection bounding outline
10 dict begin
  gsave
  FontDescentTM setmatrix
  /l Text SelectionY1 Yindex get length 2 add def
  SelectionX1 l gt {
    /SelectionX1 l def
  } if
  SelectionY SelectionY1 eq {
    SelectionX SelectionY SelectionX1 SelectionY1 1 sub
    points2rect rectpath
  }{
    SelectionY SelectionY1 gt {
      /y SelectionY1 def /x SelectionX1 def
      /y1 SelectionY def /x1 SelectionX def
    }{
      /y SelectionY def /x SelectionX def
      /y1 SelectionY1 def /x1 SelectionX1 def
    } ifelse
    1 y moveto
    x y lineto
    0 -1 rlineto
    /l Text y Yindex get length 2 add def
    1 y 1 sub lineto
    0 1 rlineto
    /y y 1 add def
    y 1 y1 1 sub {
      /i exch def
      /l Text i Yindex get length 2 add def
      1 i 1 sub lineto
    }
  }
end
}

```

```

        l i lineto
    } for
    x1 y1 1 sub lineto
    x1 y1 lineto
    l y1 lineto
    closepath
} ifelse
/SelectionPath currentpath store
stroke
grestore
end
} def

/GetSelection { % - => string
% --- Returns the text of the current selection
10 dict begin
    SelectionPath null eq {
        ()
    }{
        % --- We always want y <= y1, no matter what direction the selection
        %         was done in
        SelectionY SelectionY1 lt {
            /y SelectionY def /x SelectionX def
            /y1 SelectionY1 def /x1 SelectionX1 1 sub def
        } if
        SelectionY SelectionY1 gt {
            /y SelectionY1 def /x SelectionX1 def
            /y1 SelectionY def /x1 SelectionX 1 sub def
        } if
        SelectionY SelectionY1 eq {
            /y SelectionY def /x SelectionX def
            /y1 SelectionY1 def /x1 SelectionX1 1 sub def
            % --- If we are on the same line, we want x <= x1
            x x1 gt {
                x
                /x x1 1 add def
                /x1 exch 1 sub def
            } if
        } if
        % --- Make a string that is at least the right size
        /slen 0 def
        y 1 y1 {/i exch def /slen Text i Yindex get length slen add 1 add def}
for
        /s slen string def
        /sptr 0 def
        % --- Get the first line of the selection text
        /s1 Text y Yindex get def
        /l s1 length def
        % --- Index into it by x (add a LF if x > linelength)
        x l gt {
            s sptr LF put
            /sptr sptr 1 add def
        }{
            /s1 s1 x Xindex 1 x sub 1 add getinterval def
            s sptr s1 putinterval
            /sptr sptr s1 length add def
        } ifelse
        % --- Check for a single line selection

```

```

y y1 eq {
  % --- Clip the line at x1 (add a LF if x1 > linelength)
  x1 l gt {
    % --- Make sure we don't put in two LF's
    x1 l le {
      s sptr LF put
      /sptr sptr 1 add def
    } if
  }{
    /sptr x1 x sub 1 add def
  } ifelse
}{ % --- Multi-line selection
% --- Put LF after first line if needed
x1 l le {
  s sptr LF put
  /sptr sptr 1 add def
} if
y 1 add 1 y1 {
  /i exch def
  % --- Get the i'th line
  /l Text i Yindex get length def
  /s1 Text i Yindex get def
  % --- Check if this is the last line
  i y1 eq {
    x1 l gt {
      s sptr s1 putinterval
      /sptr sptr 1 add def
      s sptr LF put
      /sptr sptr 1 add def
    }{
      /s1 s1 0 x1 getinterval def
      s sptr s1 putinterval
      /sptr sptr s1 length add def
    } ifelse
  }{
    s sptr s1 putinterval
    /sptr sptr 1 add def
    s sptr LF put
    /sptr sptr 1 add def
  } ifelse
} for
} ifelse
s 0 sptr getinterval
} ifelse
} def

```

% ----- New Methods -----

```

/changefont { % fname fheight - => -
% --- Change the current font and point size. Either fname or fheight
% may be null, in which case they are ignored.
10 dict begin
  /fheight exch def
  /fname exch def
  fheight null ne {
    /FontHeight fheight store
    % --- Check for minimum visibility
    FontHeight 6 lt {

```

```

        /FontHeight 6 store
    } def
} if
fname null ne {
    /FontName fname store
} if
InitFont
false Reshape
1 1 TextWidth TextHeight DrawText
CaretOn? {
    ShapeCaret
    MapCaret
    MoveCaret
} if
/ResizeCallback self send
end
} def

/writelines { % arrayofstrings insertmode? col row => -
% --- Write an array of strings, starting the first string at col,row
%     with subsequent strings going at 1,row+1 1,row+2 etc. insertmode?
%     specifies whether the new lines will overwrite existing text or
%     be inserted at the specified location.
    BaseY add
    WriteLines pop pop
    pause
} def

/writeatcaret { % arrayofstrings insertmode? => -
% --- Similar to writelines, except start at the current caret location. The
%     caret is moved to the next available character position when the write is
%     done.
    CaretX CaretY WriteLines
    /CaretY exch store
    /CaretX exch store
    pause
} def

/deletestring { % length col row => -
% --- Delete a string starting at col,row for length characters.
%     length must be 0 or a positive integer.
    BaseY add
    10 dict begin
        /row exch def
        /col exch def
        /len exch def

    % --- Set the input buffer to this line if it isn't already there
    row InputBufferLine ne {
        FlushInputBuffer
        % --- Copy the existing Text string into InputBuffer
        /oldline Text row Yindex get def
        /oldlength oldline length def
        oldline InputBuffer copy pop
        /InputBufferLine row store
        /InputBufferLength oldlength store
    }{
        /oldlength InputBufferLength def

```



```

    } ifelse
    col InputBufferLength le {
        % --- Move old text over
        InputBuffer col Xindex
        InputBuffer col len add Xindex InputBufferLength len add getinterval
        putinterval
        % --- Update line length
        /InputBufferLength InputBufferLength len sub col 1 sub max store
    } if
    % --- Update the Text array
    Text row Yindex InputBuffer 0 InputBufferLength getinterval put
    % --- Update display
    col row oldlength 1 ClearScreenArea
    col row oldlength 1 DrawText
end
} def

/insertline { % numlines row => -
% --- Insert numlines blank lines, starting at line row.
%   numlines must be 0 or a positive integer
    BaseY add
    10 dict begin
        /row exch def
        /numlines exch def
        numlines row TextHeight ScrollDown
    end
} def

/deleteline { % numlines row => -
% --- Delete numlines lines, starting at line row.
%   numlines must be 0 or a positive integer
    BaseY add
    10 dict begin
        /row exch def
        /numlines exch def
        numlines row TextHeight ScrollUp
    end
} def

/setscrollinglimits { % toprow bottomrow => -
% --- Sets the scrolling limits for the TextCanvas. All up or down scrolling
%   will be limited to this region instead of affecting the entire Text
%   canvas. When scrolling limits are set, those methods which can
%   trigger scrolling (writelines, writeatcaret, movecaret delta) will
%   only cause scrolling if they affect lines within the scrolling region.
%   Any scrolling that is initiated will only move lines within the
%   scrolling region.
    /BotScrollLimit exch BaseY add def
    /TopScrollLimit exch BaseY add def
    /ScrollRegionLength BotScrollLimit TopScrollLimit sub 1 add def
    /ScrollLimitOn? true def
} def

/removescrollinglimits { % - => -
% --- Removes any scrolling bounds set by setscrollinglimits.
    /BotScrollLimit TextHeight def
    /TopScrollLimit CanY def
    /ScrollRegionLength BotScrollLimit TopScrollLimit sub 1 add def

```

```

    /ScrollLimitOn? false def
} def

/clearviewport { % - => -
% --- Clear the text and screen area of the base viewport
    ScrollLimitOn? {
        CanHeight BaseY 1 add BaseY CanHeight add ScrollUp
    }{
        CanHeight RollAllTextUp
        1 BaseY 1 add TextWidth CanHeight ClearScreenArea
    } ifelse
} def

/flashviewport { % - => -
% --- Flash the contents of the viewport (visible bell)
    gsave
        Can setcanvas
        initclip
        clipcanvaspath
        5 setrasteropcode
        fill
        clipcanvaspath
        fill
    grestore
} def

/moveviewport { %x y => -
% --- Move the viewport to another part of the underlying Text array.
%     x and y must be between 0 and 1. This represents a percentage of the
%     current total width or height of the Text array. Either argument
%     can be null, in which case it is ignored.
    10 dict begin
        SupressCaret
        dup null ne {
            /newY exch TextHeight CanHeight sub mul 1 add round def
            newY 1 lt {
                /newY 1 def
            } if
            /ydelta CanY newY sub def
            ClearMySelection
        }{
            /newY exch def
            /ydelta 0 def
        } ifelse
        dup null ne {
            /newX exch TextWidth CanWidth sub mul 1 add round def
            newX 1 lt {
                /newX 1 def
            } if
            /xdelta CanX newX sub def
            ClearMySelection
        }{
            /newX exch def
            /xdelta 0 def
        } ifelse
    gsave
        Can setcanvas
        FontDescentTM setmatrix

```

```

CanX CanY 1 sub CanWidth CanHeight 1 add rectpath
xdelta ydelta copyarea
xdelta ydelta translate
/FontDescentTM 6 array currentmatrix store
% --- XXX translate doesn't work with a matrix operand yet
TM setmatrix
xdelta ydelta translate
/TM 6 array currentmatrix store
grestore
xdelta 0 ne {
  /CanX newX store
  xdelta 0 gt {
    CanX CanY xdelta CanHeight ClearScreenArea
    CanX CanY xdelta CanHeight DrawText
  }{
    CanX CanWidth add xdelta add CanY
    xdelta neg CanHeight ClearScreenArea
    CanX CanWidth add xdelta add CanY
    xdelta neg CanHeight DrawText
    % --- Clear out column 0
    CanX 1 sub CanY 1 TextHeight ClearScreenArea
  } ifelse
} if
ydelta 0 ne {
  /CanY newY store
  ydelta 0 gt {
    CanX CanY CanWidth ydelta 1 add ClearScreenArea
    CanX CanY CanWidth ydelta 1 add DrawText
  }{
    CanX CanY CanHeight add ydelta add
    CanWidth ydelta neg ClearScreenArea
    CanX CanY CanHeight add ydelta add
    CanWidth ydelta neg DrawText
  } ifelse
} if
CanX CanY CanHeight add CanWidth 1 ClearScreenArea
/ViewportXdelta ViewportXdelta xdelta add store
/ViewportYdelta ViewportYdelta ydelta add store
MoveCaret
UnSupressCaret
end
} def

/getviewportsize { % - => col rows xpixels ypixels
% --- Return the number of columns, rows, pixel height and width of
% the viewport.
CanWidth CanHeight CanPixWidth CanPixHeight
} def

/getlinelength { % row => length
% --- Return the current length of the line at row.
BaseY add
Text exch Yindex get
length
} def

/calcareasize { % pixwidth pixheight => numcols numRows
% --- Returns the number of rows and columns that will fit into the pixel

```

```

%      area specified by pixwidth and pixheight, given the current Font and
%      point size.
10 dict begin
    /pixheight exch def
    /pixwidth exch def
    % --- Compute numcols
    pixwidth PixColWidth idiv
    % --- Compute numRows
    pixheight PixRowHeight idiv 1 add
end
} def

/calcpixarea { % numcols numRows => pixwidth pixheight
% --- Returns the minimum pixel area required to display numRows and numcols,
%      given the current Font and point size.
10 dict begin
    /numcols exch def
    /numrows exch def
    % --- Compute pixwidth
    numcols PixColWidth mul
    % --- Compute pixheight
    numRows PixRowHeight mul 1 sub
end
} def

/oncaret { % - => -
% --- Turn the caret on.
    /CaretOn? true def
    MapCaret
    MoveCaret
} def

/offcaret { % - => -
% --- Turn the caret off.
    UnMapCaret
    /CaretOn? false def
} def

/movecaret { % col row => -
% --- Move the caret to an absolute position. col and row are integers. Scrolling
is
%      never triggered.
    /CaretY exch BaseY add def
    /CaretX exch def
    CaretX TextWidth gt {/TextWidth CaretX store} if
    MoveCaret
} def

/movecaretdelta { % deltax deltay => -
% --- Move the caret relative to its current position. deltax and deltay must be
%      integers (negatives allowed). Scrolling is triggered if deltay moves the
caret
%      outside of the scrolling limits. If no scrolling limits are set, scrolling
is
%      triggered if deltay moves the caret outside of the original viewport
region.
10 dict begin
    /deltay exch def

```

```

/deltax exch def
/CaretY CaretY deltay add store
/CaretX CaretX deltax add store
CaretY TopScrollLimit lt {
    TopScrollLimit CaretY sub TopScrollLimit BotScrollLimit ScrollDown
    /CaretY TopScrollLimit store
} if
CaretY BotScrollLimit gt {
    CaretY BotScrollLimit sub TopScrollLimit BotScrollLimit ScrollUp
    /CaretY TextHeight store
} if
CaretX 1 lt {/CaretX 1 store} if
CaretX TextWidth gt {/TextWidth CaretX store} if
MoveCaret
end
} def

/setcaretblink { % blink-rate duty-cycle => -
% --- Set the caret blink rate and the blink duty cycle. blink-rate is
% in seconds and represents a complete on/off cycle. duty-cycle is
% between 0 and 1, and represents the percentage of on time.
dup null ne {
    /CaretDutyCycle exch def
}{
    pop
} ifelse
dup null ne {
    dup 0 ne {
        /CaretBlinkTime exch def
        /CaretBlinkEnabled? true def
    }{
        % --- Disable caret, but keep blink events going at a
        % 2 second rate
        pop
        /CaretBlinkTime 2 def
        /CaretBlinkEnabled? false def
        CaretOn? {MapCaret} if
    } ifelse
}{
    pop
} ifelse
} def

/setcaretcolor { % color => -
% --- Set the current caret color. color is a color object.
    /CaretColor exch def
    CaretOn? {MapCaret} if
} def

/setcaretshape { % shapename => successful?
% --- Set the caret shape. shapename should be an entry in the CaretShapeDict.
% Return a boolean that tells whether shapename was found.
dup CaretShapeDict exch known {
    /CaretShape exch def
    Caret null ne {ShapeCaret MoveCaret} if
    true
} {
    pop pop

```

```

        false
    } ifelse
} def

/getcaretpos { % - => col row
% --- Return the current caret position
    CaretX
    CaretY BaseY sub
} def

/setbgcolor { % color => -
% --- Set the current canvas background color
    ClearMySelection
    /BgColor exch def
} def

/setfgcolor { % color => -
% --- Set the current text color
    ClearMySelection
    /FgColor exch def
} def

/fixdamage { % - => -
% --- Damage handler; goes in the PaintClient window callback routine
    BgColor fillcanvas
    CanX CanY CanWidth CanHeight DrawText
} def

/new { % numrows can => object
% --- Create a new instance of the TextCanvas. numrows is the number
%       of lines to be allocated in the Text array; it is fixed for
%       the life of the instance. can is the viewport canvas.
    /new super send begin
        /Can exch def
        /TextHeight exch def
        Can /Retained true put % XXX - Non-retained will work, but NeWS sometimes
                                % reports more damage than has actually occurred,
                                % so going non-retained can be very costly

        gsave
            % --- Determine canvas pixel width and height
            Can setcanvas
            initclip clipcanvaspath pathbbox % llx lly urx ury
            points2rect % x y w h
        grestore
        /CanPixHeight exch def
        /CanPixWidth exch def
        /CanPixY exch def
        /CanPixX exch def
        InitFont
        true Reshape
        currentdict
    end
} def

/reshape { % - => -
% --- This method must be called whenever the viewport canvas has changed
%       size. It updates the number of rows and columns in the TextCanvas,
%       repositions the caret to be as close to its old position as possible,

```

```

%      resets the scrolling region, and moves the viewport to its base position.
gsave
% --- Determine the new pixel width and height
Can setcanvas
initclip clipcanvaspath pathbbox % llx lly urx ury
points2rect % x y w h
grestore
/CanPixHeight exch def
/CanPixWidth exch def
/CanPixY exch def
/CanPixX exch def
false Reshape
/ResizeCallback self send
} def

/destroy { % - => -
    KeyboardInterest Can revokekbdinterests
    KeyboardEventManager null ne { % added! -deh
        KeyboardEventManager killprocess
    } if
    EventMgr null ne {
        EventMgr killprocess
    } if
    DelayedMoveProc null ne { % added! -deh
        DelayedMoveProc killprocess
    } if
    MouseDragEventManager null ne {
        MouseDragEventManager killprocess
    } if
} def

classend def
end} if % systemdict /TextCanvas known not...

```