XVT Portability Toolkit

Reference

xvt_dwin_*

Drawable Window Functions

xvt_dwin_clear
xvt_dwin_close_pict
xvt_dwin_draw_aline
xvt_dwin_draw_arc
xvt_dwin_draw_icon
xvt_dwin_draw_image
xvt_dwin_draw_line
xvt_dwin_draw_oval
xvt_dwin_draw_pict
xvt_dwin_draw_pie
xvt_dwin_draw_pmap
xvt_dwin_draw_polygon
xvt_dwin_draw_polyline
xvt_dwin_draw_rect
xvt_dwin_draw_roundrect
xvt_dwin_draw_set_pos
xvt_dwin_draw_text
xvt_dwin_get_clip
xvt_dwin_get_draw_ctools
xvt_dwin_get_font
xvt_dwin_get_font_app_data
xvt_dwin_get_font_family
xvt_dwin_get_font_family_mapped
xvt_dwin_get_font_metrics
xvt_dwin_get_font_native_desc
xvt_dwin_get_font_size
xvt_dwin_get_font_size_mapped
xvt_dwin_get_font_style
xvt_dwin_get_font_style_mapped
xvt_dwin_get_text_width
xvt_dwin_invalidate_rect
xvt_dwin_is_update_needed
xvt_dwin_open_pict
xvt_dwin_scroll_rect
xvt_dwin_set_back_color
xvt_dwin_set_cbrush
xvt_dwin_set_clip
xvt_dwin_set_cpen
xvt_dwin_set_draw_ctools
xvt_dwin_set_draw_mode
xvt_dwin_set_font
xvt_dwin_set_font_app_data
xvt_dwin_set_font_family
xvt_dwin_set_font_native_desc
xvt_dwin_set_font_size
xvt_dwin_set_font_style
xvt_dwin_set_fore_color
xvt_dwin_set_std_cbrush

```
xvt_dwin_set_std_cpen
xvt_dwin_update
```

xvt_dwin_clear

Clear a Window's Client Area

Summary

```
void xvt_dwin_clear(WINDOW win, COLOR color)
```

`WINDOW win`

Window whose client area is being filled with color.

`COLOR color`

Specified color.

Description

This function clears a window's client area by filling it with a specified color. If a clipping rectangle has been set for the target window, the filled area will be limited to that of the rectangle set by `xvt_dwin_set_clip`.

To clear a window with the window system's default background color set by the user, use `xvt_dwin_clear` like this:

```
xvt_dwin_clear(win, (COLOR)(xvt_vobj_get_attr(
    NULL_WIN, ATTR_BACK_COLOR)));
```

If the color must match the `DRAW_CTOOLS` background color for that window, use `xvt_dwin_clear` like this:

```
DRAW_CTOOLS ct;xvt_dwin_clear(
    win, (xvt_dwin_get_draw_ctools (win,
    &ct))->back_color);
```

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`

Implementation Note

On XVT/Win32/64 with some device drivers, clearing a print window can result in excessively large print data being sent to the printer.

Normally, TASK_WIN is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute ATTR_WIN_PM_DRAWABLE_TWIN before the call to xvt_app_create. In that case, TASK_WIN would be a valid window for this function.

See Also

ATTR_BACK_COLOR
COLOR
DRAW_CTOOLS
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
XVT_PIXMAP
xvt_dwin_get_draw_ctools
xvt_dwin_set_clip
xvt_vobj_get_attr

The "Drawing and Pictures" chapter in the *XVT Portability Toolkit Guide*

xvt_dwin_close_pict

Finish an Encapsulated Picture

Summary

```
PICTURE xvt_dwin_close_pict(WINDOW win)
```

WINDOW win

Window in which to finish an encapsulated picture.

Description

This function completes the encapsulation of a PICTURE that was begun with a call to xvt_dwin_open_pict, and returns the PICTURE object. To free the memory occupied by a PICTURE, call xvt_pict_destroy.

Return Value

A PICTURE if successful; NULL on an error (such as inadequate memory).

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a control window, print windows, `XVT_PIXMAP`, dialog, or control

See Also

```
CBRUSH
PICTURE
WINDOW
XVT_PIXMAP
xvt_dwin_open_pict
xvt_pict_destroy
```

The "Drawing and Pictures" chapter in the *XVT Portability Toolkit Guide*

Example

```
WINDOW win;
(PICTURE) NULL;
PICTURE pict;
RCT frame;
CBRUSH cbrsh;cbrsh.pat = PAT_DIAGCROSS;
cbrsh.color = COLOR_BLACK;xvt_rect_set(&frame, 20, 20,
    100, 100);
if (xvt_dwin_open_pict(win, &frame)) {
    xvt_dwin_set_cbrush(win, &cbrsh);
    xvt_dwin_set_std_cpen(win, TL_PEN_BLACK);
    xvt_dwin_draw_oval(win, &frame);
    pict = xvt_dwin_close_pict(win);
}
```

xvt_dwin_draw_aline

Draw Line with Arrows at Ends

Summary

```
void xvt_dwin_draw_aline(WINDOW win, PNT pnt,
    BOOLEAN start_arrow, BOOLEAN end_arrow)

WINDOW win
```

Window in which to draw the line.

`PNT pnt`

Ending point of line.

`BOOLEAN start_arrow`

If `TRUE`, the starting point has an arrow head.

`BOOLEAN end_arrow`

If `TRUE`, the ending point has an arrow head.

Description

This function draws a line, from the current pen position to the point indicated by `pnt`, in the client area of `win`. If `start_arrow` is `TRUE`, the starting point has an arrowhead. If `end_arrow` is `TRUE`, the ending point has an arrowhead. If both are `FALSE`, this function behaves identically to `xvt_dwin_draw_line`.

The current pen position is set with `xvt_dwin_draw_set_pos` or with a previous call to `xvt_dwin_draw_aline` or `xvt_dwin_draw_line`. Intervening calls to other drawing functions leave the pen position in an undefined location, so make sure to set the pen position explicitly in that case.

The current `CPEN` and `DRAW_MODE` are used to draw the line and the arrowheads. The current `CBRUSH` is not used.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on XVT/Win32/64, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case `TASK_WIN` would be a valid window for this function.

See Also

CBRUSH
CPEN
DRAW_MODE
PNT
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_app_create
xvt_dwin_draw_line
xvt_dwin_draw_set_pos

Example

The following code draws a line from (10, 15) to (100, 75), with an arrowhead at (10, 15):

```
PNT pnt;xvt_dwin_set_std_cpen(win, TL_PEN_BLACK);
pnt.h = 10;
pnt.v = 15;
xvt_dwin_draw_set_pos(win, pnt);
pnt.h = 100;
pnt.v = 75;
xvt_dwin_draw_aline(win, pnt, TRUE, FALSE);
```

xvt_dwin_draw_arc

Draw the Arc of an Oval

Summary

```
void xvt_dwin_draw_arc(WINDOW win, RCT *rctp,
     int start_x, int start_y, int stop_x, int stop_y)
```

`WINDOW win`

Window in which to draw the arc of an oval.

`RCT *rctp`

Bounding rectangle of the oval. If the `RCT*` parameter to this function is an empty rectangle, nothing will be drawn.

`int start_x`

Arc's starting x-coordinate.

`int start_y`

Arc's starting y-coordinate.

`int stop_x`

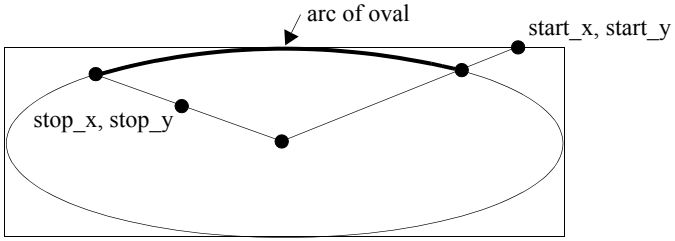
Arc's stopping x-coordinate.

`int stop_y`

Arc's stopping y-coordinate.

Description

This function draws an arc that is a section of the perimeter of an oval bounded by `rctp` in the client area of `win`. The arc is drawn counter-clockwise along the oval, from the point (`start_x`, `start_y`) to the point (`stop_x`, `stop_y`). If one (or both) of these points is not exactly on the oval, an imaginary line is drawn from the center of the bounding rectangle to the point, and the intersection of that line and the oval is used as the starting or stopping point.



This function expects the rectangle to be normalized. That is, the rectangle must be valid, with `rctp->top` less than `rctp->bottom` and `rctp->left` less than `rctp->right`.

The current `CPEN` and `DRAW_MODE` are used to draw the arc. The current `CBRUSH` is not used.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

CBRUSH
CPEN
DRAW_MODE
RCT
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_app_create
xvt_dwin_draw_oval
xvt_dwin_draw_pie
xvt_dwin_set_std_cpen
xvt_rect_set

Example

This code draws an arc that covers the northeast quadrant of an oval:

```
RCT rct;
xvt_dwin_set_std_cpen(win, TL_PEN_BLACK);
xvt_rect_set(&rct, 10, 20, 60, 80);
xvt_dwin_draw_arc(win, &rct, 60, 50, 35, 20);
```

xvt_dwin_draw_icon

Draw an Icon

Summary

```
void xvt_dwin_draw_icon(WINDOW win, int x, int y,
     int rid)
```

`WINDOW win`

Window in which to draw the icon.

`int x`

Icon’s x-coordinate.

`int y`

Icon’s y-coordinate.

`int rid`

Icon’s resource ID.

Description

This function draws the icon whose resource ID is `rid` so that its upper-left corner is at point `x`, `y` in the client area of `win`. The current

background and foreground colors are used. The current DRAW_MODE and the current CPEN or CBRUSH are not used.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- win must be a valid XVT WINDOW of type W_* (except W_SCREEN and W_TASK)
- win is not a dialog or control
- Print windows and XVT_PIXMAPs are valid values for win

Implementation Note

- For XVT/Mac, the icon must have a resource type of ICON or CICN
- For XVT/Win32/64, the icon must be declared in an ICON statement in the resource script
- For XVT/XM, there must be an ICON definition in your resource manager file
- Normally, TASK_WIN is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute ATTR_WIN_PM_DRAWABLE_TWIN before the call to xvt_app_create. In that case, TASK_WIN would be a valid window for this function.

See Also

```
CBRUSH
CPEN
DRAW_MODE
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_app_create
xvt_dwin_draw_image
xvt_dwin_draw_pmap
```

The "Icons" section of the "Controls" chapter in the *XVT Portability Toolkit Guide*
The *XVT Platform-Specific Books*

Example

```
xvt_dwin_clear(win, COLOR_WHITE);
xvt_dwin_draw_text(win, 4, 30, "Hello World!", -1);
xvt_dwin_draw_icon(win, 10, 40, ICON_RID);
```

xvt_dwin_draw_image

Draw an Image in a Window or Pixmap

Summary

```
void xvt_dwin_draw_image(WINDOW dstwin,
     XVT_IMAGE srcimage, RCT *dstrctp, RCT *srcrctp)
```

`WINDOW dstwin`

Window in which to draw the image. It can be a drawable window, a print window, or an `XVT_PIXMAP`.

`XVT_IMAGE srcimage`

Image to be drawn.

`RCT *dstrctp`

Pointer to a rectangle that defines, in the coordinates of `dstwin`, the location and size of the drawn image. If the "destination" rectangle (`dstrctp`) is empty, nothing will be drawn.

`RCT *srcrctp`

Pointer to a rectangle that defines, in the coordinates of `srcimage`, the location and size of the portion of the image to be copied to `dstwin`.

Description

This function draws an image in a window (including print windows) or pixmap. The color of each pixel is mapped to the nearest available color in the palette of the destination window or pixmap.

If `*srcrctp` and `*dstrctp` are not congruent, this function translates and scales the image as necessary to fit it into the destination rectangle. Any parts of the source or destination rectangles that fall outside of the bounds of their respective containers are ignored (clipped).

`xvt_dwin_draw_image` ignores the drawing mode of `dstwin`, and always uses `M_COPY`.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- `srcimage` is `NULL` or invalid
- `dstwin` is not a valid drawable window or pixmap
- `dstwin` is a dialog or control
- `dstrctp` or `srcrctp` is not a valid pointer to a rectangle
- `srsrstp` is an empty rectangle

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
M_* Values for DRAW_MODE
RCT
WINDOW
XVT_IMAGE
XVT_PIXMAP
xvt_dwin_draw_icon
xvt_dwin_draw_pmap
xvt_image_transfer
```

The "Color Mapping" and "Transfer Operations" sections of the "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

```
/* draw image at double size into window */
RCT src_rect, dst_rect;
short height, width;
xvt_image_get_dimensions(image, &width, &height);
xvt_rect_set(&src_rect, 0, 0, width, height);
xvt_rect_set(&dst_rect, 0, 0, (width-1)*2 + 1,
     (height-1)*2 + 1);
 xvt_dwin_draw_image(window, image, &dst_rect,
     &src_rect);
```

xvt_dwin_draw_line

Draw Line from Current Position to Point

Summary

```
void xvt_dwin_draw_line(WINDOW win, PNT pnt)
```

`WINDOW win`

Window in which to draw the line.

`PNT pnt`

Ending point of the line.

Description

This function draws a line (without arrowheads) from the current pen position to the point indicated by `pnt`, in the client area of `win`.

The current pen position is set with a previous call to `xvt_dwin_draw_set_pos`, `xvt_dwin_draw_aline`, or `xvt_dwin_draw_line`. Intervening calls to other drawing functions leave the pen position in an undefined location, so make sure to set the pen position explicitly if this is the case.

The current `CPEN` and `DRAW_MODE` are used to draw the line. The current `CBRUSH` is not used.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

CBRUSH
CPEN
DRAW_MODE
PNT
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_app_create
xvt_dwin_draw_aline
xvt_dwin_draw_line
xvt_dwin_draw_set_pos

Example

This code draws a line from (10, 15) to (100, 75).

```
PNT pnt;pnt.h = 10;
pnt.v = 15;
xvt_dwin_draw_set_pos(win, pnt);
pnt.h = 100;
pnt.v = 75;
xvt_dwin_draw_line(win, pnt);
```

xvt_dwin_draw_oval

Draw Oval

Summary

```
void xvt_dwin_draw_oval(WINDOW win, RCT *rctp)
```

`WINDOW win`

Window in which to draw the oval.

`RCT *rctp`

Bounding rectangle of an oval (ellipse). If the `RCT*` parameter to this function is an empty rectangle, nothing will be drawn.

Description

This function draws an oval (ellipse) that is bounded by the rectangle pointed to by `rctp`. The rectangle must be valid, with `rctp->top` less than `rctp->bottom` and `rctp->left` less than `rctp->right`.

The current `CPEN`, `CBRUSH`, and `DRAW_MODE` are used to draw the oval.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`
- `rctp` must be a valid non-empty rectangle

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
CBRUSH
COLOR_*, COLOR_INVALID Constants
PAT_* Values for PAT_STYLE
RCT
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_app_create
xvt_dwin_draw_arc
xvt_dwin_draw_oval
xvt_dwin_draw_rect
xvt_dwin_set_cbrush
xvt_dwin_set_std_cpen
xvt_rect_set
```

Example

The following code draws a circle surrounded by an ellipse. Note that the "circle" will be circular only if the horizontal and vertical resolutions are equal.

```
RCT rct;
CBRUSH brush;xvt_dwin_set_std_cpen(
    win,TL_BLACK_PEN);
brush.pat = PAT_SOLID;
brush.color = COLOR_LTGRAY;
xvt_dwin_set_cbrush(win, &brush); /* ellipse */
xvt_rect_set(&rct, 50, 75, 200, 125);
xvt_dwin_draw_oval(win, &rct);
brush.color = COLOR_DKGRAY;
xvt_dwin_set_cbrush(win, &brush);
xvt_rect_set(&rct, 100, 75, 150, 125); /* circle */
xvt_dwin_draw_oval(win, &rct);
```

xvt_dwin_draw_pict

Draw Encapsulated Picture

Summary

```
void xvt_dwin_draw_pict(WINDOW win, PICTURE pic,
     RCT *rctp)
```

`WINDOW win`

Window in which to draw an encapsulated picture.

`PICTURE pic`

Picture.

`RCT *rctp`

`win` must be a valid XVT `WINDOW` of `type W_*` except `W_SCREEN` and `W_TASK`.. Destination rectangle. If the `RCT*` parameter to this function is an empty rectangle, nothing will be drawn.

Description

This function draws a `PICTURE` in the client area of `win`.

Note: The current drawing tools have no effect on the rendering of the `PICTURE`. It is recommended that the `PICTURE` drawing rectangle be cleared either in the `PICTURE` or in the window prior to calling `xvt_dwin_draw_pict`.

The `PICTURE` will be expanded, reduced or stretched to fit in the rectangle specified by `rctp`. For best results, the destination frame should be the same shape as the frame in which the `PICTURE` was originally drawn.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`
- `rctp` must be a valid non-empty rectangle

Implementation Note

On XVT/Mac, `PICTURE`s are Mac PICTs, which scale and stretch nicely. On XVT/Win32/64 and XVT/XM, a `PICTURE` is a bitmap, which tends to look "jaggy" when scaled or stretched. To avoid any of these artifacts, you should draw the `PICTURE` in its original size.

On systems where `PICTURE`s are bitmaps, drawing the `PICTURE` in its original size is significantly faster than scaling or stretching it.

Normally, `TASK_WIN` is not a valid window for this function. However, on XVT/Win32/64, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
PICTURE
RCT
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_app_create
xvt_dwin_open_pict
xvt_pict_create
```

The "Drawing and Pictures"chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_pict_create`.

xvt_dwin_draw_pie

Draw a Pie Section of an Oval

Summary

```
void xvt_dwin_draw_pie(WINDOW win, RCT *rctp,
     int start_x, int start_y, int stop_x, int stop_y)
```

`WINDOW win`

Window in which to draw the pie section of an oval.

`RCT *rctp`

`win` must be a valid XVT `WINDOW` of type `W_*` except `W_SCREEN` and `W_TASK`. . Bounding rectangle of oval. If the `RCT*` parameter to this function is an empty rectangle, nothing will be drawn.

`int start_x`

Starting x-coordinate.

`int start_y`

Starting y-coordinate.

`int stop_x`

Stopping x-coordinate.

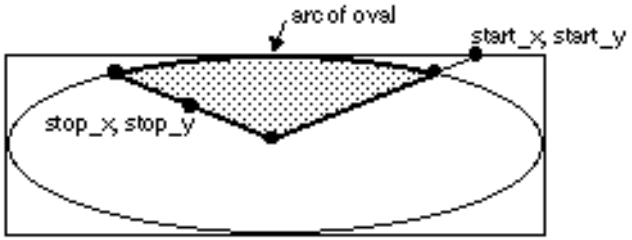
`int stop_y`

Stopping y-coordinate.

Description

This function draws a section of an oval (a pie slice) in the client area of `win`. The oval is bounded by the rectangle pointed to by `rctp`. Just as with `xvt_dwin_draw_arc`, an arc is drawn counter-clockwise along the oval, from the point (`start_x`, `start_y`) to the point (`stop_x`, `stop_y`).

If one (or both) of these points is not exactly on the oval, an imaginary line is drawn from the center of the bounding rectangle to the point, and the intersection of that line and the oval is taken as the starting or stopping point. Then lines are drawn from the center of the rectangle to each end of the arc to construct an enclosed shape. `xvt_dwin_draw_pie` requires a normalized rectangle; that is, `rctp->top` less than `rctp->bottom` and `rctp->left` less than `rctp->right`.



The current CPEN, CBRUSH, and DRAW_MODE are used to draw the pie section.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- win must be a valid XVT WINDOW of type W_* (except W_SCREEN and W_TASK)
- win is not a dialog or control
- Print windows and XVT_PIXMAPs are valid values for win
- rctp must be a valid non-empty rectangle

Implementation Note

Normally, TASK_WIN is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute ATTR_WIN_PM_DRAWABLE_TWIN before the call to xvt_app_create. In that case, TASK_WIN would be a valid window for this function.

See Also

CBRUSH
COLOR_*, COLOR_INVALID Constants
PAT_* Values for PAT_STYLE
RCT
TASK_WIN
TL_* Constants
WINDOW
XVT_PIXMAP
xvt_app_create
xvt_dwin_draw_arc
xvt_dwin_set_cbrush
xvt_dwin_set_std_cpen
xvt_rect_set
xvt_vobj_get_client_rect

Example

The following code draws a pie section that covers an eighth of an oval, going from a point due south to a point southwest:

```
RCT rct;
CBRUSH brush;brush.pat = PAT_BDIAG;
brush.color = COLOR_WHITE;
xvt_dwin_set_cbrush(win, &brush);
xvt_dwin_set_std_cpen(win, TL_PEN_BLACK);
xvt_rect_set(&rct, 100, 40, 200, 90);
xvt_dwin_draw_pie(win, &rct, 100, 90, 150, 90);
```

Assuming that the piece of pie just drawn was the first piece cut, this code draws the remaining part of the pie, by reversing the starting and stopping points:

```
RCT rct;
CBRUSH brush;brush.pat = PAT_BDIAG;
brush.color = COLOR_WHITE;
xvt_dwin_set_cbrush(win, &brush);
xvt_dwin_set_std_cpen(win, TL_PEN_BLACK);
xvt_rect_set(&rct, 100, 40, 200, 90);
xvt_dwin_draw_pie(win, &rct, 150, 90, 100, 90);
```

xvt_dwin_draw_pmap

Draw a Pixmap in a Window or Pixmap

Summary

```
void xvt_dwin_draw_pmap(WINDOW dstwin,
     XVT_PIXMAP srcpmap, RCT *dstrctp, RCT *srcrctp)
```

`WINDOW dstwin`

Window in which to draw the image. It can be a drawable window, a print window, or an `XVT_PIXMAP`.

`XVT_PIXMAP srcpmap`

Pixmap to be drawn.

`RCT *dstrctp`

Pointer to a rectangle that defines, in the coordinates of `dstwin`, the location and size of the drawn pixmap. If the "destination" rectangle (`dstrctp`) is empty, nothing will be drawn.

`RCT *srcrctp`

Pointer to a rectangle that defines, in the coordinates of `srcpmap`, the location and size of the portion of the pixmap to be copied to `dstwin`.

Description

This function draws a pixmap in a window or pixmap. No color mapping is performed; the colors of the pixmap are unpredictably altered if `srcpmap` and `dstwin` do not have identical color palettes.

If `*srcrctp` and `*dstrctp` are not congruent, this function translates and scales the image as necessary to fit it into the destination rectangle. Any parts of the source or destination rectangles that fall outside of the bounds of their respective containers are ignored.

`xvt_dwin_draw_pmap` uses the drawing mode of `dstwin` when drawing the pixmap.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- `dstwin` is a dialog or control
- `dstwin` is not a valid window or pixmap
- `srcpmap` is `NULL` or invalid
- `dstrctp` or `srcrctp` are `NULL`, empty, or invalid rectangles

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
RCT
WINDOW
XVT_PIXMAP
xvt_dwin_draw_icon
xvt_dwin_draw_image
xvt_dwin_set_draw_mode
xvt_rect_set
```

The "Palettes" and "Transfer Operations" sections of the "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

```
/* draw Pixmap at double size into window */
RCT src_rect, dst_rect;
xvt_dwin_get_client_rect(pixmap, &src_rect);
xvt_rect_set(&dst_rect, 0, 0, (src_rect.right-1)*2 + 1,
     (src_rect.bottom-1)*2 + 1);
 xvt_dwin_draw_pmap(window, pixmap, &dst_rect,
     &src_rect);
```

xvt_dwin_draw_polygon

Draw a Polygon

Summary

```
void xvt_dwin_draw_polygon(WINDOW win, PNT *lpnts,
     int npnts)
```

`WINDOW win`

Window in which to draw the polygon.

`PNT *lpnts`

Array of points.

`int npnts`

Number of points.

Description

This function draws a polygon described by `npnts` vertices in the array `lpnts`. The polygon is drawn into the client area of `win`. If the starting and ending points don't coincide, an additional side is drawn to close the shape by connecting the starting and ending points, so that there is an enclosed interior. The points are connected in order found in the array. If any sides intersect, the determination of what's inside and what's outside is platform-specific.

The current `CPEN`, `CBRUSH`, and `DRAW_MODE` are used to draw the polygon.

Note: For efficiency, set the first point equal to the last point. Otherwise, XVT might have to allocate a temporary array with `npnts + 1` points in it.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`
- The list of points must not be `NULL`

Implementation Note

For portability, the polygon's sides should not intersect (on some platforms, intersecting sides result in undefined behavior).

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to xvt_app_create. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
CBRUSH
CPEN
DRAW_MODE
PICTURE
PNT
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_app_create
xvt_dwin_draw_line
xvt_dwin_draw_polyline
xvt_dwin_draw_rect
xvt_dwin_set_std_cbrush
xvt_dwin_set_std_cpen
```

Example

This code draws a triangle. Note that this uses only three points, since the shape is automatically closed. Compare this example to the one under the topic `xvt_dwin_draw_polyline`.

```
static PNT p[] = { {100, 50}, {200, 75}, {150, 125}
    };xvt_dwin_set_std_cpen(win, TL_PEN_BLACK);
xvt_dwin_set_cbrush(win, &white_cbrush);
xvt_dwin_draw_polygon(win, p, 3);
```

As mentioned above, on some platforms it may be more efficient to specify four points, like this:

```
static PNT p[] = { {100, 50}, {200, 75}, {150, 125},
     {100, 50} };
```

With this call:

```
xvt_dwin_draw_polygon(win, p, 4);
```

In addition, closing the polygon (i.e., setting the first point equal to the last point) is consistent with using `xvt_dwin_draw_polyline` to draw a polygon with a border.

xvt_dwin_draw_polyline

Draw a Polyline

Summary

```
void xvt_dwin_draw_polyline(WINDOW win, PNT *lpnts,
     int npnts)
```

`WINDOW win`

Window in which to draw the `polyline`.

`PNT *lpnts`

Array of points.

`int npnts`

Number of points.

Description

This function connects the npnts points in the `lpnts` array with straight lines, and draws the lines in the client area of `win`. The last point is *not* automatically connected to the first; if you want a closed shape, make them the same. However, even if you do, the shape is not considered to have an interior. If you want an interior, use `xvt_dwin_draw_polygon`.

The current `CPEN` and `DRAW_MODE` are used to draw the polyline. The current `CBRUSH` is not used.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)

- win is not a dialog or control
- Print windows and XVT_PIXMAPs are valid values for win
- lpnts must be a valid list of points

Implementation Note

Normally, TASK_WIN is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute ATTR_WIN_PM_DRAWABLE_TWIN before the call to xvt_app_create. In that case, TASK_WIN would be a valid window for this function.

See Also

CBRUSH
CPEN
DRAW_MODE
PNT
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_app_create
xvt_dwin_draw_line
xvt_dwin_draw_polygon
xvt_dwin_draw_rect
xvt_dwin_set_std_cpen

Example

This code draws a triangle. Note that this takes four points, not three, since the first and last points have to be the same to create a closed shape.

```
static PNT p[] = { {100, 50}, {200, 75}, {150, 125},
     {100, 50} };
xvt_dwin_set_std_cpen(win, TL_PEN_BLACK);
xvt_dwin_draw_polyline(win, p, 4);
```

xvt_dwin_draw_rect

Draw a Rectangle

Summary

```
void xvt_dwin_draw_rect(WINDOW win, RCT *rctp)

WINDOW win
```

Window in which to draw the rectangle.

`RCT *rctp`

Rectangle. If the `RCT*` parameter to this function is an empty rectangle, nothing will be drawn.

Description

This function draws the rectangle pointed to by `rctp` in the client area of `win`. The rectangle must be normalized, with `rctp->top` less than `rctp->bottom` and `rctp->left` less than `rctp->right`.

The current `CPEN`, `CBRUSH`, and `DRAW_MODE` are used to draw the rectangle.

A special usage of `xvt_dwin_draw_rect` is to support the inverting of text to show selection. Draw a rectangle over previously drawn text in `M_XOR` mode using a `PAT_HOLLOW` pen and a `COLOR_BLACK` solid brush. Other methods might display gaps between selection rectangles that are supposed to touch. The above method doesn't produce this problem.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`
- `rctp` must be a valid non-empty rectangle

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
CBRUSH
COLOR_*, COLOR_INVALID Constants
CPEN
DRAW_MODE
M_* Values for DRAW_MODE
PAT_* Values for PAT_STYLE
RCT
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_app_create
xvt_dwin_draw_line
xvt_dwin_draw_polygon
xvt_dwin_draw_polyline
xvt_dwin_draw_roundrect
```

The "Showing Text Selections" section of the "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

This code draws a rectangle:

```
RCT rct;
CBRUSH brush;brush.pat = PAT_DIAGCROSS;
brush.color = COLOR_WHITE;
xvt_dwin_set_cbrush(win, &brush);
xvt_dwin_set_std_cpen(win, TL_PEN_BLACK);
xvt_rect_set(&rct, 100, 40, 200, 90);
xvt_dwin_draw_rect(win, &rct);
```

xvt_dwin_draw_roundrect

Draw a Rectangle with Rounded Corners

Summary

```
void xvt_dwin_draw_roundrect(WINDOW win, RCT *rctp,
     int oval_width, int oval_height)
```

`WINDOW win`

Window in which to draw the rectangle with rounded corners.

`RCT *rctp`

Bounding rectangle. If the `RCT*` parameter to this function is an empty rectangle, nothing will be drawn.

`int oval_width`

Width of corner oval.

`int oval_height`

Height of corner oval.

Description

This function draws the rounded rectangle bounded by `rctp`, in the client area of `win`. This is similar to `xvt_dwin_draw_rect`, but the corners are rounded. Each corner is a quadrant of an oval that is `oval_width` wide and `oval_height` high. The rectangle must be normalized, with `rctp->top` less than `rctp->bottom` and `rctp->left` less than `rctp->right`.

The current `CPEN`, `CBRUSH`, and `DRAW_MODE` are used to draw the rounded rectangle.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`
- `rctp` must be a valid non-empty rectangle

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

CBRUSH
COLOR_*, COLOR_INVALID Constants
CPEN
DRAW_MODE
PAT_* Values for PAT_STYLE
PICTURE
PNT
RCT
TASK_WIN
TL_* Constants
WINDOW
XVT_PIXMAP
xvt_app_create
xvt_dwin_draw_oval
xvt_dwin_draw_polygon
xvt_dwin_draw_rect

Example

This code draws a rounded rectangle.

```
RCT rct;
CBRUSH brush;brush.pat = PAT_DIAGCROSS;
brush.color = COLOR_WHITE;
xvt_dwin_set_cbrush(win, &brush);
xvt_dwin_set_std_cpen(win, TL_PEN_BLACK);
xvt_rect_set(&rct, 100, 40, 200, 90);
xvt_dwin_draw_roundrect(win, &rct, 10, 15);
```

xvt_dwin_draw_set_pos

Move Pen Position to Point

Summary

```
void xvt_dwin_draw_set_pos(WINDOW win, PNT pnt)
```

`WINDOW win`

Window whose current pen position is to be moved.

`PNT pnt`

Location of the point to which the pen is being moved.

Description

This function moves the current pen position for win to the location indicated by pnt, without drawing anything. The current pen

position is used in conjunction with the xvt_dwin_draw_aline and xvt_dwin_draw_line functions.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- win must be a valid XVT WINDOW of type W_* (except W_SCREEN and W_TASK)
- win is not a dialog or control
- Print windows and XVT_PIXMAPs are valid values for win

Implementation Note

Normally, TASK_WIN is not a valid value for win. However, on XVT/Win32, you can set the non-portable attribute ATTR_WIN_PM_DRAWABLE_TWIN before the call to xvt_app_create. In that case, TASK_WIN would be a valid window for this function.

See Also

PNT
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_app_create
xvt_dwin_draw_aline
xvt_dwin_draw_line
xvt_dwin_draw_polyline

Example

See the example for xvt_dwin_draw_line.

xvt_dwin_draw_text

Draw Text String

Summary

```
void xvt_dwin_draw_text(WINDOW win, int x, int y,
    char *s, int len)
```

WINDOW win

Window in which to draw the text string.

int x

Starting x-coordinate; the left side of the first character.

`int y`

Starting y-coordinate; the text's baseline.

`char *s`

Single-byte or multbyte text string.

`int len`

Number of characters in the string (if -1, the entire string is output).

Description

This function outputs the text string `s` starting at the point (`x`, `y`), into the client area of `win`. The drawing is performed such that the text's baseline is at `y`, and the left side of the first character starts at `x`.

At most `len` characters are output, but if `len` is -1, the entire string is output. The results are undefined if `len` is greater than the length of the string. If `len` is -1, the string `s` must be `NULL`-terminated.

Text is drawn in the current font. Normally, only the "ink" or foreground making up the characters is transferred during drawing. Therefore, if text is drawn on top of existing graphics, the background will show through and around the text. However, if the `opaque_text` member of the current `DRAW_CTOOLS` is set to `TRUE`, the text and its opaque background will cover whatever is behind the text. The current `CPEN` and `CBRUSH` are ignored. Text is always drawn in the current foreground color.

ASCII control characters (e.g., tab, backspace, or return) in the string are ignored. Text layout implied by these controls must instead be achieved through drawing the text in segments and positioning each segment in the window with appropriate `x` and `y` values.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`
- `s` must be a valid `NULL`-terminated string

Implementation Note

Normally, TASK_WIN is not a valid window for this function. However, on the XVT/Win32 platforms, you can set the non-portable attribute ATTR_WIN_PM_DRAWABLE_TWIN before the call to xvt_app_create. In that case, TASK_WIN would be a valid window for this function.

See Also

DRAW_CTOOLS
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_app_create

For a diagram that depicts the positioning of text, see the "Drawing and Pictures" chapter in the *XVT Portability Toolkit Guide*

Example

In this code, the first call to xvt_dwin_draw_text draws characters on a white background. In the second call the text is drawn on a gray background. Note that the gray background makes the characters hard to read on a system where gray is synthesized by “dithering” black-and-white pixels (on a monochrome Mac, for example).

```
RCT rct;
CBRUSH brush; xvt_dwin_set_std_cpen(
    win, TL_PEN_BLACK);
xvt_dwin_set_font_family(win, XVT_FFN_HELVETICA);
xvt_dwin_set_font_style(win, XVT_FS_BOLD);
xvt_dwin_set_font_size(win, 24);
xvt_dwin_set_std_cbrush(win, TL_BRUSH_WHITE);
xvt_rect_set(&rct, 100, 40, 300, 90);
xvt_dwin_draw_rect(win, &rct);
xvt_dwin_draw_text(win, 110, 75, "Hello World", -1);
    /* very legible */
 brush.pat = PAT_DIAGCROSS;
brush.color = COLOR_GRAY;
xvt_dwin_set_cbrush(win, &brush);
win_xvt_rect_set(win, &rct, 100, 100, 300, 150);
xvt_dwin_draw_rect(win, &rct);
xvt_dwin_draw_text(win, 110, 135, "Hello World", -1);
    /* barely legible */
```

To draw text that’s guaranteed to be readable without first drawing a rectangle, use an opaque background:

```
DRAW_CTOOLS tools;
xvt_app_get_default_ctools(&tools);
tools.opaque_text = TRUE;
xvt_dwin_set_draw_ctools(win, &tools);
xvt_dwin_draw_text(win, 110, 75, "Hello World", -1);
```

xvt_dwin_get_clip

Get a Clipping Rectangle for a Window

Summary

```
RCT *xvt_dwin_get_clip(WINDOW win, RCT *rctp)
```

`WINDOW win`

Window for which to get the clipping rectangle. It can be any regular window, print window, or `XVT_PIXMAP`.

`RCT *rctp`

Clipping rectangle.

Description

This function gets the clipping rectangle for any regular or print window. The clipping rectangle limits drawing in a window to a particular rectangle, and is set by a previous call to `xvt_dwin_set_clip`.

The clipping rectangle returned by `xvt_dwin_get_clip` is relative to the coordinates of the `WINDOW`, and is stored in the `RCT` pointed to by `rctp`. If no previous call to `xvt_dwin_set_clip` has been made, then `xvt_dwin_get_clip` returns a rectangle at least as large as the client area.

Return Value

`RCT` pointed to by `rctp`.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Drawable windows, print windows, and `XVT_PIXMAP`s are valid values for `win`
- `rctp` must be a valid pointer to a rectangle

Implementation Note

You cannot get a clipping rectangle for the task window unless the application is running on the XVT/Win32/64 platforms, and the attribute ATTR_WIN_PM_DRAWABLE_TWIN has been set.

Normally, TASK_WIN is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute ATTR_WIN_PM_DRAWABLE_TWIN before the call to xvt_app_create. In that case, TASK_WIN would be a valid window for this function.

See Also

RCT
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_dwin_set_clip

xvt_dwin_get_draw_ctools

Get Color Drawing Tools

Summary

```
DRAW_CTOOLS *xvt_dwin_get_draw_ctools(WINDOW win,
     DRAW_CTOOLS *ctoolsp)
```

`WINDOW win`

Window whose color drawing tools are being retrieved.

`DRAW_CTOOLS *ctoolsp`

Pointer to a set of color drawing tools.

Description

This function gets the current set of color drawing tools for win. The argument points to a DRAW_CTOOLS structure that will contain the retrieved tools. The main purpose for calling this function is to save the tools for later restoration with a call to xvt_dwin_set_draw_ctools. Another purpose for this function is to allow your application to modify fields that cannot be set directly, such as opaque_text.

Note: This function does not get the drawing font for a window. To accomplish this task, you can call xvt_dwin_get_font.

Return Value

Value of `ctoolsp` argument.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`
- `ctoolsp` must not be `NULL`

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
CBRUSH
CPEN
DRAW_CTOOLS
M_* Values for DRAW_MODE
TASK_WIN
WINDOW
XVT_FNTID
XVT_PIXMAP
xvt_app_create
xvt_dwin_set_draw_ctools
xvt_dwin_get_font
```

Example

In this example, pairing the calls to `xvt_dwin_get_draw_ctools` and `xvt_dwin_set_draw_ctools` is used to avoid disturbing the tool settings for a window:

```
/* draw selection box around "objp" */
RCT rct;
DRAW_CTOOLS save_tools, t;
xvt_dwin_get_draw_ctools(win, &save_tools);
xvt_app_get_default_ctools(&t);
t.pen.width = FRM_WIDTH;
t.brush.pat = PAT_HOLLOW;
t.mode = M_XOR;
xvt_dwin_set_draw_ctools(win, &t);
xvt_rect_set(&rct,
    objp->left - FRM_WIDTH, objp->top - FRM_WIDTH,
    objp->right + FRM_WIDTH,
    objp->bottom + FRM_WIDTH);
xvt_dwin_draw_rect(win, &rct);
xvt_dwin_set_draw_ctools(win, &save_tools);
```

Note: Also see the example for `xvt_win_trap_pointer`.

xvt_dwin_get_font*

xvt_dwin_get_font* Funtions

```
xvt_dwin_get_font
xvt_dwin_get_font_app_data
xvt_dwin_get_font_family
xvt_dwin_get_font_family_mapped
xvt_dwin_get_font_metrics
xvt_dwin_get_font_native_desc
xvt_dwin_get_font_size
xvt_dwin_get_font_size_mapped
xvt_dwin_get_font_style
xvt_dwin_get_font_style_mapped
```

xvt_dwin_get_font

Get Logical Font Information for a Window

Summary

```
XVT_FNTID xvt_dwin_get_font(WINDOW win)

WINDOW win
```

Window whose logical font is being inquired.

Description

This function gives an application information about the logical font associated with a drawable window. It does this by copying the window's logical font information into a new logical font and returning it. The application's calling function owns the logical font and is responsible for destroying it (with `xvt_font_destroy`).

Changes you make to this logical font do not affect the logical font used by the window. To change the window's logical font, you must call `xvt_dwin_set_font` or any of the `f0 attribute setting functions`.

Return Value

A copy of the logical font associated with the window.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`
- The window is `NULL`, invalid, or non-drawable

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
WINDOW
XVT_FNTID
XVT_PIXMAP
xvt_dwin_get_font*
xvt_dwin_set_font
xvt_dwin_set_font_*
xvt_font_destroy
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

For getting and setting font attributes for a window, is it simpler and more efficient to use `f0 and xvt_dwin_set_font*` functions, like this:

```
WINDOW window;
long size;
...
size = xvt_dwin_get_font_size(window);
xvt_dwin_set_font_size(window, size * 2);
```

than it is to use `xvt_font_get_*` and `xvt_font_set_*` with `xvt_dwin_get_font`, like this:

```
WINDOW window;
long size;
XVT_FNTID font_id;
...
font_id = xvt_dwin_get_font(window);
size = xvt_font_get_size(font_id);
xvt_font_set_size(font_id, size * 2);
xvt_dwin_set_font(window, font_id);
xvt_font_destroy(font_id);
```

xvt_dwin_get_font_app_data

Get the Application Data From a Window's Font

Summary

```
long xvt_dwin_get_font_app_data(WINDOW win)
```

`WINDOW win`

Window from which to get the logical font application data.

Description

This function gets the application data from the specified window's logical font.

This function behaves just like `xvt_font_get_app_data`, except that it applies to the logical font owned by the window, instead of to an application-specific logical font.

Return Value

The application data from the window's logical font if successful; `0` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
WINDOW
XVT_PIXMAP
xvt_dwin_get_font
xvt_dwin_set_font_app_data
xvt_font_get_app_data
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

For information on the use of the `xvt_dwin_get_font*` and `xvt_dwin_set_font*` functions, see the example for `xvt_dwin_get_font`.

xvt_dwin_get_font_family

Get the Family from a Window's Font

Summary

```
BOOLEAN xvt_dwin_get_font_family(WINDOW win,
     char* buf, long max_buf)
```

`WINDOW win`

Window from which to get font family.

`char* buf`

Buffer into which family is put.

`long max_buf`

Maximum size of buffer, in bytes.

Description

This function gets the family from the specified window's logical font and places it into the application-supplied buffer. If an error occurs, the buffer is filled with `NULL`.

This function behaves just like `xvt_font_get_family`, except that it applies to the logical font owned by the window, instead of to an application-specific logical font.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`
- `win` must be a drawable window
- `buf` must be a valid character pointer
- family should fit into `buf`
- `max_buf` should be greater than zero

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

`WINDOW`
`XVT_PIXMAP`
`xvt_dwin_get_font`
`xvt_dwin_set_font_family`
`xvt_font_get_family`

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

For information on the use of the `xvt_dwin_get_font_*` and `xvt_dwin_set_font_*` functions, see the example for `xvt_dwin_get_font`.

xvt_dwin_get_font_family_mapped

Get the Mapped Family from a Window's Font

Summary

```
BOOLEAN xvt_dwin_get_font_family_mapped(WINDOW
    win, char* buf, long max_buf)
```

`WINDOW win`

Window from which to get font family.

`char* buf`

Buffer into which family is to be put.

`long max_buf`

Maximum size of buffer, in bytes.

Description

This function gets the mapped family from the specified window's logical font and places it into the application-supplied buffer. If an error occurs, the buffer is filled with `NULL`. If the window's logical font is not already mapped, this function maps it.

This function behaves just like `xvt_font_get_family_mapped`, except that it applies to the logical font owned by the window, instead of to an application-specific logical font. Also, unlike `xvt_font_get_family_mapped`, if the logical font is not mapped, this function maps it, rather than generating an error.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)

- `win` is not a dialog or control
- Print windows and `XVT_PIXMAPs` are valid values for `win`
- `buf` must be a valid character pointer
- family should fit into `buf`
- `max_buf` should be greater than zero

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
WINDOW
XVT_PIXMAP
xvt_dwin_get_font
xvt_dwin_set_font_family
xvt_font_get_family_mapped
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

For information on the use of the `xvt_dwin_get_font*` and `xvt_dwin_set_font*` functions, see the example for `xvt_dwin_get_font`.

xvt_dwin_get_font_metrics

Get Mapped Logical Font Metrics for a Window

Summary

```
void xvt_dwin_get_font_metrics(WINDOW win,
    int *leadingp, int *ascentp, int *descentp)
```

`WINDOW win`

Window whose mapped logical font metrics are being queried.

`int *leadingp`

Pointer to font’s leading.

`int *ascentp`

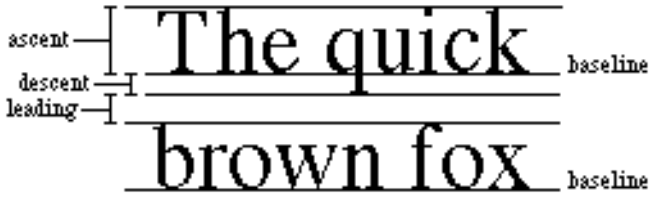
Pointer to font's ascent.

`int *descentp`

Pointer to font's descent.

Description

This function lets you quickly get metrics for a mapped logical font in a window. It gets three attributes of `win`'s current logical font: leading, ascent, and descent; see figure below. These values are returned through the corresponding integer-pointer arguments. If any of these three pararmeters is `NULL`, that particular metric isn't returned.



Font metrics

If the application previously set a window's logical font with `xvt_dwin_set_font`, or with any of the `xvt_dwin_set_font*` attribute setting functions, this inquiry returns metrics for that logical font.

For normally spaced text, you should use a line spacing equal to the sum of the three metric values.

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on XVT/Win32/64, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before calling `xvt_app_create`, in which case `TASK_WIN` would be a valid window for this function.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAPs` are valid values for `win`

- The window is `NULL` or invalid

See Also

```
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_app_create
xvt_dwin_get_font
xvt_dwin_draw_text
xvt_dwin_set_font
xvt_dwin_set_font_*
xvt_font_get_metrics
xvt_dwin_get_text_width
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

See the examples for `xvt_dwin_get_font` and `xvt_dwin_get_text_width`.

xvt_dwin_get_font_native_desc

Get The Native Font Descriptor from a Window's Font

Summary

```
BOOLEAN xvt_dwin_get_font_native_desc(WINDOW win,
    char* buf, long max_buf)
```

`WINDOW win`

Window from which to get native font descriptor.

`char* buf`

Buffer into which native descriptor is to be put.

`long max_buf`

Maximum size of buffer, in bytes.

Description

This function gets the native font descriptor from the specified window's logical font and places it into the application-supplied buffer.

This function behaves just like `xvt_font_get_native_desc`, except that it applies to the logical font owned by the window, instead of to an application-specific logical font.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- `win` must be a drawable window
- Print windows and `XVT_PIXMAP`s are valid values for `win`
- The native descriptor must fit into `buf`
- `buf` must be a valid character pointer
- `max_buf` must be greater than zero

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
WINDOW
XVT_PIXMAP
xvt_dwin_get_font
xvt_dwin_set_font_native_desc
xvt_font_get_native_desc
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

For information on the use of the `xvt_dwin_get_font_*` and `xvt_dwin_set_font*` functions, see the example for `xvt_dwin_get_font`.

xvt_dwin_get_font_size

Get the Size from a Window's Font

Summary

```
long xvt_dwin_get_font_size(WINDOW win)
```

`WINDOW win`

Window from which to get the font size.

Description

This function gets the size from the specified window's logical font. This function behaves just like `xvt_font_get_size`, except that it applies to the logical font owned by the window, instead of to an application-specific logical font.

Return Value

The logical font size if successful; `0` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
WINDOW
XVT_PIXMAP
xvt_dwin_set_font
xvt_dwin_set_font_size
xvt_font_get_size
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

For information on the use of the `xvt_dwin_get_font*` and `xvt_dwin_set_font*` functions, see the example for `xvt_dwin_set_font`.

xvt_dwin_get_font_size_mapped

Get the Mapped Size from a Window's Font

Summary

```
long xvt_dwin_get_font_size_mapped(WINDOW win)
```

`WINDOW win`

Window from which to get the mapped size.

Description

This function gets the mapped size from the specified window's logical font. If the window's logical font is not already mapped, this function maps it.

This function behaves just like `xvt_font_get_size_mapped`, except that it applies to the logical font owned by the window, instead of to an application-specific logical font. Also, unlike `xvt_font_get_size_mapped`, if the logical font is not mapped, this function maps it, rather than generating an error.

Return Value

The font size if successful; `0` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-

portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
WINDOW
XVT_PIXMAP
xvt_dwin_get_font
xvt_dwin_set_font_size
xvt_font_get_size_mapped
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

For information on the use of the `xvt_dwin_get_font*` and `xvt_dwin_set_font*` functions, see the example for `xvt_dwin_get_font`.

xvt_dwin_get_font_style

Get the Style from a Window's Font

Summary

```
XVT_FONT_STYLE_MASK xvt_dwin_get_font_style(WINDOW win)
```

`WINDOW win`

Window from which to get the style.

Description

This function gets the style from the specified window's logical font.

This function behaves just like `xvt_font_get_style`, except that it applies to the logical font owned by the window, instead of to an application-specific logical font.

Return Value

The style mask if successful; `XVT_FS_NONE` if no styles apply to the logical font, or if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
WINDOW
XVT_FS_* Constants
XVT_FONT_STYLE_MASK
XVT_PIXMAP
xvt_dwin_get_font
xvt_dwin_set_font_style
xvt_font_get_style
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

For information on the use of the `xvt_dwin_get_font*` and `xvt_dwin_set_font*` functions, see the example for `xvt_dwin_get_font`.

xvt_dwin_get_font_style_mapped

Get the Mapped Style from a Window's Font

Summary

```
XVT_FONT_STYLE_MASK
     xvt_dwin_get_font_style_mapped(WINDOW win)

WINDOW win
```

Window from which to get the mapped style.

Description

This function gets the mapped style from the specified window's logical font. If the window's logical font is not already mapped, this function maps it.

This function behaves just like `xvt_font_get_style_mapped`, except that it applies to the logical font owned by the window, instead of to an application-specific logical font. Also, unlike `xvt_font_get_style_mapped`, if the logical font is not mapped, this function maps it, rather than generating an error.

Return Value

The style mask if successful; `XVT_FS_NONE` if no styles apply to the mapped font, or if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
WINDOW
XVT_FS_* Constants
XVT_FONT_STYLE_MASK
XVT_PIXMAP
xvt_dwin_get_font
xvt_dwin_set_font_style
xvt_font_get_style_mapped
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

For information on the use of the `xvt_dwin_get_font*` and `xvt_dwin_set_font*` functions, see the example for `xvt_dwin_get_font`.

xvt_dwin_get_text_width

Get Width of Text String

Summary

```
int xvt_dwin_get_text_width(WINDOW win, char *s,
     int len)
```

`WINDOW win`

Window whose current mapped logical font information is being queried.

`char *s`

String whose width is being measured.

`int len`

Number of characters, or all characters if `len` is -1.

Description

This function gets the width in pixels of the text string `s` using `win`'s current logical font. At most, `len` characters are considered, or all characters in `s` if `len` is -1. This function is useful for calculating text layout, especially word wrapping.

You have to set the current logical font with `xvt_dwin_set_font` or with any of the `xvt_dwin_set_font*` attribute setting functions before you call `xvt_dwin_get_text_width`, even if you don't plan to draw in that logical font. Otherwise, you will get the text width for whatever logical font was set previously.

To get the width of a string made of several different logical fonts (e.g., when the size or style varies), call `xvt_dwin_get_text_width` for the substrings that share a common logical font and add up the widths. Using a `len` argument other than -1 is handy for this because the substrings need not be `NULL`-terminated. The text width is not always a sum of the widths of individual characters.

Return Value

The width in pixels if successful; -1 if an error occurs.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- win must be a valid XVT WINDOW of type W_* (except W_SCREEN and W_TASK)
- win is not a dialog or control
- Print windows and XVT_PIXMAPs are valid values for win

Implementation Note

Normally, TASK_WIN is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute ATTR_WIN_PM_DRAWABLE_TWIN before the call to xvt_app_create. In that case, TASK_WIN would be a valid window for this function.

See Also

TASK_WIN
WINDOW
XVT_PIXMAP
xvt_app_create
xvt_dwin_draw_text
xvt_dwin_set_font
xvt_dwin_set_font_*

Example

```
int ascent, descent, width;
char *text;
RCT rect;
...
/* draw text with box around it */
xvt_dwin_get_font_metrics(window, NULL, &ascent,
    &descent);
width = xvt_dwin_get_text_width(window, text, -1);
xvt_rect_set(&rect, 50, 100 - ascent, 50 + width,
    100 + descent);
xvt_dwin_draw_rect(window, &rect);
xvt_dwin_draw_text(window, 50, 100, text, -1)
```

xvt_dwin_invalidate_rect

Schedule a Rectangular Area for Updating

Summary

```
void xvt_dwin_invalidate_rect(WINDOW win, RCT *rctp)

WINDOW win
```

Window containing the rectangular region to be updated. It can be any regular XVT window, but it cannot be a screen window, dialog, control, print window, or `XVT_PIXMAP`.

`RCT *rctp`

Pointer to the invalid rectangle. If `NULL`, the entire client area is invalidated.

Description

This function tells XVT that the contents of a rectangular region of a `WINDOW` is invalid and should be redrawn.

`rctp` should point to `RCT`, specifying the invalid rectangle in the coordinates of `win`. If `rctp` is `NULL`, the entire client area is considered to be invalid. `RCT` must not specify an empty rectangle, as the resulting behavior is undefined.

After your application calls `xvt_dwin_invalidate_rect`, if the event mask for the `WINDOW` does not screen out `E_UPDATE` events, XVT sends an `E_UPDATE` event to the event handler for the `WINDOW`. The `v.update.rct` field in the `EVENT` structure will contain a rectangle the same size or larger than the rectangle defined by `rctp`.

Unless the conditions described in the "`E_UPDATE` Events" section of "Events" in the *XVT Portability Toolkit Guide* apply, calling this function is the preferred way to cause something to be drawn in a window. For the reasons explained there, it is preferred to redrawing the area directly.

You must not assume anything about the generation of `E_UPDATE` event(s), such as when they will be generated, how many will be generated, or what region they will cover (except that they will include the invalidated region). To force the `E_UPDATE` event(s) to be processed immediately, call the function `xvt_dwin_update`. When updating disjointed regions via multiple `xvt_dwin_invalidate_rect` calls, insert `xvt_dwin_update` between them to speed updates on some systems. If you choose to use this approach, be careful to handle the resulting recursion properly.

If the rectangle you are invalidating has a border that you want redrawn, you might need to increase the rectangle dimensions by the line width on all sides.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- `W_PIXMAPS` and `W_PRINT` are not valid windows
- A non-`NULL` rectangle must be valid
- `xvt_dwin_invalidate_rect` must not be called during an `E_UPDATE` event

Implementation Note

On XVT/Win32, `xvt_dwin_invalidate_rect` can be called for the task window if the attribute `ATTR_WIN_PM_DRAWABLE_TWIN` was set when the application was started.

See Also

```
RCT
TASK_WIN
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
E_UPDATE
XVT_PIXMAP
xvt_dwin_update
```

The "`E_UPDATE` Events" section of the "Events" chapter in the *XVT Portability Toolkit Guide*

xvt_dwin_is_update_needed

Test if a Rectangle Requires Updating

Summary

```
BOOLEAN xvt_dwin_is_update_needed(WINDOW win,
     RCT *rctp)
```

`WINDOW win`

Regular or print window in which to test a rectangle.

`RCT *rctp`

Specified rectangle. If the `RCT*` parameter to this function is an empty rectangle, this function returns `FALSE`.

Description

This function is used to optimize drawing in regular or print windows. Calling `xvt_dwin_is_update_needed` will tell your application whether or not the rectangle specified by `rctp` needs to be redrawn. For non-print windows, it can be called only in response to an `E_UPDATE` event. For print windows, it can be called within a printing loop (when `xvt_print_get_next_band` returns a non-`NULL` result). Like drawing functions, this function uses window-relative coordinates.

If your application performs a lot of computation when drawing, then you should draw only the part that needs to be redrawn. This will make your application respond faster to updates, and print faster.

However, calling `xvt_dwin_is_update_needed` is always optional, because you can draw the entire window contents if you want to. XVT will clip away drawing that falls outside the update region or print band. Thus, calling `xvt_dwin_is_update_needed` only saves the cost of executing the drawing functions--it doesn't cut down on the drawing itself, which is automatically minimized.

Applications that consist of a collection of objects drawn at arbitrary places on the window might find calling `xvt_dwin_is_update_needed` to be better suited than using the rectangle provided in the `v.update.rct` field of an `E_UPDATE` event, or returned by `xvt_print_get_next_band`. In this case, your application calls `xvt_dwin_is_update_needed` once for the bounding rectangle of each object, and draws that object only if `xvt_dwin_is_update_needed` returns `TRUE`.

Applications that arrange drawing in regular rows and columns might find using the rectangle provided in the `v.update.rct` field of an `E_UPDATE` event, or returned by `xvt_print_get_next_band`, to be better suited than calling `xvt_dwin_is_update_needed`. In this case, your application can computationally determine the set of rows and columns needing to be drawn in a straightforward fashion.

Return Value

`TRUE` if the argument rectangle overlaps the region that needs updating; `FALSE` otherwise.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not an `XVT_PIXMAP`, dialog, or control

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
E_UPDATE
RCT
TASK_WIN
WINDOW
xvt_app_create
xvt_print_get_next_band
```

The "Drawing and Pictures" chapter in the *XVT Portability Toolkit Guide*

xvt_dwin_open_pict

Prepare to Encapsulate Picture

Summary

```
BOOLEAN xvt_dwin_open_pict(WINDOW win, RCT *rctp)
```

`WINDOW win`

Window in which the picture is to be encapsulated.

`RCT *rctp`

Bounding rectangle.

Description

This function starts the process of encapsulating drawing operations into a `PICTURE` by diverting all subsequent drawing operations intended for `win`. Only one open picture is allowed for the window, and only the drawing that occurs within the rectangle pointed to by `rctp` becomes part of the picture. That rectangle, shifted upward and leftward to have `top` and `left` coordinates of zero, becomes the frame rectangle for the `PICTURE`. If you need to encapsulate a

PICTURE larger than the client area of win, the frame rectangle can exceed the window boundaries.

You don't get the PICTURE object until you call xvt_dwin_close_pict.

Return Value

TRUE if successful; FALSE if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- win must be a valid XVT WINDOW of type W_* (except W_SCREEN and W_TASK)
- win is not an XVT_PIXMAP, dialog, or control
- rctp must be a valid non-empty rectangle

Implementation Note

Normally, TASK_WIN is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute ATTR_WIN_PM_DRAWABLE_TWIN before the call to xvt_app_create. In that case, TASK_WIN would be a valid window for this function.

Your application should clear the picture before drawing, as the "initial" contents of the picture are not guaranteed to be portable.

See Also

PICTURE
RCT
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_app_create
xvt_dwin_clear
xvt_dwin_close_pict

The "Drawing and Pictures" chapter in the *XVT Portability Toolkit Guide*

Example

See the example under xvt_dwin_close_pict.

xvt_dwin_scroll_rect

Scroll a Window's Pixels

Summary

```
void xvt_dwin_scroll_rect(WINDOW win, RCT *rctp,
     int dh, int dv)
```

`WINDOW win`

Window whose pixels are to be scrolled.

`RCT *rctp`

Rectangle. If the `RCT*` parameter to this function is an empty rectangle, no scrolling occurs.

`int dh`

Controls horizontal scrolling. If `dh` is positive, scrolling is to the right by `dh` pixels. If `dh` is negative, scrolling is to the left by `dh` pixels. `dh` can be zero.

`int dv`

Controls vertical scrolling. If `dv` is positive, scrolling is downward by `dv` pixels. If `dv` is negative, scrolling is upward by `dv` pixels. `dv` can be zero.

Description

This function scrolls the pixels bounded by `rctp` in the client area of `win`. No pixels outside of the rectangle are affected. Pixels scrolled beyond the boundary of `rctp` are discarded.

An `E_UPDATE` event is automatically generated for the part of the rectangle whose pixels were scrolled away. Before `xvt_dwin_scroll_rect` returns, this event is recursively sent to `win`'s event handler. If the client area being scrolled is partially obscured by other windows, including child windows and controls, then the resulting `E_UPDATE` event might encompass a larger area than just the rectangle exposed by the scrolling. Because of this, your application must not make assumptions about the `E_UPDATE` events that will be generated during scrolling. If the client area being scrolled contains child windows or controls, these objects will not be scrolled by `xvt_dwin_scroll_rect`.

Normally, this function is called when your application is changing the view of a document. Usually, your application keeps an internal

data structure reflecting the view of the document, and part of the data structure indicates the origin of the window viewport into that document. Before you scroll a window's contents, you should first adjust your internal origin, so that the recursively generated E_UPDATE event is encountered by an event handler whose origin has already been properly set.

If you are scrolling your window in response to an E_VSCROLL event, remember that when you receive a line up or page up event you want to move the pixels downward so that the dv argument to xvt_dwin_scroll_rect is positive. When you get a line down or page down, dv is negative. A similar relationship holds for E_HSCROLL events.

Before scrolling a window's pixels, you must ensure that the client area is valid, by calling xvt_dwin_update. This call to update a window should be made even before you change your application's internal viewport origin.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- win must be a valid XVT WINDOW of type W_* (except W_SCREEN and W_TASK)
- win is not a print window, dialog, or control
- This function must not be called during an E_UPDATE event

Implementation Note

Normally, TASK_WIN is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute ATTR_WIN_PM_DRAWABLE_TWIN before the call to xvt_app_create. In that case, TASK_WIN would be a valid window for this function.

An E_UPDATE event is not generated for XVT_PIXMAPS.

See Also

CBRUSH
E_HSCROLL
E_UPDATE
E_VSCROLL
RCT
TASK_WIN
WINDOW
xvt_app_create
xvt_dwin_update

The "E_UPDATE, E_HSCROLL, and E_VSCROLL Events" section of the "Events" and the "Windows" chapters in the *XVT Portability Toolkit Guide*

xvt_dwin_set_back_color

Set Background Color

Summary

```
void xvt_dwin_set_back_color(WINDOW win,
    COLOR color)
```

`WINDOW win`

Window whose background color is to be set.

`COLOR color`

Background color.

Description

This function sets the background color for `win`. The background color is used for the spaces between the hatch marks of a patterned brush, for the background of icons, and for the text background when `opaque_text` is set (see `DRAW_CTOOLS`).

Do not confuse the background color set by this function with any sort of automatic background painting. Your application must explicitly paint the background color of a window when it receives an `E_UPDATE` event, usually by calling `xvt_dwin_clear`.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to

`xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

`COLOR`
`DRAW_CTOOLS`
`TASK_WIN`
`WINDOW`
`XVT_PIXMAP`
`xvt_app_create`
`xvt_dwin_clear`
`xvt_dwin_set_draw_ctools`
`xvt_dwin_set_fore_color`

The "Drawing and Pictures" chapter in the *XVT Portability Toolkit Guide*

xvt_dwin_set_cbrush

Set Color Brush Tool

Summary

```
void xvt_dwin_set_cbrush(WINDOW win, CBRUSH *cbrushp)
```

`WINDOW win`

Window whose color brush tool is to be set.

`CBRUSH *cbrushp`

Pointer to the color brush.

Description

This function sets the current color brush for `win`. Setting the current color brush affects the following:

- The brush pattern
- The color of hatched marks in hatched brushes
- The color of solid fill brushes

Recall that brushes are used for filling the interior of shapes. `cbrushp` should point to a completely initialized `CBRUSH` structure.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
CBRUSH
PAT_* Values for PAT_STYLE
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_app_create
xvt_dwin_draw_oval
xvt_dwin_set_draw_ctools
```

The "Drawing and Pictures" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_dwin_draw_oval`.

xvt_dwin_set_clip

Set a Clipping Rectangle for Window

Summary

```
void xvt_dwin_set_clip(WINDOW win, RCT *rctp)
```

`WINDOW win`

Window whose clipping rectangle is to be set.

`RCT *rctp`

Pointer to the clipping rectangle.

Description

This function sets the clipping rectangle for any non-print window, print window, or pixmap. The clipping rectangle limits drawing in a window to a particular rectangle. Pixels outside of this rectangle are not affected by subsequent drawing calls. Setting clipping rectangles is especially useful when drawing shapes that XVT doesn't support directly, such as semi-circles.

The clipping rectangle you specify is relative to the coordinates of the `WINDOW`, and is stored in `RCT` pointed to by `rctp`. Setting `rctp` to `NULL` restores the clipping rectangle to the client area of `win`, which is the default. If the rectangle pointed to by `rctp` has a height or width of zero (i.e., it is an empty rectangle), all drawing to the window is clipped. If the rectangle pointed to by `rctp` is an empty rectangle, all drawing to the window is clipped.

As with the drawing tools, an application must be certain that the clip area is set appropriately before any drawing. For example, if you have set the clip area to a sub-rectangle of a window during a drawing operation, and you receive an `E_UPDATE` for the entire window, then you should reset the clip rectangle before attempting to update the window's client area.

After a clipping rectangle has been set by a call to `xvt_dwin_set_clip`, your application can retrieve the clipping rectangle by calling `xvt_dwin_get_clip`.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

E_UPDATE
RCT
WINDOW
XVT_PIXMAP
xvt_dwin_get_clip

The "Windows" chapter in the *XVT Portability Toolkit Guide*

Example

```
RCT rect;
WINDOW window;
char *text;
...
/* draw text within rect */
xvt_dwin_set_clip(window, &rect);
xvt_dwin_draw_text(window, rect.left,
     (rect.bottom + rect.top)/2, text, -1);
```

xvt_dwin_set_cpen

Set Color Pen Tool

Summary

```
void xvt_dwin_set_cpen(WINDOW win, CPEN *cpenp)
```

`WINDOW win`

Window whose color pen tools are to be set.

`CPEN *cpenp`

Pointer to the color pen tool.

Description

This function sets the current color pen for `win`. Setting the current color pen will affect the pen pattern and the color of hatched marks in hatched pens, as well as the color of solid fill pens. Also recall that pens are used for drawing the outline of shapes. `cpenp` should point to a completely initialized `CPEN` structure.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)

- win is not a dialog or control
- Print windows and XVT_PIXMAPs are valid values for win
- cpenp must point to a valid CPEN structures

Implementation Note

On the XVT/Win32/64 platforms, you can set the non-portable attribute ATTR_WIN_PM_DRAWABLE_TWIN before the call to xvt_app_create. In that case, TASK_WIN would be a valid window for this function.

See Also

CPEN
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_app_create
xvt_dwin_set_draw_ctools

The "Drawing and Pictures" chapter in the *XVT Portability Toolkit Guide*

xvt_dwin_set_draw_ctools

Set the Color Drawing Tools

Summary

```
void xvt_dwin_set_draw_ctools(WINDOW win,
     DRAW_CTOOLS *ctoolsp)
```

`WINDOW win`

Window whose color drawing tools are being set.

`DRAW_CTOOLS *ctoolsp`

Pointer to the color drawing tools.

Description

This function sets the current DRAW_CTOOLS for win. The current DRAW_CTOOLS affect all subsequent drawing into that window. ctoolsp should point to a completely initialized DRAW_CTOOLS structure, such as one obtained from either xvt_dwin_get_draw_ctools or xvt_app_get_default_ctools.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`
- `ctoolsp` must point to a valid `DRAW_CTOOLS` structure
- The window's font is not included in the window's drawing tools. (Call `xvt_dwin_set_font` to set the window's font.)

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

`DRAW_CTOOLS`
`DRAW_MODE`
`TASK_WIN`
`WINDOW`
`XVT_PIXMAP`
`xvt_app_create`
`xvt_app_get_default_ctools`
`xvt_dwin_get_draw_ctools`
`xvt_dwin_set_font`
`xvt_win_trap_pointer`

The "Drawing and Pictures" chapter in the *DSC Guide*

Example

See the examples for `xvt_dwin_get_draw_ctools` and `xvt_win_trap_pointer`.

xvt_dwin_set_draw_mode

Set the Current Drawing Mode

Summary

```
void xvt_dwin_set_draw_mode(WINDOW win, DRAW_MODE mode)
```

`WINDOW win`

Window whose current drawing mode is to be set.

`DRAW_MODE mode`

Drawing mode.

Description

This function sets the `DRAW_MODE` for `win`. The drawing mode for a window affects all subsequent drawing into that window.

Note: For print windows, only the `M_COPY` draw mode is assured to work properly, since some print drivers can't handle other modes.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case `TASK_WIN` would be a valid window for this function.

See Also

DRAW_CTOOLS
DRAW_MODE
M_* Values for DRAW_MODE
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_app_create
xvt_dwin_set_draw_ctools

The "Drawing and Pictures" chapter in the *XVT Portability Toolkit Guide*

xvt_dwin_set_font*

xvt_dwin_set_font_* Functions

xvt_dwin_set_font_app_data
xvt_dwin_set_font_family
xvt_dwin_set_font_native_desc
xvt_dwin_set_font_size
xvt_dwin_set_font_style

xvt_dwin_set_font

Set Logical Font Information for a Window

Summary

```
void xvt_dwin_set_font(WINDOW win, XVT_FNTID font_id)
```

`WINDOW win`

Window whose logical font is to be set.

`XVT_FNTID font_id`

Handle of logical font.

Description

This function sets the logical font to be used for drawing in the client area of a window.

The function copies the `font_id` contents into an internal logical font that is owned by the window. This allows the application to reuse `font_id` for other purposes after making this call.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a drawable window, dialog, or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`
- `font_id` must be valid
- `font_id` must be a valid logical font

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on XVT/Win32/64, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before calling `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
XVT_FNTID
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_app_create
xvt_dwin_get_font
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

```
XVT_FNTID fid = xvt_dwin_get_font(window2);
xvt_dwin_set_font(window1, fid );
xvt_font_font_destroy(fid);
```

xvt_dwin_set_font_app_data

Set Application Data for a Logical Font in a Window

Summary

```
void xvt_dwin_set_font_app_data(WINDOW win,
     long app_data)
```

`WINDOW win`

Window whose logical font application data is to be set.

`long app_data`

Application data.

Description

This function behaves just like `xvt_font_set_app_data`, except that it applies to the logical font owned by the window, instead of to an application-specific logical font.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
WINDOW
xvt_dwin_get_font_app_data
xvt_font_set_app_data
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

For information on the use of the `xvt_dwin_get_font*` and `xvt_dwin_set_font*` functions, see the example for `xvt_dwin_get_font`.

xvt_dwin_set_font_family

Set Logical Font Family for a Window

Summary

```
void xvt_dwin_set_font_family(WINDOW win,
     char *family)
```

`WINDOW win`

Window whose logical font family is to be changed.

`char *family`

String containing family name.

Description

This function behaves just like `xvt_font_set_family`, except that it applies to the logical font owned by the window, instead of to an application-specific logical font.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- `family` must be a valid string

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
WINDOW
xvt_dwin_get_font_family
xvt_dwin_get_font_family_mapped
xvt_font_set_family
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

For information on the use of the `xvt_dwin_get_font*` and `xvt_dwin_set_font*` functions, see the example for `xvt_dwin_get_font`.

xvt_dwin_set_font_native_desc

Set Logical Font Native Descriptor for a Window

Summary

```
void xvt_dwin_set_font_native_desc(WINDOW win,
     char *native_font_desc)
```

`WINDOW win`

Window whose logical font native descriptor is to be set.

`char *native_font_desc`

String specification of native font.

Description

This function behaves just like `xvt_font_set_native_desc`, except that it applies to the logical font owned by the window, instead of to an application-specific logical font.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- `native_font_desc` must be a valid string

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

`WINDOW`
`xvt_dwin_get_font`
`xvt_dwin_get_font_native_desc`
`xvt_font_set_native_desc`

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

For information on the use of the `xvt_dwin_get_font*` and `xvt_dwin_set_font*` functions, see the example for `xvt_dwin_get_font`.

xvt_dwin_set_font_size

Set Logical Font Size for a Window

Summary

```
void xvt_dwin_set_font_size(WINDOW win, long size)
```

`WINDOW win`

Window whose logical font size is to be changed.

`long size`

Value of new font size.

Description

This function behaves just like `xvt_font_set_size`, except that it applies to the logical font owned by the window, instead of to an application-specific logical font.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- `size` must be positive

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
WINDOW
xvt_dwin_get_font
xvt_dwin_get_font_size
xvt_dwin_get_font_size_mapped
xvt_font_set_size
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

`For information on the use of the xvt_dwin_get_font*` and `xvt_dwin_set_font*` functions, see the example for `xvt_dwin_get_font`.

xvt_dwin_set_font_style

Set Logical Font Style for a Window

Summary

```
void xvt_dwin_set_font_style(WINDOW win,
     XVT_FONT_STYLE_MASK mask)
```

`WINDOW win`

Window whose logical font style is to be changed.

`XVT_FONT_STYLE_MASK mask`

Font style mask composed of one or more `XVT_FS_*` flag values.

Description

This function behaves just like `xvt_font_set_style`, except that it applies to the logical font owned by the window, instead of to an application-specific logical font.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before the call to `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

```
WINDOW
XVT_FONT_STYLE_MASK
XVT_FS_* Constants
xvt_dwin_get_font
xvt_dwin_get_font_style
xvt_dwin_get_font_style_mapped
xvt_font_set_style
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

For information on the use of the `xvt_dwin_get_font*` and `xvt_dwin_set_font*` functions, see the example for `xvt_dwin_get_font`.

xvt_dwin_set_fore_color

Set Foreground Color

Summary

```
void xvt_dwin_set_fore_color(WINDOW win, COLOR color)

WINDOW win
```

Window whose foreground color is to be set.

`COLOR color`

The foreground color.

Description

This function sets the foreground color for `win`.

Foreground color is used only by the `xvt_dwin_draw_text` and `xvt_dwin_draw_icon` functions, and only for the ink.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- Print windows and `XVT_PIXMAP`s are valid values for `win`

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before calling `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

`COLOR`
`TASK_WIN`
`WINDOW`
`XVT_PIXMAP`
`xvt_app_create`
`xvt_dwin_draw_icon`
`xvt_dwin_draw_text`
`xvt_dwin_set_back_color`
`xvt_dwin_set_draw_ctools`

The "Drawing and Pictures" chapter in the *XVT Portability Toolkit Guide*

xvt_dwin_set_std_cbrush

Set a Standard Brush

Summary

```
void xvt_dwin_set_std_cbrush(WINDOW win, long flag)
```

`WINDOW win`

Window whose standard brush tool is to be set.

`long flag`

`TL_BRUSH_*` constants used to set the standard brush.

Description

This function sets one of several predefined brushes into the drawing tools for `win`. This is a convenience function that allows you to set a common brush tool without taking the normal route of first filling in a `CBRUSH` structure, and then calling `xvt_dwin_set_cbrush`.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a dialog or control
- `win` must be a valid drawable window
- Print windows and `XVT_PIXMAP`s are valid values for `win`
- `flag` must be either `TL_BRUSH_BLACK` or `TL_BRUSH_WHITE`

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on the XVT/Win32/64 platforms, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before calling `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function.

See Also

CBRUSH
TASK_WIN
TL_* Constants
WINDOW
xvt_app_create
xvt_dwin_draw_text
xvt_dwin_set_cbrush
xvt_dwin_set_draw_ctools

Example

See the example for xvt_dwin_draw_text.

xvt_dwin_set_std_cpen

Set a Standard Pen Tool

Summary

```
void xvt_dwin_set_std_cpen(WINDOW win, long flag)
```

WINDOW win

Window whose standard pen tool is to be set.

long flag

TL_PEN_* flags used to set a standard pen.

Description

This function sets one of several predefined pens into the drawing tools for win. This is a convenience function that allows you to set a common pen tool without taking the normal route of first filling in a CPEN structure and then calling xvt_dwin_set_cpen.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- win must be a valid XVT WINDOW of type W_* (except W_SCREEN and W_TASK)
- win is not a dialog or control
- Print windows and XVT_PIXMAPs are valid values for win
- flag must be of type TL_PEN_BLACK, TL_PEN_WHITE, TL_PEN_HOLLOW, or TL_PEN_ROBBER

Implementation Note

Normally, TASK_WIN is not a valid window for this function. However, on the XVT/Win32 platforms, you can set the non-portable attribute ATTR_WIN_PM_DRAWABLE_TWIN before the call to xvt_app_create. In that case, TASK_WIN would be a valid window for this function.

See Also

CPEN
TASK_WIN
WINDOW
XVT_PIXMAP
TL_* Constants
xvt_app_create
xvt_dwin_set_cpen
xvt_dwin_set_draw_ctools

Example

See the example for xvt_dwin_draw_aline.

xvt_dwin_update

Force Update Events to be Processed

Summary

```
void xvt_dwin_update(WINDOW win)

WINDOW win
```

Window to update.

Description

This function updates win immediately by expediting all pending E_UPDATE events. To insure that the window's contents are correct before moving any of the windows pixels, this function should be called prior to calling xvt_dwin_scroll_rect.

Note: If there are any E_UPDATE events to be processed, this function causes an immediate recursive call to the event handler for win. Your application must be prepared for this.

Parameter Validity and Error Conditions

XVT issues an error if any of the following error conditions are not met:

- `win` must be a valid XVT `WINDOW` of type `W_*` (except `W_SCREEN` and `W_TASK`)
- `win` is not a print window, `XVT_PIXMAP`, dialog, or control
- `ATTR_SUPPRESS_UPDATE_CHECK` must be set to `TRUE` when this function is called during an `E_UPDATE` event

See Also

```
ATTR_SUPPRESS_UPDATE_CHECK
E_UPDATE
WINDOW
XVT_PIXMAP
xvt_dwin_invalidate_rect
xvt_dwin_scroll_rect
```

Example

```
RCT rect;
WINDOW window;
int dh, dv;
...
/* scroll window by dh, dv */
xvt_vobj_get_client_rect(window, &rect);
xvt_dwin_update(window);
xvt_dwin_scroll_rect(window, &rect, dh, hv);
```

PTK REFERENCE

INDEX

B

C

E

F

gotolink ptkrefb.fm

H

M

O

P

S

T

U

V

W

X

xvt_errid_*

Error Message Identifiers

```
xvt_errid_create_*
xvt_errid_get_*
xvt_errid_is_*
```

xvt_errid_create_*

Generate Error Message Identifiers

Summary

```
XVT_ERRID xvt_errid_create_*(XVT_ERRID base,
     unsigned short number)
```

`XVT_ERRID base`

Base error message category.

`unsigned short number`

Message number within base category.

Description

This set of macros creates error message identifiers:

Function	Message Identifier Created
`xvt_errid_create_mjr`	Major error category; `base` is ignored
`xvt_errid_create_cat`	Error category; `base` must be major category
`xvt_errid_create_num`	Non-standard error message; `base` must be error category
`xvt_errid_create_std`	Standard error message; `base` must be error category

Error messages are identified using an opaque data type `XVT_ERRID`, which is composed of several fields:

- Message number (16 bits unsigned short)
- Standard message flag (1 bit: distinguishes predefined, standard toolkit messages from the ones defined by an `xvt_errmsg_sig call`)
- Message category minor portion (4 bit)

- Message category major portion (4 bit)

You should not make any assumptions about the individual field position within the identifier. The recommended way to define error message identifiers is by using `xvt_errmsg_def_*` macros, which are processed by the **errscan** tool. This in turn uses the `xvt_errid_create_*` macros in a generated header file.

Return Value

`XVT_ERRID number` identifying an error message or message category.

Parameter Validity and Error Conditions

These macros do not perform any validity checking.

See Also

```
XVT_ERRID
xvt_errid_get_*
xvt_errid_is_*
xvt_errmsg_def_*
xvt_errmsg_sig
```

The"Diagnostics and Debugging" chapter in the *XVT Portability Toolkit Guide*

Example

The `XVT_ERRID` value is created using a hierarchy of definitions, defining the major category first, then deriving a category from it and defining `ERR_ARG_NULL_WIN` as an error within that category.

```
#define ERR_ARGxvt_errid_create_mjr(ERR, 1)
#define ERR_ARG_NULLxvt_errid_create_cat(ERR_ARG, 1)
#define ERR_ARG_NULL_WIN
xvt_errid_create_num(ERR_ARG_NULL, 1)
```

xvt_errid_get_*

Access Error Identifier Components

Summary

```
unsigned long xvt_errid_get_*(XVT_ERRID msg_id)

XVT_ERRID msg_id
```

Parsed error message identifier.

Description

This set of macros provides access to individual components of the error identifier:

Function	Component Accessed
`xvt_errid_get_cat`	Error category
`xvt_errid_get_mjr`	Major category portion
`xvt_errid_get_mnr`	Minor category portion
`xvt_errid_get_num`	Message number

Components are returned as numbers, inverting the process of `xvt_errid_create_*`.

Return Value

A number corresponding to the value of `number` in the `xvt_errid_create_*` function, hiding the actual value position within the identifier.

Parameter Validity and Error Conditions

These macros do not perform any validity checking.

See Also

`XVT_ERRID`
`XVT_ERRMSG_HANDLER`
`xvt_ctl_create`
`xvt_ctl_create_def`
`xvt_dlg_create_def`
`xvt_dlg_create_res`
`xvt_errid_create_*`
`xvt_errid_is_*`
`xvt_errmsg_def_*`
`xvt_errmsg_push_handler`
`xvt_errmsg_sig`
`xvt_win_create`
`xvt_win_create_def`
`xvt_win_create_res`

The "Diagnostics and Debugging" chapter in the *XVT Portability Toolkit Guide*

xvt_errid_is_*

Compare Error Message Identifier Components

Summary

```
BOOLEAN xvt_errid_is_*(XVT_ERRID msg_1,
     XVT_ERRID msg_2)
```

`XVT_ERRID msg_1`

First message identifier compared.

`XVT_ERRID msg_2`

Second message identifier compared.

Description

This set of macros compares individual components of an error message identifier. You can use them to check for a particular error category, major or minor category, or message number. You can also check for a standard message:

Function	Component Compared
`xvt_errid_is_cat`	Error category
`xvt_errid_is_mjr`	Major category portion
`xvt_errid_is_mnr`	Minor category portion
`xvt_errid_is_num`	Message number
`xvt_errid_is_std`	Checks for standard message (`msg_2` is not used)

Return Value

`TRUE` if a given field matches; `FALSE` otherwise.

Parameter Validity and Error Conditions

These macros do not perform any validity checking.

See Also

`XVT_ERRID`
`XVT_ERRMSG_HANDLER`
`xvt_errid_create_*`
`xvt_errid_get_*`
`xvt_errmsg_def_*`
`xvt_errmsg_push_handler`
`xvt_errmsg_sig`

The "Diagnostics and Debugging" chapter in the *XVT Portability Toolkit Guide*

Example

```
if (xvt_errid_is_cat (msg_id, ERR_ARG_NULL_WIN))
     /* compares complete category */
    ...
if (xvt_errid_is_mjr (msg_id, ERR_ARG_NULL_WIN))
     /* compares major category portion only */
    ...
if (xvt_errid_is_mnr (msg_id, ERR_ARG_NULL_WIN))
     /* compares minor category portion only */
    ...
if (xvt_errid_is_std (msg_id, 0))
     /* checks for standard message */
    ...
```

xvt_errmsg_*

Error Handling Functions

```
xvt_errmsg_def_*
xvt_errmsg_get_*
xvt_errmsg_get_text
xvt_errmsg_pop_handler
xvt_errmsg_push_handler
xvt_errmsg_sig
xvt_errmsg_sig_if
xvt_errmsg_sig_std
xvt_errmsg_sig_std_if
```

xvt_errmsg_def_*

Predefine Error Messages for **errscan**

Summary

```
void xvt_errmsg_def_*(XVT_ERRID category,
     const char* suffix, unsigned short number,
     const char *text)
```

`XVT_ERRID category`

Base error message category.

`const char* suffix`

Constant name suffix of the error message identifier.

`unsigned short number`

Message number within a given category.

`const char *text`

Message text.

Description

This set of macros predefines error messages for the **errscan** tool:

Function	Error Message Defined
`xvt_errmsg_def_mjr`	Major error category; `category` must be `ERR`
`xvt_errmsg_def_cat`	An error category; `category` must be an existing major category
`xvt_errmsg_def_num`	An error number; `category` must be an existing application category
`xvt_errmsg_def_std`	A standard message; `category` must be an existing error category

These macros do not generate any executable code; they can appear even outside a code section. Within each category, the suffix and corresponding number must be unique.

Application-defined errors should be derived from the `ERR_APP` major category. Within `ERR_APP`, there can be up to 15 minor categories. For each category, you can define up to 3000 distinct standard messages. You can define another 3000 non-standard messages directly by using the `xvt_errmsg_sig` and `xvt_errmsg_sig_if` macros.

The **errscan** tool uses such definitions to generate the message file **ERRCODES.TXT** and a platform-specific header file **xvt_perr.h**. For each definition, errscan generates a message file entry and two constants in a header file.

Return Value

None.

Parameter Validity and Error Conditions

These macros do not generate any executable code. **errscan** reports any syntactically incorrect `xvt_errmsg_def_*` statements; however, its validity checks are limited.

See Also

```
XVT_ERRID
xvt_errid_create_*
ERRCODES.TXT
xvterr.h
xvt_perr.h
```

The "Diagnostics and Debugging" chapter in the *XVT Portability Toolkit Guide*

Example

An application can define the category `ERR_APP_DOC` and the message `ERR_APP_DOC_EMPTY` using message definition, like this:

```
    xvt_errmsg_def_cat (ERR_APP, "DOC", 5, "Document
error")
    xvt_errmsg_def_std (ERR_APP_DOC, "EMPTY", 1,
"Document is empty")
```

errscan will then generate the following file entries:

ERRCODES.TXT:

```
    0x00150000 ERR_APP_DOC "Document error"
    0x00150001 ERR_APP_DOC_EMPTY "Document is empty"
```

xvt_perr.h:

```
    #define ERR_APP_DOCxvt_errid_create_cat
        (ERR_APP,5)
    #define ERR_APP_DOC_EMPTYxvt_errid_create_std
        (ERR_APP_DOC,1)
    #define TXT_ERR_APP_DOC"Document error"
    #define TXT_ERR_APP_DOC_EMPTY"Document is empty"
```

xvt_errmsg_get_*

Get Information About a Signaled Error

Summary

```
return_type xvt_errmsg_get_*(XVT_ERRMSG err)

XVT_ERRMSG err
```

Error message object handle.

Description

These functions give an error handler all the information about an error signaled by one of the `xvt_errmsg_sig` call forms.

The information is valid only during the error handler scope. If any of the strings returned by one of the inquiries is to be preserved, an error handler must make a copy of this string. Do not attempt to use `xvt_mem_free` to free any const `char*` arguments; they are owned by the `XVT_ERRMSG` object.

The `xvt_errmsg_get_*` functions listed below differ only in what they return:

`xvt_errmsg_get_api_name`
Return type: `const char*`
Return description: Called XVT API function name string (concatenation in case of nested API calls)

`xvt_errmsg_get_cat_text`
Return type: `const char*`
Return description: Text describing message category

`xvt_errmsg_get_code_file`
Return type: `const char*`
Return description: Source filename string

`xvt_errmsg_get_code_line`
Return type: `long`
Return description: Number of the source code line signaling an error

`xvt_errmsg_get_msg_id`
Return type: `XVT_ERRID`
Return description: Error message

`xvt_errmsg_get_msg_text`
Return type: `const char*`
Return description: Message string associated with error message identifier

`xvt_errmsg_get_sev_id`
Return type: `XVT_ERRSEV`
Return description: Severity code from the `xvt_errmsg_sig` call

`xvt_errmsg_get_sev_text`
Return type: `const char*`
Return description: Localized severity code name string

xvt_errmsg_get_tgt_object

Return type: WINDOW

Return description: NULL_WIN or handle to the operation target (window argument from xvt_errmsg_sig call)

Implementation Note

All strings returned by the XVT_ERRMSG object inquiries are retrieved from the error message file, **ERRCODES.TXT**. In case this file cannot be accessed, XVT provides hardcoded English messages for the most frequently used messages. Remaining messages are represented by message number, both decimal and hexadecimal, which allows you to look them up manually in **ERRCODES.TXT**.

You can localize the **ERRCODES.TXT** file (that is, translate it into another language), by replacing quoted message text with other language equivalents.

Parameter Validity and Error Conditions

XVT_ERRMSG object inquiries do not signal any errors. If an argument is invalid, they return an empty ("") string or zero value.

See Also

ATTR_ERRMSG_FILENAME
XVT_ERRMSG
xvt_errmsg_pop_handler
xvt_errmsg_push_handler
xvt_errmsg_sig

The "Diagnostics and Debugging" chapter in the *XVT Portability Toolkit Guide*

xvt_errmsg_get_text

Get a Message from the Error Message File

Summary

```
const char xvt_errmsg_get_text(XVT_ERRMSG errmsg,
     XVT_ERRID msg_id, char* buf, long int bufsiz)
```

XVT_ERRMSG errmsg

Error message object pointer.

XVT_ERRID msg_id

Desired message identifier.

`har* buf`

Buffer for retrieved message.

`long int bufsiz`

Buffer size in bytes.

Description

This function retrieves an arbitrary message from the error message file. The message is processed replacing C language escapes "<fcn", "<fct", "\" and "<f0" with a single character. The message is truncated to the size of the provided buffer.

Contrary to `xvt_errmsg_get_msg_text`, which returns a message for an `XVT_ERRID` contained within an `XVT_ERRMSG` object, this function retrieves an arbitrary message given an explicit `msg_id`.

The `errmsg` argument is *required*, because this function is intended solely for use within error handlers.

Return Value

Pointer to message string stored in `buf` argument.

Parameter Validity and Error Conditions

This function does not signal any errors. A string of `msg_id` value represents non-existent messages.

See Also

```
ATTR_ERRMSG_FILENAME
XVT_ERRID
XVT_ERRMSG
xvt_errmsg_get_*
```

The "Diagnostics and Debugging" chapter in the *XVT Portability Toolkit Guide*

xvt_errmsg_pop_handler

Remove a Temporary Error Handler

Summary

```
void xvt_errmsg_pop_handler(XVT_ERRMSG_HANDLER handler)
```

`XVT_ERRMSG_HANDLER handler`

Error handler function pointer.

Description

This function removes a temporary error handler pushed by the `xvt_errmsg_push_handler`. Popping the handler automatically removes any handler(s) pushed above the specified one. Any stacked handlers not removed at the point of return from the XVT window event handler are removed automatically; this generates a warning.

Return Value

None.

Parameter Validity and Error Conditions

If you use an invalid handler argument (i.e., if you try to remove a handler that was not previously pushed onto the stack with `xvt_errmsg_push_handler`), this function signals a warning.

See Also

`XVT_ERRMSG_HANDLER`
`xvt_errmsg_push_handler`
`xvt_errmsg_sig`

The "Diagnostics and Debugging" chapter in the *XVT Portability Toolkit Guide*

xvt_errmsg_push_handler

Establish a Temporary Error Handler

Summary

```
void xvt_errmsg_push_handler
     (XVT_ERRMSG_HANDLER handler, DATA_PTR context)
```

`XVT_ERRMSG_HANDLER handler`

Error handler function pointer.

`DATA_PTR context`

Context delivered to the application's error handler.

Description

This function establishes a temporary error handler. Any error messages signaled by `xvt_errmsg_sig` (and other signaling calls) are delivered to handlers in this order (from top to bottom):

- The most recently pushed handler
- The less recently pushed handlers
- An application-supplied permanent error handler
- The XVT-provided "last chance" handler

Each of those handlers can either handle the error (returning `TRUE`), or pass it to a subsequent handler (returning `FALSE`).

Any handlers pushed onto the stack within an XVT window event handler must be removed prior to the return from the event handler. Failure to do so results in a warning, which is signaled when the window event handler returns. Stacked handler lifetime is limited to the scope of the XVT window event handler.

Return Value

None.

Parameter Validity and Error Conditions

This call is illegal inside an error handler. If you try to push a handler from within error handler code, this function generates a warning. The warning is delivered to subsequent handlers and the request is ignored.

See Also

```
DATA_PTR
XVT_ERRMSG_HANDLER
xvt_errmsg_push_handler
xvt_errmsg_sig
```

The "Diagnostics and Debugging" chapter in the *XVT Portability Toolkit Guide*

xvt_errmsg_sig

Signal an Error

Summary

```
void xvt_errmsg_sig
     (WINDOW object, XVT_ERRSEV severity,
     XVT_ERRID category, const char* suffix,
     unsigned short number, const char* message)
```

`WINDOW object`

`NULL_WIN` or window handle to object in error.

`XVT_ERRSEV severity`

Error severity.

`XVT_ERRID category`

One of the predefined error categories.

`const char* suffix`

Unique error ID constant suffix literal (for **errscan** only).

`unsigned short number`

Unique error message number within specified category.

`const char* message`

Message prototype literal (for **errscan** only).

Description

This macro signals an abnormal condition to the error message handling facility, which in turn calls one or more error handler(s). Unless caught by application-provided error handlers, the error signal reaches the XVT-provided "last chance" error handler, which posts a dialog corresponding to the error message severity.

The macro strips the `suffix` and `message` arguments. They are used only by the **errscan** tool to generate message file entry, and to generate `#define` constants identifying this message. The constant is formed by adding the `suffix` to the category name. The `suffix` and corresponding message `number` must be unique within a given message category.

Return Value

None.

See Also

```
ATTR_ERRMSG_HANDLER
WINDOW
XVT_ERRID
XVT_ERRSEV
xvt_errmsg_def_*
xvt_errmsg_pop_handler
xvt_errmsg_push_handler
xvt_errmsg_sig_if
xvt_errmsg_sig_std
xvt_errmsg_sig_std_if
```

The "Diagnostics and Debugging" chapter in the *XVT Portability Toolkit Guide*
For a list of predefined error categories and messages, see the **xvt_perr.h** file

xvt_errmsg_sig_if

Conditionally Signal an Error

Summary

```
void xvt_errmsg_sig_if(BOOLEAN cond, WINDOW object,
     XVT_ERRSEV severity, XVT_ERRID category,
     const char* suffix, unsigned short number,
     const char* message)
```

`BOOLEAN cond`

Conditional statement (`TRUE` evaluation triggers error signal).

`WINDOW object`

`NULL_WIN` or window handle to object in error.

`XVT_ERRSEV severity`

Error severity.

`XVT_ERRID category`

One of the predefined error categories.

`const char* suffix`

Unique error ID constant suffix literal (for **errscan** only).

`unsigned short number`

Unique error message number within specified category.

`const char* message`

Message prototype literal (for **errscan** only).

Description

This macro conditionally signals an abnormal condition to the error message handling facility, which in turn calls one or more error handler(s). Unless caught by application-provided error handlers, the error signal reaches the XVT-provided "last chance" error handler, which posts a dialog corresponding to the error message severity.

The macro strips the `suffix` and `message` arguments. They are used only by the **errscan** tool to generate message file entry, and to generate `#define` constants identifying this message. The constant is formed by adding the `suffix` to the category name. The `suffix and corresponding message number` must be unique within a given message category.

Return Value

None.

See Also

`ATTR_ERRMSG_HANDLER`
`WINDOW`
`XVT_ERRID`
`XVT_ERRSEV`
`xvt_errmsg_pop_handler`
`xvt_errmsg_push_handler`
`xvt_errmsg_sig`
`xvt_errmsg_sig_std`
`xvt_errmsg_sig_std_if`

The "Diagnostics and Debugging" chapter in the *XVT Portability Toolkit Guide*
For a list of predefined error categories and messages, see the **xvt_perr.h** file

xvt_errmsg_sig_std

Signal an Error with a Predefined Error Message

Summary

```
void xvt_errmsg_sig_std(WINDOW object,
     XVT_ERRSEV severity, XVT_ERRID msg_id)
```

`WINDOW object`

`NULL_WIN` or window handle to object in error.

`XVT_ERRSEV severity`

Error severity.

`XVT_ERRID msg_id`

One of the predefined message identifiers.

Description

This macro signals an abnormal condition to the error message handling facility, which in turn calls one or more error handler(s). Unless caught by application-provided error handlers, the error signal reaches the XVT-provided "last chance" error handler, which posts a dialog corresponding to the error message severity.

This macro differs from `xvt_errmsg_sig` in that `msg_id` must be one of the predefined error messages. For list of predefined error categories and messages, see the **xvt_perr.h** file.

Return Value

None.

See Also

```
ATTR_ERRMSG_HANDLER
WINDOW
XVT_ERRID
XVT_ERRSEV
xvt_errmsg_def_*
xvt_errmsg_pop_handler
xvt_errmsg_push_handler
xvt_errmsg_sig
xvt_errmsg_sig_if
xvt_errmsg_sig_std_if
```

The "Diagnostics and Debugging" chapter in the *XVT Portability Toolkit Guide*

xvt_errmsg_sig_std_if

Conditionally Signal an Error with a Predefined Error Message

Summary

```
void xvt_errmsg_sig_std_if(BOOLEAN cond, WINDOW object,
     XVT_ERRSEV severity, XVT_ERRID msg_id)
```

`BOOLEAN cond`

Conditional statement (`TRUE` evaluation triggers error signal).

`WINDOW object`

`NULL_WIN` or window handle to object in error.

`XVT_ERRSEV severity`

Error severity.

`XVT_ERRID msg_id`

One of the predefined message identifiers.

Description

This macro conditionally signals an abnormal condition to the error message handling facility, which in turn calls one or more error handler(s). Unless caught by application-provided error handlers, the error signal reaches the XVT-provided "last chance" error handler, which posts a dialog corresponding to the error message severity.

Unlike `xvt_errmsg_sig_if`, this macro must use one of the predefined error messages.

Return Value

None.

See Also

`ATTR_ERRMSG_HANDLER`
`WINDOW`
`XVT_ERRID`
`XVT_ERRSEV`
`xvt_errmsg_pop_handler`
`xvt_errmsg_push_handler`
`xvt_errmsg_sig`
`xvt_errmsg_sig_if`
`xvt_errmsg_sig_std`

The "Diagnostics and Debugging" chapter in the *XVT Portability Toolkit Guide*
For list of predefined error messages, see the **xvt_perr.h** file

xvt_event_*

Event Access Functions

```
xvt_event_get_font
xvt_event_is_virtual_key
xvt_event_set_font
```

xvt_event_get_font

Get XVT_FNTID Contained In E_FONT Event

Summary

```
XVT_FNTID xvt_event_get_font(EVENT *ep)
```

`EVENT *ep`

`E_FONT` event.

Description

This function returns an `XVT_FNTID` contained in the `E_FONT` event. This function always returns the XVT encapsulated-font-model `XVT_FNTID`--as long as the event is the most recent one that has been dispatched and received by a window event handler. This function is primarily for use in programs that might use both the encapsulated and exposed font models simultaneously.

This function provides a "safe" way for event handlers to obtain the logical font information from an `E_FONT` event without having to determine whether the event is based on the exposed font model or the encapsulated font model.

Parameter Validity and Error Conditions

If the event is not an `E_FONT` event, or if it is not the "current" event, XVT issues an error.

Return Value

`XVT_FNTID`

See Also

`E_FONT`
`XVT_FNTID`
`xvt_event_set_font`
`xvt_win_dispatch_event`

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

xvt_event_is_virtual_key

Check Virtual Key

Summary

```
BOOLEAN xvt_event_is_virtual_key(EVENT *ep)

EVENT *ep
```

`E_CHAR` event.

Description

This function checks if the `ch` field of the `E_CHAR` event represents a virtual key.

Return Value

`TRUE` if the `ch` field represents a virtual key; `FALSE` if not.

This function behaves the same in both single-byte and multibyte aware (`ATTR_MULTIBYTE_AWARE` is `TRUE`) modes.

See Also

`ATTR_MULTIBYTE_AWARE`
`E_CHAR`
`XVT_MOD_KEY`

The "Diagnostics and Debugging" chapter in the *XVT Portability Toolkit Guide*

xvt_event_set_font

Set Logical Font Data in the E_FONT Structure

Summary

```
void xvt_event_set_font(EVENT *ep, XVT_FNTID
font_id)EVENT *ep
```

`E_FONT event.`

`XVT_FNTID font_id`

Handle of logical font.

Description

This function takes an encapsulated-font-model `XVT_FNTID` and sets the logical font data in the `E_FONT` event structure. Even if the application is using the exposed font model, this function correctly places the passed-in `font_id` into the event--as long as the event is the most recent one that has been dispatched and received by a window event handler. This function is primarily for use in programs that might use both the encapsulated and exposed font models simultaneously.

This function provides a "safe" way for event handlers to change the logical font information inside an `E_FONT` event without having to determine whether the event is based on the exposed font model or the encapsulated font model.

Parameter Validity and Error Conditions

XVT issues an error under the following conditions:

- The event is not an `E_FONT` event
- The event is not the "current" event
- The `font_id` is invalid

See Also

`E_FOCUS`
`xvt_win_dispatch_event`

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

xvt_fmap_*

Font Mapper Functions

```
xvt_fmap_get_families
xvt_fmap_get_family_sizes
xvt_fmap_get_family_styles
xvt_fmap_get_familysize_styles
xvt_fmap_get_familystyle_sizes
```

xvt_fmap_get_families

List All Logical Font Families

Summary

```
long xvt_fmap_get_families(PRINT_RCD *precp,
     char **family_array, long max_families)
```

`PRINT_RCD *precp`

Print record or `NULL`.

`char **family_array`

Array of family names filled in by this function. The calling function must pre-allocate this array.

`long max_families`

Maximum number of logical font families to return.

Description

This function lists all logical font families supported by the URL and default XVT font mappers. It does not reflect any families supported by application-supplied font mappers.

If the print record is `NULL`, the inquiry is for the screen window; if it is non-`NULL`, the inquiry is for the specified printer.

The filled-in `char**` array contains the ASCII names of all supported logical font families. The calling function must previously have allocated this array, but `xvt_fmap_get_families` will allocate the individual family names.

The `max_families` parameter indicates the maximum number of logical font family names to put into the `family_array`.

Return Value

Number of families actually placed into `family_array`. When this array of strings is no longer needed, the application is responsible for freeing the strings with `xvt_mem_free`.

Parameter Validity and Error Conditions

XVT issues an error if any of the following are true:

- family_array is `NULL`
- `max_families` is less than 1
- `precp` is non-`NULL` *but* does not point to a valid `PRINT_RCD`

See Also

`PRINT_RCD`
`xvt_fmap_get_family_sizes`
`xvt_fmap_get_family_styles`
`xvt_fmap_get_familysize_styles`
`xvt_fmap_get_familystyle_sizes`

Example

This code adds available logical font families to a list box:

```
int f;
WINDOW listbox;
long num_families;
char *families[MAX_FAMILIES];
...
/* show available families in a listbox */
num_families = xvt_fmap_get_families(NULL, families,
     MAX_FAMILIES);
 xvt_list_suspend(listbox);
xvt_list_clear(listbox);
for (f = 0; f < num_families; f++)
{
     xvt_list_add(listbox, f, families[f]);
    xvt_mem_free(families[f]);
 }
xvt_list_resume(listbox);
```

xvt_fmap_get_family_sizes

List Available Sizes for a Logical Font Family

Summary

```
long xvt_fmap_get_family_sizes(PRINT_RCD *precp,
     char *family, long *size_array, BOOLEAN *scalable,
     long max_sizes)
```

`PRINT_RCD *precp`

Print record or `NULL`.

`char *family`

Logical font family name.

`long *size_array`

Array of sizes filled in by this function. The calling function must pre-allocate this array.

`BOOLEAN *scalable`

Set to `TRUE` on output if a scalable font (such as TrueType) is available.

`long max_sizes`

Maximum number of sizes to return in `size_array`.

Description

Given a logical font family, this function lists all available sizes supported by the URL and default XVT font mappers. It does not reflect any sizes supported by application-supplied font mappers.

If the print record is `NULL`, the inquiry is for the screen window; if it is non-`NULL`, the inquiry is for the specified printer.

The filled-in array of `long`s contains the list of all supported logical font sizes for the specified family. The `max_sizes` parameter indicates the maximum number of logical font sizes to put into `size_array`. Your application must allocate `size_array` and specify the maximum number of elements in `size_array` with `max_sizes`.

If the `BOOLEAN Scalable` parameter is returned as `TRUE`, the font is continuously scalable in size. In this instance, the return value of the function and the contents of `size_array` are not relevant.

Return Value

Number of logical font sizes actually placed into `size_array`.

Parameter Validity and Error Conditions

XVT issues an error if any of the following are true:

- `family` is `NULL`
- `size_array` is `NULL`
- `scalable` is `NULL`
- `max_sizes` is less than 1
- `precp` is non-`NULL` *but* does not point to a valid `PRINT_RCD`

See Also

```
PRINT_RCD
xvt_fmap_get_families
xvt_fmap_get_family_styles
xvt_fmap_get_familysize_styles
xvt_fmap_get_familystyle_sizes
```

Example

```
long sizes[MAX_SIZES];
long num_sizes;
BOOLEAN scalable;
WINDOW listbox;
char buffer[10];
long s;
...
/* add avaliable sizes to a listbox */
num_sizes = xvt_fmap_get_family_sizes(NULL, family,
     sizes, &scalable, MAX_SIZES);
 xvt_list_suspend(listbox);
xvt_list_clear(listbox);
if (scalable == TRUE)
     for (s = 1; s = 128; s++) {
         sprintf(buffer, "%d", s);
        xvt_list_add(listbox, -1, buffer);
     }
 else
     for (s = 0; s < num_sizes; s++) {
         sprintf(buffer, "%d", sizes[s]);
         xvt_list_add(listbox, -1, buffer);
     }
 xvt_list_resume(listbox);
```

xvt_fmap_get_family_styles

List Available Styles for a Logical Font Family

Summary

```
long xvt_fmap_get_family_styles
     (PRINT_RCD *precp, char *family,
     XVT_FONT_STYLE_MASK *style_array, long max_styles)
```

`PRINT_RCD *precp`

Print record or `NULL`.

`char *family`

Logical font family name.

`XVT_FONT_STYLE_MASK *style_array`

Array of styles filled in by this function. The calling function must pre-allocate this array.

`long max_styles`

Maximum number of styles to return in `style_array`.

Description

Given a logical family, this function lists all available styles supported by the URL and default XVT font mappers. It does not reflect any styles supported by application-supplied font mappers.

If the print record is `NULL`, the inquiry is for the screen window; if it is non-`NULL`, the inquiry is for the specified printer.

The filled-in `XVT_FONT_STYLE_MASK` array represents all supported logical font styles for the specified family. Your application must previously have allocated this array and is responsible for freeing it.

The `max_styles` parameter indicates the maximum number of logical font styles to put into the `style_array`.

Return Value

Number of logical font styles actually placed into `style_array`.

Parameter Validity and Error Conditions

XVT issues an error if any of the following are true:

- `family` or `style_array` is `NULL`
- `max_styles` is less than 1

- `precp` is non-`NULL` *but* does not point to a valid `PRINT_RCD`

See Also

`PRINT_RCD`
`XVT_FONT_STYLE_MASK`
`xvt_fmap_get_families`
`xvt_fmap_get_family_sizes`
`xvt_fmap_get_familysize_styles`
`xvt_fmap_get_familystyle_sizes`

xvt_fmap_get_familysize_styles

List Available Styles for a Logical Font Family and Size

Summary

```
long xvt_fmap_get_familysize_styles
     (PRINT_RCD *precp, char *family, long size,
     XVT_FONT_STYLE_MASK *style_array, long max_styles)
```

`PRINT_RCD *precp`

Print record or `NULL`.

`char *family`

Logical font family name.

`long size`

Logical font size.

`XVT_FONT_STYLE_MASK *style_array`

Array of styles filled in by this function. The calling function must pre-allocate this array.

`long max_styles`

Maximum number of styles to return in `style_array`.

Description

Given a logical family and size, this function lists all available logical font styles supported by the URL and default XVT font mappers. It does not reflect any styles supported by application-supplied font mappers.

If the print record is `NULL`, the inquiry is for the screen window; if it is non-`NULL`, the inquiry is for the specified printer.

The filled-in XVT_FONT_STYLE_MASK array represents all supported logical font styles for the specified family and size. Your application must have previously allocated the array and is responsible for freeing it.

The max_styles parameter indicates the maximum number of logical font styles to put into style_array.

Return Value

Number of logical font styles actually placed into style_array.

Parameter Validity and Error Conditions

XVT issues an error if any of the following are true:

- family or style_array is NULL
- size or max_styles is less than 1
- precp is non-NULL *but* does not point to a valid PRINT_RCD

See Also

PRINT_RCD
XVT_FONT_STYLE_MASK
xvt_fmap_get_families
xvt_fmap_get_family_sizes
xvt_fmap_get_familysize_styles
xvt_fmap_get_family_styles

xvt_fmap_get_familystyle_sizes

List Available Sizes for a Logical Font Family and Style

Summary

```
long xvt_fmap_get_familystyle_sizes(PRINT_RCD *precp,
    char *family, XVT_FONT_STYLE_MASK style,
    long *size_array, BOOLEAN *scalable, long max_sizes)
```

PRINT_RCD *precp

Print record or NULL.

char *family

Logical Font family name.

XVT_FONT_STYLE_MASK style

Logical font style.

`long *size_array`

Array of sizes filled in by this function. The calling function must pre-allocate this array.

`BOOLEAN *scalable`

Set to `TRUE` on output if a scalable font (such as TrueType) is available.

`long max_sizes`

Maximum number of sizes to return.

Description

Given a logical family and style, this function lists all available sizes supported by the URL and default XVT font mappers. It does not reflect any sizes supported by application-supplied font mappers.

If the print record is `NULL`, the inquiry is for the screen window; if it is non-`NULL`, the inquiry is for the specified printer.

The filled-in array of `long`s contains the list of all supported logical font sizes for the specified family and style. The `max_sizes` parameter indicates the maximum number of sizes to put into the `size_array`. Your application must allocate `size_array` and specify the maximum number of elements in `size_array` with `max_sizes`.

If the `scalable` parameter is returned as `TRUE`, the return value of the function is 2, indicating that `size_array` has two entries. The two `size_array` entries show the range of the scalable font: from the smallest point size available (the entry in element zero) to the largest (the entry in element one).

Return Value

Number of logical font sizes actually placed into `size_array`.

Parameter Validity and Error Conditions

XVT issues an error if any of the following are true:

- `family` is `NULL`
- `size_array` is `NULL`
- `scalable` is `NULL`
- `max_sizes` is less than 1
- `precp` is non-`NULL` *but* does not point to a valid `PRINT_RCD`

See Also

PRINT_RCD
XVT_FONT_STYLE_MASK
xvt_fmap_get_families
xvt_fmap_get_family_sizes
xvt_fmap_get_familysize_styles
xvt_fmap_get_family_styles

xvt_font_*

Font Functions

xvt_font_copy
xvt_font_create
xvt_font_deserialize
xvt_font_destroy
xvt_font_get_app_data
xvt_font_get_family
xvt_font_get_family_mapped
xvt_font_get_metrics
xvt_font_get_native_desc
xvt_font_get_size
xvt_font_get_size_mapped
xvt_font_get_style
xvt_font_get_style_mapped
xvt_font_get_win
xvt_font_has_valid_native_desc
xvt_font_is_mapped
xvt_font_is_print
xvt_font_is_scalable
xvt_font_is_valid
xvt_font_map
xvt_font_map_using_default
xvt_font_serialize
xvt_font_set_app_data
xvt_font_set_family
xvt_font_set_native_desc
xvt_font_set_size
xvt_font_set_style
xvt_font_unmap

xvt_font_copy

Copy a Logical Font

Summary

```
void xvt_font_copy(XVT_FNTID dest_font_id,
     XVT_FNTID src_font_id, XVT_FONT_ATTR_MASK mask)
```

`XVT_FNTID dest_font_id`

Destination logical font handle.

`XVT_FNTID src_font_id`

Source logical font handle.

`XVT_FONT_ATTR_MASK mask`

Mask of the `XVT_FA_*` flag values.

Description

This function copies logical font values from the logical font specified by `src_font_id` to the logical font specified by `dest_font_id`, based on the font attribute mask.

Only the portions of the logical font specified by the mask are copied. A mask of `XVT_FA_ALL` copies all attributes.

This function does not create or allocate a new logical font. Both the source and destination must be valid logical fonts. Note that if `XVT_FA_APP_DATA` is specified in the mask, only the *pointer* to the application data is copied, not the application data itself.

Parameter Validity and Error Conditions

If either `scr_font_id` or `dest_font_id` is invalid, XVT issues an error.

See Also

```
XVT_FA_* Constants
XVT_FNTID
XVT_FONT_ATTR_MASK
xvt_font_create
xvt_font_destroy
xvt_menu_set_font_sel
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

```
DOCUMENT *doc = (DOCUMENT*) xvt_vobj_get_data(win);
switch (ep->type){
case E_FONT:
     xvt_font_copy(doc->font_id, ep->v.font.font_id,
XVT_FA_ALL);
    xvt_menu_set_font(win, doc->font_id);
    break;
 case E_CREATE:
     ...
    doc->font_id = xvt_font_create();
    ...
    break;
 case E_DESTROY:
     xvt_font_destroy(doc->_id);
    break;
 }
```

xvt_font_create

Create a Logical Font

Summary

```
XVT_FNTID xvt_font_create(void)
```

Description

This function creates a logical font by allocating an internal `XVT_FNTID` and giving it a set of standard default values: family is `XVT_FFN_SYSTEM` ("system"), style is `XVT_FS_NONE`, size is 12 points, `app_data` is `NULL`, native descriptor is `NULL`, and window is `NULL_WIN`. It then returns the `XVT_FNTID` to the calling function.

If the application needs to change the logical font by giving it non-default values, it can call any of the logical font attribute setting functions. The newly allocated `XVT_FNTID` is not mapped at the time this function is called.

This function, `xvt_ctl_get_font`, `xvt_dwin_get_font`, `xvt_menu_get_font_sel`, `xvt_res_get_font`, and `xvt_win_get_ctl_font`, are the only ones that create a new logical font.

To remove this logical font from the system, you must use `xvt_font_destroy`.

Return Value

A handle to a newly-allocated logical font.

See Also

```
NULL_WIN
XVT_FFN_* Constants
XVT_FNTID
XVT_FS_* Constants
xvt_ctl_get_font
xvt_dwin_get_font
xvt_font_destroy
xvt_font_set_app_data
xvt_font_set_family
xvt_font_set_native_desc
xvt_font_set_size
xvt_font_set_style
xvt_font_get_win
xvt_menu_get_font_sel
xvt_res_get_font
xvt_win_get_ctl_font
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_font_copy`.

xvt_font_deserialize

Deserialize a Previously Serialized Logical Font

Summary

```
BOOLEAN xvt_font_deserialize(XVT_FNTID font_id,
     char *buf)
```

`XVT_FNTID font_id`

Handle of the logical font to deserialize.

`char *buf`

Buffer containing serialized logical font.

Description

This function sets a logical font to correspond to the one previously serialized by `xvt_font_serialize`. Your application must have created a logical font identified by `font_id` before passing it as a

parameter to this function. The `font_id` is filled with the logical font attributes stored in `buf`.

`xvt_font_deserailize` is useful for reading a previously serialized and archived logical font from a file.

Return Value

`TRUE` if deserialization is successful; `FALSE` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if any of the following are true:

- `font_id` is invalid
- `buf` is `NULL`
- `buf` does not contain a serialized logical font

See Also

```
XVT_FNTID
xvt_font_create
xvt_font_serialize
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

```
#define BUF_SIZE 256
void read_font_from_file(FILE *fp, XVT_FNTID font_id)
{
     EOL_FORMAT format;
    long length;
    char buffer[BUF_SIZE];
    fgets(buffer, BUF_SIZE, fp);
    xvt_str_find_eol(buffer,
xvt_str_get_byte_count(buffer),
         &length, &format);
     if (format != EOL_NONE)
         buffer[length] = '0';
     if (!xvt_font_deserialize(font_id, buffer))
         xvt_dm_post_warning(
             "Could not deserialize font");
 }
```

xvt_font_destroy

Destroy a Logical Font

Summary

```
void xvt_font_destroy(XVT_FNTID font_id)
```

`XVT_FNTID font_id`

Handle of the logical font to be destroyed.

Description

This function destroys the logical font identified by `font_id`. If this logical font is mapped at the time it is destroyed, the physical font to which it is mapped is released back to the window system on which the application is running.

This function destroys a logical font, but does not destroy application data. You must create, manage, and destroy your own application data.

Parameter Validity and Error Conditions

If the `font_id` is not `NULL_FNTID` and invalid, XVT issues an error.

See Also

`NULL_FNTID`
`NULL_WIN`
`XVT_FNTID`
`xvt_dwin_get_font`
`xvt_font_copy`
`xvt_font_create`

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

See the examples for `xvt_dwin_get_font and xvt_font_copy`.

xvt_font_get_app_data

Get Logical Font Application Data

Summary

```
long xvt_font_get_app_data(XVT_FNTID font_id)
```

`XVT_FNTID font_id`

Handle of the logical font whose application data is retrieved.

Description

This function gets the application data for a logical font and returns it as a `long` value. The default value for application data is `NULL`.

Any logical font can have a `long` data word associated with it for your application's use. This function retrieves that data.

Frequently the application data is a pointer to a structure of your own design. In this case, your application should cast the return value from `xvt_font_get_app_data` into a pointer of the correct type.

Return Value

`long` integer for the application data associated with the `XVT_FNTID`.

Parameter Validity and Error Conditions

If `font_id` is not a valid `XVT_FNTID`, XVT issues an error.

See Also

`XVT_FNTID`
`xvt_font_set_app_data`
`xvt_dwin_get_font_app_data`

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

xvt_font_get_family

Get Logical Font Family

Summary

```
BOOLEAN xvt_font_get_family(XVT_FNTID font_id,
     char *buf, long max_buf)
```

`XVT_FNTID font_id`

Handle of the logical font for which family is retrieved.

`char *buf`

Buffer into which the logical font family is placed.

`long max_buf`

Available buffer size in bytes.

Description

This function gets the "desired" (as opposed to "mapped") family value from the logical font identified by `font_id`. The returned logical font family string must be pre-allocated and owned by the application.

Return Value

`TRUE` if successful; `FALSE` if an error is detected.

Parameter Validity and Error Conditions

XVT issues an error if any of the following are true:

- `font_id` is invalid
- `buf` is `NULL`
- `max_buf` is less than 1
- returned family name requires more bytes than `max_buf`

If an error occurs,the contents of the returned string should be considered invalid.

See Also

`XVT_FNTID`
`xvt_dwin_get_font_family`
`xvt_font_get_family_mapped`
`xvt_font_set_family`

The "Font Definitions" section of the "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

xvt_font_get_family_mapped

Get Mapped Logical Font Family

Summary

```
BOOLEAN xvt_font_get_family_mapped(XVT_FNTID font_id,
     char *buf, long max_buf)
```

`XVT_FNTID font_id`

Handle of the logical font for which family is retrieved.

`char *buf`

Buffer into which the mapped logical font family is placed.

`long max_buf`

Available buffer size in bytes.

Description

This function gets the mapped logical font family value of the logical font identified by `font_id`. The returned logical font family string must be pre-allocated and owned by the application.

This inquiry function is intended for use in implementing application-supplied font mappers, but you can also use it to obtain information outside of an application font mapper.

Return Value

`TRUE` if successful; `FALSE` if an error is detected.

Parameter Validity and Error Conditions

XVT issues an error if any of the following are true:

- `font_id` is invalid
- `buf` is `NULL`
- `max_buf` is less than 1
- font is not mapped
- returned family name requires more bytes than `max_buf`

If an error occurs,the contents of the returned string should be considered invalid.

See Also

`XVT_FNTID`
`xvt_dwin_get_font_family_mapped`
`xvt_font_get_family`
`xvt_font_map`
`xvt_font_set_family`

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

xvt_font_get_metrics

Get a Logical Font's Leading, Ascent, and Descent

Summary

```
void xvt_font_get_metrics(XVT_FNTID font_id,
     int *leadingp, int *ascentp, int *descentp)
```

`XVT_FNTID font_id`

Handle of the logical font for which metrics are being retrieved.

`int *leadingp`

Pointer to the logical font's leading.

`int *ascentp`

Pointer to the logical font's ascent.

`int *descentp`

Pointer to the logical font's descent.

Description

This function gets three mapped attributes of a logical font: leading, ascent, and descent. These values are returned through the corresponding integer-pointer arguments. If an argument is `NULL`, that particular metric isn't returned.

`xvt_font_get_metrics` requires that font_id already be associated with a valid window (i.e., `xvt_font_get_win` does not return `NULL_WIN`). This is the case if the logical font was retrieved from a window (`xvt_dwin_get_font`), or if the logical font has already been mapped (`xvt_font_map`). If the logical font is not mapped

when you call this function, and if `xvt_font_get_win` returns a valid window, this function automatically maps it to that window.

For normally spaced text, you should use a line spacing equal to the sum of the leading, ascent, and descent values.

Parameter Validity and Error Conditions

If `font_id` is invalid, or if the window associated with the logical font is `NULL_FNTID` is invalid, XVT issues an error.

See Also

```
NULL_FNTID
XVT_FNTID
xvt_dwin_get_font
xvt_dwin_get_font_metrics
xvt_font_get_win
xvt_font_map
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

This code gets the logical font from a window, inquires its metrics, and then draws text using the leading and ascent to specify the text baseline:

```
static void do_update(WINDOW win)
{
     int leading, ascent, descent;
    XVT_FNTID font_id = xvt_dwin_get_font(win);
    xvt_dwin_clear(win, COLOR_WHITE);
    xvt_font_get_metrics(font_id, &leading, &ascent,
         &descent);
     xvt_font_destroy(font_id);
    xvt_dwin_draw_text(win, 4, leading + ascent,
         "Hello World!", -1);
     xvt_dwin_draw_icon(win, 10, 40, ICON_RID);
 }
```

xvt_font_get_native_desc

Get Native Font Descriptor

Summary

```
BOOLEAN xvt_font_get_native_desc(XVT_FNTID font_id,
     char *buf, long max_buf)

XVT_FNTID font_id
```

Handle of the logical font for which the native descriptor is being returned.

`char *buf`

Buffer into which the native descriptor is copied.

`long max_buf`

Available buffer size in bytes.

Description

This function gets the native font descriptor data from the logical font identified by `font_id`. If the native descriptor has been set, it will be returned in `buf`. Otherwise, `buf` will be empty.

Note: The native descriptor can be set by a call to `xvt_font_set_native_desc`, `xvt_font_map`, `xvt_font_map_using_default`, by any function that causes implicit font mapping to occur, or by the font dialog. The native descriptor is `NULL` by default.

Return Value

`TRUE` if successful; `FALSE` if an error is detected.

Parameter Validity and Error Conditions

XVT issues an error if any of the following are true:

- `font_id` is invalid
- `buf` is `NULL`
- if the native descriptor does fit into `max_buf` bytes
- `max_buf` is less than 1

See Also

```
XVT_FNTID
xvt_dwin_get_font_native_desc
xvt_font_map
xvt_font_map_using_default
xvt_font_set_native_desc
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

xvt_font_get_size

Get Logical Font Size

Summary

```
long xvt_font_get_size(XVT_FNTID font_id)
```

`XVT_FNTID font_id`

Handle of the logical font for which size is retrieved.

Description

This function gets the "desired" (as opposed to "mapped") size value of the logical font identified by `font_id`. The default size is 12 points.

Return Value

The logical font size in points, if successful; zero if an error is detected.

Parameter Validity and Error Conditions

If `font_id` is invalid, XVT issues an error.

See Also

```
XVT_FNTID
xvt_dwin_get_font_size
xvt_font_get_size_mapped
xvt_font_set_size
```

The "Font Definitions" section of the "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*''''

xvt_font_get_size_mapped

Get Mapped Logical Font Size

Summary

```
long xvt_font_get_size_mapped(XVT_FNTID font_id)
```

`XVT_FNTID font_id`

Handle of the logical font for which size is retrieved.

Description

This function gets the mapped size value of the logical font identified by `font_id`.

This inquiry function is intended for use in implementing application-supplied font mappers, but you can also use it to obtain information outside of a font mapper.

Return Value

The mapped logical font size in points if successful; zero if an error is detected.

Parameter Validity and Error Conditions

If `font_id` is invalid, or if the logical font is not mapped, XVT issues an error.

See Also

```
XVT_FNTID
xvt_dwin_get_font_size_mapped
xvt_font_get_size
xvt_font_map
xvt_font_map_using_default
xvt_font_set_size
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

xvt_font_get_style

Get Logical Font Style

Summary

```
XVT_FONT_STYLE_MASK xvt_font_get_style
     (XVT_FNTID font_id)

XVT_FNTID font_id
```

Handle of the logical font for which styles are being inquired.

Description

This function gets the "desired" (as opposed to "mapped") style value of the logical font identified by `font_id`. The default font style is `XVT_FS_NONE`.

Return Value

The desired logical font style if successful; `XVT_FS_NONE` if an error is detected.

Parameter Validity and Error Conditions

If `font_id` is invalid, XVT issues an error.

See Also

```
XVT_FNTID
XVT_FONT_STYLE_MASK
XVT_FS_* Constants
xvt_dwin_get_font_style
xvt_font_get_style_mapped
xvt_font_set_style
```

The "Font Definitions" section of the "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

xvt_font_get_style_mapped

Get Mapped Logical Font Style

Summary

```
XVT_FONT_STYLE_MASK xvt_font_get_style_mapped
     (XVT_FNTID font_id)
```

`XVT_FNTID font_id`

Handle of the logical font for which mapped styles are being retrieved.

Description

This function gets the mapped style value of the logical font identified by `font_id`. As an inquiry function, it is intended for use in implementing application-supplied font mappers, but you can also use it to obtain information outside of a font mapper.

Return Value

The mapped logical font style if successful; `XVT_FS_NONE` if unsuccessful.

Parameter Validity and Error Conditions

If `font_id` is invalid or if the logical font is not mapped, XVT issues an error.

See Also

```
XVT_FNTID
XVT_FONT_STYLE_MASK
XVT_FS_* Constants
xvt_dwin_get_font_style_mapped
xvt_font_get_style
xvt_font_map
xvt_font_map_using_default
xvt_font_set_style
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

xvt_font_get_win

Get Window Associated With a Logical Font

Summary

```
WINDOW xvt_font_get_win(XVT_FNTID font_id)

XVT_FNTID font_id
```

Handle of the logical font whose window is being retrieved.

Description

This function gets the window associated with a logical font. The font mapping controller associates a window with a logical font during mapping. The default window value for a logical font is `NULL_WIN`.

Return Value

The associated window if successful; `NULL_WIN` on error.

Parameter Validity and Error Conditions

If `font_id` is invalid, XVT issues an error.

See Also

```
XVT_FNTID
WINDOW
xvt_font_create
xvt_font_map
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

```
font_id = xvt_font_create();
xvt_font_map(font_id, window);
xvt_font_get_metrics(font_id, ...);
...
/* later, when you have forgotten the window of the
     font */
    win1 = xvt_font_get_win(font_id);
```

xvt_font_has_valid_native_desc

Determine if Native Font Descriptor Is Valid

Summary

```
BOOLEAN xvt_font_has_valid_native_desc
     (XVT_FNTID font_id)
```

`XVT_FNTID font_id`

Handle of the logical font whose native descriptor is being verified.

Description

This function determines if the native descriptor in the `font_id` refers to a valid physical font.

Return Value

`TRUE` if the native descriptor in the `font_id` refers to a valid physical font; `FALSE` otherwise.

Parameter Validity and Error Conditions

XVT issues an error if any of the following are true:

- `font_id` is invalid
- The window associated with the logical font is not a valid drawable window

See Also

```
NULL_FNTID
XVT_FNTID
xvt_font_map
xvt_font_map_using_default
xvt_font_set_native_desc
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_font_map_using_default`.

xvt_font_is_mapped

Determine if a Logical Font is Mapped

Summary

```
BOOLEAN xvt_font_is_mapped(XVT_FNTID font_id)

XVT_FNTID font_id
```

Handle of the logical font whose mapped status is being determined.

Description

This function determines if a logical font is mapped. By default, a logical font is "not mapped."

Return Value

`TRUE` if the logical font is mapped; `FALSE` if an error is detected.

Parameter Validity and Error Conditions

If `font_id` is invalid, XVT issues an error.

See Also

```
XVT_FNTID
xvt_font_map
xvt_font_map_using_default
xvt_font_unmap
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

xvt_font_is_print

Determine if a Logical Font is Mapped to a Print Window

Summary

```
BOOLEAN xvt_font_is_print(XVT_FNTID font_id)

XVT_FNTID font_id
```

Handle of the logical font whose "printer mapped" status is being determined.

Description

This function determines if a logical font is mapped to a print window.

Return Value

`TRUE` if the logical font is mapped *and* the mapping is for a print window font; `FALSE` otherwise.

Parameter Validity and Error Conditions

If `font_id` is invalid, or if the logical font is not mapped, XVT issues an error and returns `FALSE`.

See Also

```
XVT_FNTID
xvt_font_get_win
xvt_font_map
xvt_font_map_using_default
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

---

xvt_font_is_scalable

Determine if a Mapped Logical Font can be Scaled

Summary

```
BOOLEAN xvt_font_is_scalable(XVT_FNTID font_id)

XVT_FNTID font_id
```

Handle of the logical font whose "scalable" status is being determined.

Description

This function determines if a mapped logical font can be scaled to a continuous arbitrary size.

Return Value

`TRUE` if the logical font is mapped *and* the mapped logical font is scalable; `FALSE` otherwise.

Parameter Validity and Error Conditions

If `font_id` is invalid, or if the logical font is not mapped, XVT issues an error.

See Also

`XVT_FNTID`
`xvt_font_map`

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

xvt_font_is_valid

Determine if Font ID is Defined

Summary

```
BOOLEAN xvt_font_is_valid(XVT_FNTID font_id)
```

`XVT_FNTID font_id`

Handle of the logical font whose validity is being determined.

Description

This function determines if a `font_id` has been defined (i.e., if a logical font is valid). A valid logical font is one that has been created and not yet destroyed.

Return Value

`TRUE` if the `font_id` is valid; `FALSE` otherwise.

Parameter Validity and Error Conditions

None.

See Also

`XVT_FNTID`
`xvt_font_create`

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

xvt_font_map

Map a Logical Font in the Context of a Window

Summary

```
void xvt_font_map(XVT_FNTID font_id, WINDOW win)
```

`XVT_FNTID font_id`

Handle of the logical font to be mapped.

`WINDOW win`

Window to map against.

Description

This function forces a logical font to be mapped in the context of the specified window. It invokes the same mapping that would be done automatically if you used an unmapped logical font in a call to `xvt_dwin_set_font` followed by `xvt_dwin_draw_text`.

The function attempts to map the logical font with the following methods, in order: (1) honor the logical font's native descriptor if it is set and valid in the context of the window parameter, (2) use the application-supplied font mapper, (3) use the application-supplied URL mapping extensions, and (4) use the default XVT font mapper.

You can use a logical font in this function immediately after it is created (with `xvt_font_create`), but typically the application sets at least the family, style, and size first. As a result, the mapping process consists of setting the logical font's window and native font descriptor (if not already set), and marking the logical font as "mapped." Once this has occurred, the logical font is considered to be mapped.

If no errors are detected, `xvt_font_map` *always* succeeds in mapping. However, the physical font to which the logical font is mapped might not necessarily closely match the portable logical font attributes. This would occur only if no physical font was a good match for the portable logical font attributes.

Tip: This function is very useful if you need to set a single logical font in several windows simultaneously. By allowing the logical font to be mapped "up front," the system can avoid having to map several times when text is drawn in the windows.

It is also useful to call this function in an application if you need to create a logical font and then immediately inquire about mapped font attributes before the logical font is actually assigned to a window with `xvt_dwin_set_font`. If the logical font is already mapped, this function checks its validity and returns if valid, or remaps if invalid.

Parameter Validity and Error Conditions

If `font_id` is invalid, or if a window parameter is not a valid, drawable window, XVT issues an error. Print windows and `XVT_PIXMAP`s are valid values for `win`.

See Also

`ATTR_FONT_MAPPER`
`XVT_FNTID`
`WINDOW`
`XVT_PIXMAP`
`xvt_dwin_draw_text`
`xvt_dwin_get_font`
`xvt_dwin_set_font`
`xvt_font_create`
`xvt_font_get_win`
`xvt_font_map_using_default`
`font` URL statement
`font_map` URL statement

The "Font Mapping and the Font Mapping Controller" section of the "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_font_get_win`.

---

xvt_font_map_using_default

Invoke XVT Default Font Mapper

Summary

```
void xvt_font_map_using_default(XVT_FNTID font_id)
```

`XVT_FNTID font_id`

Handle of the logical font to be mapped.

Description

This function invokes the XVT default font mapper only. It does not use any URL-based or application-supplied font mappers. Normally this function is used inside an application-supplied font mapper that has been registered with XVT by means of the `ATTR_FONT_MAPPER` attribute.

The function assumes that the logical font's window attribute is valid. The window is not settable from the application, but it is set implicitly by `xvt_font_map`. If the logical font's native descriptor field is set and is valid in the context of the window associated with the font, this function uses it in the mapping. If that field is not set or is not valid, the function uses the portable logical font attributes to derive a native descriptor with which to map the font.

If no errors are detected, `xvt_font_map_using_default` *always* succeeds in mapping. However, the resulting physical font is not necessarily a close approximation of the portable logical font attributes.

XVT's font mapping controller invokes this function after it has already tried to map the logical font using the registered application-supplied font mapper and the URL font mapper.

Tip: This function is particularly useful when called from an application-supplied font mapper. Your application-supplied font mapper might iteratively change the portable logical font attributes, call this function, and get the mapped results until the desired mapping is achieved.

Parameter Validity and Error Conditions

If the `font_id` is invalid, or if a window associated with a logical font is invalid, XVT issues an error.

See Also

`ATTR_FONT_MAPPER`
`XVT_FNTID`
`xvt_font_map`
`font` URL statement
`font_map` URL statement

The "Font Mapping and the Font Mapping Controller" section of the "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

```
#define NATIVE_DESC "X1101/b&h/lucidatypewriter/medium/
r/normal/sans/12/120/75/75/m/70/iso8859/1"
...
xvt_vobj_set_attr(NULL_WIN, ATTR_FONT_MAPPER,
     (long) application_font_mapper);
 ...
BOOLEAN XVT_CALLCONV1
application_font_mapper(XVT_FNTID font_id)
{
     /* map font_id to NATIVE_DESC if it is valid */
    BOOLEAN native_desc_ok;
    xvt_font_set_native_desc(font_id, NATIVE_DESC);
    native_desc_ok = xvt_font_has_valid_native_desc(
         font_id);
     if (native_desc_ok)
         xvt_font_map_using_default(font_id);
     return native_desc_ok;
 }
```

xvt_font_serialize

Serialize a Logical Font

Summary

```
long xvt_font_serialize(XVT_FNTID font_id,
     char *buf, long max_buf)
```

`XVT_FNTID font_id`

Handle of the logical font to be serialized.

`char *buf`

Buffer for serialized font to fill. `NULL` is a valid value for this parameter.

`long max_buf`

Maximum buffer size in bytes.

Description

This function serializes a logical font into a stream of bytes in a buffer. The serialized logical font is tagged with the serialization version. The resulting buffer can be written to a file for storage. You can later retrieve it and turn it back into a valid logical font by using `xvt_font_deserialize`.

However, if your application passes a NULL buffer, this function returns as its value the number of bytes in the serialized font (plus 1 for the NULL-termination character), but does not attempt to serialize it. Also, if the buffer is NULL, no error check is performed on the max_buf parameter.

Return Value

Number of bytes used in the non-NULL buffer, or the required size of the buffer (the serialized font plus 1) if the buffer is NULL; zero if an error occurred.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- font_id is invalid
- buf (if non-NULL) is too small to contain the serialized font string (including the NULL-termination character)
- max_buf is less than 1 (this is checked only if buf is non-NULL)

See Also

XVT_FNTID
xvt_font_deserialize

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

This code shows how xvt_font_serialize can be used to first determine the necessary buffer size, and then to serialize the font into the buffer:

```
void save_font_to_file(FILE *fp, XVT_FNTID font_id)
{
     char *buffer;
    long length;
    length = xvt_font_serialize(font_id, NULL, 0);
    buffer = xvt_mem_alloc(length);
    if (buffer)
    {
         if (xvt_font_serialize(font_id, buffer,
                length))
            fputs(buffer, fp);
         else
            xvt_dm_post_warning(
                "Could not serialize font");
         xvt_mem_free(buffer);
     }
    else
            xvt_dm_post_warning(
                "Could not serialize font");
 }
```

xvt_font_set_app_data

Set Application Data for a Logical Font

Summary

```
void xvt_font_set_app_data(XVT_FNTID font_id,
     long app_data)
```

`XVT_FNTID font_id`

Handle of the logical font whose application data is being set.

`long app_data`

Application data.

Description

This function sets the application data for a logical font. If used by the application, application data provides a hook so that the application can attach additional information to the logical font for use by application-supplied font mappers.

If the logical font was previously mapped, this function unmaps it. `app_data` is long instead of `void*` to support non-ANSI C compilers.

Parameter Validity and Error Conditions

If `font_id` is invalid, XVT issues an error.

See Also

`XVT_FNTID`
`xvt_dwin_set_font_app_data`
`xvt_font_get_app_data`

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

xvt_font_set_family

Set Logical Font Family

Summary

```
void xvt_font_set_family(XVT_FNTID font_id,
     char *family)
```

`XVT_FNTID font_id`

Handle of the logical font for which family is being set.

`char *family`

String containing family name.

Description

This function sets the family value in the `font_id` to the logical font family specified by `family`. The `family` is copied into the underlying `XVT_FNTID`. If the logical font was previously mapped and the family changes, this function unmaps it.

The `family` parameter can be any string. The font mapper guarantees that the family *will* be mapped, regardless of what the family name is. However, if the family is the name of a logical font that does not exist as a physical font, is not handled by an application-supplied font mapper, or is not handled by URL extensions to the font mapper, the mapping result may not be a close fit.

At the time that `xvt_font_set_family` is called, there is no check to determine if the family is a "valid" family. The only check is to make sure that the family is a valid string. The reasons for this are three-fold:

- It would be expensive to go through the mapping logic to determine if this family can, indeed, be mapped into a physical font
- It may be that the family does not currently exist in the font mapper, but will exist by the time this logical font is actually mapped
- If the application sets the family several times before drawing text, deferring the validity check on the family until it is actually used provides a performance benefit

Parameter Validity and Error Conditions

If `font_id` is invalid, or if `family` is `NULL`, XVT issues an error.

See Also

```
XVT_FNTID
xvt_dwin_set_font_family
xvt_font_get_family
xvt_font_get_family_mapped
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

```
/* set window font to Helvetica, 24, Bold */
XVT_FNTID font_id;
font_id = xvt_font_create();
xvt_font_set_family(font_id, "helvetica");
xvt_font_set_size(font_id, 24);
xvt_font_set_style(font_id, XVT_FS_BOLD);
xvt_dwin_post_font_sel(window, font_id, NULL, OL);
xvt_font_destroy(font_id);
```

xvt_font_set_native_desc

Set Native Font Descriptor

Summary

```
void xvt_font_set_native_desc(XVT_FNTID font_id,
     char *native_font_desc)
```

`XVT_FNTID font_id`

Handle of the logical font whose native descriptor is being set.

`char *native_font_desc`

String specification of physical font.

Description

This function sets the native font descriptor in the logical font identified by `font_id`. The application-supplied font mapper should set this attribute to tell the XVT font mapping controller which physical font to use in mapping.

The `native_font_desc` format is the same one that you would use in an URL `font` or `font_map` statement.

Note: This function is primarily intended for use in application-supplied font mappers, although it is legal to call it from anywhere. The XVT default font mapper and the URL font mapper (both internal to the XVT Portability Toolkit) implicitly call `xvt_font_set_native_desc` if they detect that a logical font's native descriptor is either `NULL` or invalid in the context of the logical font's window. If you specifically want to "unset" the native font descriptor, XVT recommends that you use the string ("") as the second parameter.

Parameter Validity and Error Conditions

If `font_id` is invalid, or if `native_font_desc` is `NULL`, XVT issues an error.

See Also

```
ATTR_FONT_MAPPER
XVT_FNTID
xvt_dwin_set_font_native_desc
xvt_font_get_native_desc
xvt_font_map_using_default
font URL statement
font_map URL statement
```

The "Application-Supplied Font Mappers" and "Setting Native Font Descriptors" sections of the "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*
The *XVT Platform-Specific Books*

Example

See the example for `xvt_font_map_using_default`.

xvt_font_set_size

Set Logical Font Size

Summary

```
void xvt_font_set_size(XVT_FNTID font_id, long size)
```

`XVT_FNTID font_id`

Handle of the logical font for which size is being set.

`long size`

Value of new size.

Description

This function sets the logical font size value in the `font_id` to the size specified by `size`. If the logical font was previously mapped and the size changes, this function unmaps it.

Parameter Validity and Error Conditions

If `font_id` is invalid, or if the size is not positive, XVT issues an error.

See Also

```
XVT_FNTID
xvt_dwin_set_font_size
xvt_font_get_size
xvt_font_get_size_mapped
xvt_font_set_family
xvt_font_set_style
```

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_font_set_family`.

xvt_font_set_style

Set Logical Font Style

Summary

```
void xvt_font_set_style(XVT_FNTID font_id,
     XVT_FONT_STYLE_MASK mask)
```

`XVT_FNTID font_id`

Handle of the logical font for which styles are being set.

`XVT_FONT_STYLE_MASK mask`

Font style mask composed of one or more `XVT_FS_*` flag values.

Description

This function sets the logical font style attributes of the `font_id` as specified by `mask`. This function overrides any existing style attributes of the specified logical font. If the logical font was previously mapped and the style changes, this function unmaps it.

Parameter Validity and Error Conditions

If `font_id` is invalid, XVT issues an error.

See Also

`XVT_FNTID`
`XVT_FONT_STYLE_MASK`
`XVT_FS_* Constants`
`xvt_dwin_set_font_style`
`xvt_font_get_style`
`xvt_font_get_style_mapped`
`xvt_font_set_family`
`xvt_font_set_size`

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_font_set_family`.

xvt_font_unmap

Unmap a Logical Font

Summary

```
void xvt_font_unmap(XVT_FNTID font_id)
```

`XVT_FNTID font_id`

Handle of the logical font to be unmapped.

Description

This functions unmaps a logical font. Native resources associated with the logical font may not be released immediately, but they will be freed when the unmapped logical font is removed from the font cache. Unmapping does not affect the XVT portable font attributes associated with the logical font. If the logical font is not mapped when this function is called, the function simply returns.

This function is called internally whenever something occurs to invalidate a font mapping. For example, if the application sets the `family`, `style_mask`, `size`, `application_data`, or `native_description` for a mapped logical font, the system may automatically unmap the logical font.

Parameter Validity and Error Conditions

If `font_id` is invalid, XVT issues an error.

See Also

`XVT_FNTID`
`xvt_font_map`
`xvt_font_map_using_default`

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide.*

xvt_fsys_*

File System Functions

```
xvt_fsys_build_pathname
xvt_fsys_convert_dir_to_str
xvt_fsys_convert_str_to_dir
xvt_fsys_get_default_dir
xvt_fsys_get_dir
xvt_fsys_get_file_attr
xvt_fsys_list_files
xvt_fsys_parse_pathname
xvt_fsys_rem_file
xvt_fsys_restore_dir
xvt_fsys_save_dir
xvt_fsys_set_dir
xvt_fsys_set_dir_startup
xvt_fsys_set_file_attr
```

xvt_fsys_build_pathname

Construct a Native Pathname

Summary

```
BOOLEAN xvt_fsys_build_pathname (char *mbs,
    const char *volname, const char *dirname,
    const char *leafroot, const char *leafext,
    char *leafvers)
```

`char *mbs`

String for full pathname.

`const char *volname`

Volume name.

`const char *dirname`

Directory name.

`const char *leafroot`

File leaf root.

`const char *leafext`

File extension.

`char *leafvers`

File version number.

Description

This function constructs a native pathname in the multibyte string `mbs` from the pathname parts passed in. The appropriate delimiters are added between parts if necessary. It has the reverse effect of `xvt_fsys_parse_pathname`.

Return Value

`TRUE` if the construction of the pathname is successful (`wbs! = NULL`); `FALSE` if no pathname is constructed (`mbs == NULL`).

Parameter Validity and Error Conditions

Severity error "`NULL handle`" if the pointer to the path is `NULL`.

See Also

```
xvt_fsys_convert_dir_to_str
xvt_fsys_convert_str_to_dir
xvt_fsys_parse_pathname
```

The "Files" chapter in the *XVT Portability Toolkit Guide*

xvt_fsys_convert_dir_to_str

Convert Directory to String Form

Summary

```
BOOLEAN xvt_fsys_convert_dir_to_str(DIRECTORY *dirp,
    char *path, int sz_path)
```

`DIRECTORY *dirp`

`DIRECTORY` to be converted to string form.

`char *path`

String into which converted directory is stored.

`int sz_path`

Maximum capacity of `path`.

Description

This function converts a `DIRECTORY` object to a `NULL`-terminated string in the form used by the local system for path names. The

output string is stored into `path`, whose maximum capacity (including `NULL`) is `sz_path`.

The string form of the directory is complete, in that it includes volume or drive designators and a complete path hierarchy from the root of the file system.

Note: The XVT constant `SZ_FNAME` is the maximum filename length in bytes used in XVT, and is platform-specific.

Return Value

`TRUE` if the successful; `FALSE` if unsuccessful (on error).

Parameter Validity and Error Conditions

Severity error "`NULL handle`" if the **`*dirp`** or `*path` is `NULL`.

Implementation Note

On XVT/Mac, the string begins with the volume name and ends with a colon (e.g., `HD:XVT:Examples:`). On UNIX, HPFS, and NTFS implementations it does not end with a slash or back-slash (e.g., c:examples). You must take this into account when concatenating additional path or filename components onto the end of the directory string.

See Also

```
DIRECTORY
SZ_FNAME
xvt_fsys_convert_str_to_dir
xvt_fsys_get_dir
```

The "Files" chapter in the *XVT Portability Toolkit Guide*

Example

In this code, the third argument to `xvt_fsys_convert_dir_to_str` uses `sizeof`, which is the usual case when the second argument is a character array:

```
DIRECTORY d;
char path[SZ_FNAME];if (!xvt_fsys_get_dir(&d))
    xvt_dm_post_error(
        "Can't get current directory.");
else if (!xvt_fsys_convert_dir_to_str(&d, path,
        SZ_FNAME(path)))
    xvt_dm_post_error(
        "Can't show current directory.");
else
    xvt_dm_post_note("Current directory is %s",
        path);
```

xvt_fsys_convert_str_to_dir

Convert String Path to Directory

Summary

```
BOOLEAN xvt_fsys_convert_str_to_dir(char *path,
    DIRECTORY *dirp)
```

`char` **`*path`**

String to be converted to directory.

`DIRECTORY *dirp`

Pointer to a `DIRECTORY`.

Description

This function converts the `NULL`-terminated string `path`, using local path-name syntax, to a **`DIRECTORY`**, returned through **`dirp`**. The `path` parameter must represent a valid platform-specific pathname.

This function is handy when pathnames are typed directly by the user, or stored as string resources.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful (on error).

See Also

```
DIRECTORY
xvt_fsys_convert_dir_to_str
xvt_fsys_set_dir
```

The "Files" chapter in the *XVT Portability Toolkit Guide*

Example

```
char buf[100];
DIRECTORY dir;if
(xvt_dm_post_string_prompt("Directory?", buf,
    sizeof(buf)) != NULL) {
    if (!xvt_fsys_convert_str_to_dir(buf, &dir))
        xvt_dm_post_error("Can't convert dir to str.");
    if (!xvt_fsys_set_dir(&dir))
        xvt_dm_post_error("Can't change directory.");
}
```

xvt_fsys_get_default_dir

Get Default Directory

Summary

```
void xvt_fsys_get_default_dir(DIRECTORY *dirp)
```

`DIRECTORY *dirp`

Pointer to a `DIRECTORY`.

Description

This function retrieves a `DIRECTORY` equivalent to the native system's concept of current or default directory specified in a relative sense. `xvt_fsys_get_default_dir` produces a `DIRECTORY` that is equivalent to using "." on UNIX, HPFS, or NTFS file systems.

You can use this function to assign a value to the `dir` member of a `FILE_SPEC` so as to specify a file in the current directory.

Return Value

`xvt_fsys_get_default_dir` has no return value. Instead, it stores the current directory in **dirp**.

Implementation Note

The difference between `xvt_fsys_get_default_dir` and `xvt_fsys_get_dir` can be clarified by considering the following XVT/Win32 or XVT/XM implementations:

- `xvt_fsys_get_default_dir` returns an empty string or "."
- `xvt_fsys_get_dir` returns its argument as an absolute path (e.g., **c:examples**)

See Also

```
DIRECTORY
FILE_SPEC
XVT_FILESYS_* Values
xvt_fsys_get_dir
xvt_fsys_set_dir
```

The "Files" chapter in the *XVT Portability Toolkit Guide*

Example

```
DIRECTORY d1, d2;xvt_fsys_get_default_dir(&d1);if
(!xvt_fsys_get_dir(&d2))
xvt_dm_post_fatal_exit("Can't get current directory.");
/*
    At this point d1 and d2 refer to the current directory.
    The variable d3 (declared and initialized elsewhere)
refers to some other directory.
*/
if (!xvt_fsys_set_dir(&d3))
    xvt_dm_post_fatal_exit("Can't change directory.");
/*  At this point d1 refers to the new current directory
    (same as d3), but d2 refers to the same directory it
    did before (which is no longer the current directory).
*/
```

xvt_fsys_get_dir

Get Current Directory

Summary

```
BOOLEAN xvt_fsys_get_dir(DIRECTORY *dirp)
```

`DIRECTORY *dirp`

Pointer to a directory.

Description

This function gets the current directory specified as an absolute path and returns its specification through the argument `dirp`.

The directory pointed to by `dirp` can be used in the **dir** field of a `FILE_SPEC` so as to specify a file located in the current directory. To change the returned directory into a string the user can recognize, you can pass it to the following:

```
char path [SZ_FNAME]
xvt_fsys_convert_dir_to_str(dirp, path,
    sizeof(path));
```

Note: The directory pointed to by `dirp` when `xvt_fsys_get_dir` returns is not the same as the directory pointed to by `dirp` when `xvt_fsys_get_default_dir` returns. `xvt_fsys_get_dir` specifies an absolute path, indicating the current directory, whereas `xvt_fsys_get_default_dir` specifies the current directory in a relative sense (e.g., an empty string or ".") on NTFS, or UNIX file systems.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful (on error). A `FALSE` return usually indicates that the current directory has somehow become invalid. An example of this happening is on network file systems where the file server becomes unreachable by the client.

Parameter Validity and Error Conditions

Severity error if `dirp` is `NULL`.

See Also

```
DIRECTORY
FILE_SPEC
SZ_FNAME
XVT_FILE_ATTR_* Constants
XVT_FILESYS_* Values
xvt_fsys_convert_dir_to_str
xvt_fsys_get_default_dir
xvt_fsys_get_file_attr
xvt_fsys_set_dir
```

The "Files" chapter in the *XVT Portability Toolkit Guide*

Example

```
static void showdir()
{
    DIRECTORY d;
    char path[SZ_FNAME];if (!xvt_fsys_get_dir(&d))
        xvt_dm_post_error(
            "Can't get current directory.");
    else if (xvt_fsys_get_file_attr ((FILE_SPEC*) (&d),
        XVT_FILE_ATTR_DIRSTR) == -1L)
        xvt_dm_post_error(
            "Can't show current directory.");
    else
        xvt_dm_post_note("Current directory is %s",
            path);
}
```

xvt_fsys_get_file_attr

Get a File Attribute

Summary

```
long xvt_fsys_get_file_attr(FILE_SPEC *file,
    long attr)

FILE_SPEC *file
```

File to query for attributes.

`long attr`

Defined constant for attribute to query.

Description

This function retrieves information about the file specified by the `file` parameter. The information available is equivalent to a subset of that provided by the `stat(3)` system call; this function is intended to be a portable replacement for `stat(3)`.

The `attr` parameter takes return values of `XVT_FILE_ATTR_*` Constants.

Return Value

If an error condition outside the scope of the current operation occurs, this function returns -1. Returned values from your time queries are in the appropriate format for the native platforms ANSI C time conversion functions. You can convert these values to useful strings by using the `ctime` function.

Parameter Validity and Error Conditions

Calling this function with a pointer that cannot be de-referenced as the `file` parameter results in a serious runtime error and a return value of -1. Calls with illegal values for the `attr` parameter generate a runtime warning and a return value of -1.

Implementation Note

The value of `time_t` type is dependent on the native implementation of the ANSI C Library, and it is not necessarily portable. If you need to transport time values between platforms, you can convert the `time_t` value into a string using `strftime`, or you might be able to save the values of the `struct tm` time structures.

On XVT/Mac, the string returned from `XVT_FILE_ATTR_DIRSTR` begins with the volume name and ends with a colon (e.g., **Mac HD:XVT:Examples:**). However, on other platforms the string does not end with a slash or back-slash (e.g., **c:examples**). You must take this into account when concatenating additional path or filename components onto the end of the directory string.

On XVT/Mac, the value returned from `XVT_FILE_ATTR_SIZE` represents the size of the data fork. On all platforms, the size represents the number of logical bytes present in the file and not necessarily the amount of space occupied by the file on the physical device.

See Also

`FILE_SPEC`
`XVT_FILE_ATTR_* Constants`
`xvt_fsys_convert_dir_to_str`
`xvt_fsys_set_file_attr`

The "Files" chapter in the *XVT Portability Toolkit Guide*
`ctime` in your C documentation

Example

```
...
FILE_SPEC aSpec;
long fileLen;
xvt_str_copy(aSpec.name, "foo.c");
fileLen = xvt_fsys_get_file_attr(&aSpec,
    XVT_FILE_ATTR_SIZE);
...
```

xvt_fsys_list_files

List Filenames

Summary

```
SLIST xvt_fsys_list_files(char *type, char *pat,
    BOOLEAN dirs)
```

`char *type`

File type or `DIR_TYPE` to list.

`char *pat`

Pattern to match.

`BOOLEAN dirs`

If `TRUE`, directory names are included; if `FALSE`, they are not included.

Description

This function gets an `SLIST` of files in the current directory with the specified `type`, or all types if `type` is an empty string. The order of filenames is indeterminate.

If `type` is equal to `DIR_TYPE`, only directories are listed.

If the string `pat` is non-`NULL`, only names matching that pattern, which can contain "`*`" and "`?`" wildcards, are listed. The wildcard "`*`"

matches a sequence of zero or more characters; the wildcard "?" matches any single character. Any number of these wildcards can be mixed arbitrarily with other characters. (XVT's matching is different from that performed by the NTFS file system, which allows only one "*" in the name part and one in the extension part.) The matching used by xvt_fsys_list_files is identical to the matching performed by xvt_str_match.

When calling xvt_fsys_list_files, all patterns are matched against the entire name; any characters to the right of a period (the extension on NTFS) is not treated any differently from the part before the period. Since only the names in the current directory are listed, no path separators or path components appear in the name. Pattern matching is case-insensitive.

If dirs is TRUE, directory names are included; if FALSE, they are omitted. To get directories only, set type to DIR_TYPE and dirs to TRUE. To get only non-directory files, set type to whatever you want and set dirs to FALSE. If type is DIR_TYPE and dirs is FALSE, you get an empty SLIST.

Return Value

SLIST containing names if successful; NULL if unsuccessful (on error).

Parameter Validity and Error Conditions

Severity error in the case of NULL type.

Implementation Note

On NTFS the extension is compared to the type argument, in addition to being matched against the pat argument. On the Mac any characters to the right of a period aren't considered to be special, nor is the period itself. The type is a separate attribute of the file, not part of the name.

Most types are different on the Mac from what they are on NTFS. For portability, you might want to store them as string resources, to be retrieved with xvt_res_get_str or xvt_res_get_str_list.

On UNIX, the type argument is meaningless.

On file systems that support drive letters, xvt_fsys_list_files inserts the names of the drives after the directories.

See Also

```
DIR_TYPE
FILE_SPEC
SLIST
XVT_FILESYS_* Values
xvt_slist_*
xvt_str_match
```

The "Files" chapter in the *XVT Portability Toolkit Guide*

Example

To list all files and directories:

```
SLIST x;
x = xvt_fsys_list_files("", "", TRUE);
```

To list all C source files:

```
x = xvt_fsys_list_files("", "*.c", FALSE);
```

This call also lists all C files on NTFS, but not on the other platforms, and is therefore *wrong*:

```
x = xvt_fsys_list_files("c", "*.*", FALSE);
```

This call also lists all C files on the Mac, but not on other platforms, so it too is *wrong*:

```
x = xvt_fsys_list_files("TEXT", "*.c", FALSE);
```

This call lists only directories whose names contain the letter "`T`" anywhere:

```
x = xvt_fsys_list_files(DIR_TYPE, "*T*", TRUE);
```

xvt_fsys_parse_pathname

Parse Multibyte String into Pathname Components

Summary

```
BOOLEAN xvt_fsys_parse_pathname (const char *mbs,
    char *volname, char *dirname, char *leafroot,
    char *leafext, char *leafvers)
```

`const char *mbs`

Full pathname string.

`char *volname`

Volume name string.

`char *dirname`

Directory name string.

`char *leafroot`

File leaf root string.

`char *leafext`

File extension string.

`char *leafvers`

File version string.

Description

This function parses the multibyte string `mbs` containing a file pathname and breaks it into pathname components. It places copies of the components into the passed arguments.

A pathname can be divided into the volume name, directory path, leaf root name, leaf extension, and leaf version. Not all of the parts are defined for all platforms. A leaf is defined as any specific file or directory at the end of the path. Leaf extension has a superficial meaning on UNIX and Mac. The convention used here, is that all text of one or more characters following the last "." in the leaf name is defined to be the extension.

The parts are broken down and returned to reflect the pathname passed in. Relative pathnames are allowed. This function does not expand any part of the pathname or attempt to interpret the pathname, it only parses it syntactically.

The directory and file do not need to exist to use this function. The function can be recursively called on the directory name to isolate any leaf in the path. Any output (`volname`, `dirname`, `leafroot`, `leaftext`, `leafvers`) pointer can be `NULL`; if `NULL`, that part is not returned.

The output arrays are assumed to be a minimum size. `dirname` must be at least `SZ_FNAME`, all others must be at least `SZ_LEAFNAME`.

Return Value

`TRUE` if successful; `FALSE` if a syntax error exists in the pathname for the current platform or any other error.

Parameter Validity and Error Conditions

Severity error is `wbs` in a `NULL` parameter.

See Also

`SZ_FNAME`
`SZ_LEAFNAME`
`XVT_FILESYS_* Values`
`xvt_fsys_build_pathname`
`xvt_fsys_convert_dir_to_str`
`xvt_fsys_convert_str_to_dir`

The "Files" chapter in the *XVT Portability Toolkit Guide*

Example

Macintosh Example

The Mac pathname **HD:xvt:mac.68k:pathname:file.ext** parses into the parts:

Volume	"HD"
Directory	":xvt:mac.68k:pathname:"
Fileroot	"file"
Extension	"ext"
Version	""

Recursively calling the function with `dirname` results in:

	1st pass	2nd pass	3rd pass
Volume	""	""	""
Directory	":xvt:mac.68k:"	":xvt:"	""
Fileroot	"pathname"	"mac"	"xvt"
Extension	"ext"	"68k"	""
Version	""	""	""

The relative Mac pathname **:foo:::bar:file.ext** recursively parses into the parts (recursively passing in Directory):

	1st pass	2nd pass	3rd pass	4th pass	5th pass
Volume	""	""	""	""	""
Directory	":foo:::bar:"	":foo:::"	":foo::"	":foo:"	""
Fileroot	"file"	"bar"	"::"	"::"	"foo"
Extension	"ext"	""	""	""	""
Version	""	""	""	""	""

Note: "::" on Mac means parent directory.

UNIX Example

The UNIX pathname **/xvt/unix.sparc/pathname/file.ext** parses into the parts:

Volume	""
Directory	"/xvt/unix.sparc/pathname/"
Fileroot	"file"
Extension	"ext"
Yersion	""

Recursively calling the function with `dirname` results in:

	1st pass	2nd pass	3rd pass
Volume	""	""	""
Directory	"/xvt/unix/sparc/"	"/xvt/"	""
Fileroot	"pathname"	"unix"	"xvt"
Extension	""	"sparc"	""
Yersion	""	""	""

The relative UNIX pathname **foo/../../bar/file.ext** recursively parses into the parts (recursively passing in Directory):

	1st pass	2nd pass	3rd pass	4th pass	5th pass
Volume	""	""	""	""	""
Directory	"foo/../../bar/"	"foo/../../"	"foo/../"	"foo/"	""
Fileroot	"file"	"bar"	".."	".."	"foo"
Extension	"ext"	""	""	""	""
Yersion	""	""	""	""	""

Note: ".." on UNIX means parent directory.)

Win32 Example

The Win32 pathname **C:dos.dirpathnamefile.ext** parses into the parts:

Volume	"C:"
Directory	"\xvt\dos.dir\pathname\"
Fileroot	"file"
Extension	"ext"
Yersion	""

Recursively calling the function with `dirname` results in:

	1st pass	2nd pass	3rd pass
Volume	""	""	""
Directory	"\xvt\dos.dir\"	"\xvt\"	""
Fileroot	"pathname"	"dos"	"xvt"
Extension	""	"dir"	""
Yersion	""	""	""

The relative Win32 pathname **foo....barfile.ext** recursively parses into the parts (recursively passing in `Directory`):

	1st pass	2nd pass	3rd pass	4th pass	5th pass
Volume	""	""	""	""	""
Directory	"foo\..\..\bar\"	"foo\..\..\"	"foo\..\"	"foo\"	""
Fileroot	"file"	"bar"	".."	".."	"foo"
Extension	"ext"	""	""	""	""
Yersion	""	""	""	""	""

Note: "**..**" on Win32 means parent directory.)

xvt_fsys_rem_file

Delete a File from the System

Summary

```
BOOLEAN xvt_fsys_rem_file(FILE_SPEC *file)
```

`FILE_SPEC *file`

File to be deleted.

Description

This file system function deletes a file from the system. An XVT `FILE_SPEC` identifies the file to be deleted.

Return Value

`FALSE` if the function succeeds; `TRUE` if it fails.

Parameter Validity and Error Conditions

If the file specifier is `NULL`, or if either the directory or file does not exist, XVT issues an error.

See Also

`FILE_SPEC`

The "Files" chapter in the *XVT Portability Toolkit Guide*

xvt_fsys_restore_dir

Restore Saved Directory

Summary

```
void xvt_fsys_restore_dir(void)
```

Description

This function restores the current directory to what it was when `xvt_fsys_save_dir` was called. If you want to avoid changing the current directory, then your application should bracket calls to functions that might change the current directory (such as `xvt_dm_post_file_open`) with calls to `xvt_fsys_save_dir` and `xvt_fsys_restore_dir`.

Parameter Validity and Error Conditions

If `xvt_fsys_save_dir` was not previously called, XVT issues an error.

See Also

`DIRECTORY`
`xvt_dm_post_file_open`
`xvt_fsys_save_dir`

The "Files" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_fsys_save_dir`.

xvt_fsys_save_dir

Save Current Directory

Summary

```
void xvt_fsys_save_dir(void)
```

Description

This function saves the current directory in a variable that XVT keeps internally, for restoration by a later call to `xvt_fsys_restore_dir`. Calls to `xvt_fsys_save_dir` and `xvt_fsys_restore_dir` do not stack; each call to `xvt_fsys_save_dir` causes the previously saved directory to be forgotten.

If you want to avoid changing the current directory, your application should bracket calls to functions that might change the current directory (such as `xvt_dm_post_file_open`) with calls to `xvt_fsys_save_dir` and `xvt_fsys_restore_dir`.

See Also

```
DIRECTORY
xvt_dm_post_file_open
xvt_fsys_restore_dir
xvt_fsys_set_dir_startup
```

The "Files" chapter in the *XVT Portability Toolkit Guide*

Example

```
xvt_fsys_save_dir();
xvt_fsys_set_dir_startup();
reprocess_startup_files();
xvt_fsys_restore_dir();
```

xvt_fsys_set_dir

Change Current Directory

Summary

```
BOOLEAN xvt_fsys_set_dir(DIRECTORY *dirp)

DIRECTORY *dirp
```

Pointer to an abstract object representing a directory in the local file system.

Description

This function changes to the directory specified by its argument. The `dirp` argument is a pointer to an abstract object representing a directory in the local file system. You must call `xvt_fsys_set_dir` whenever you use the standard C function `fopen` to open a file in directories other than the current directory. `fopen` does not accept a

DIRECTORY argument. It does accept a path as part of the filename, but the syntax of such paths is not portable.

Return Value

TRUE if successful; FALSE on error.

Parameter Validity and Error Conditions

Severity error in the case of NULL arguement dirp.

See Also

DIRECTORY
FILE_SPEC
FL_* Values for FL_STATUS
xvt_fsys_get_default_dir
xvt_fsys_restore_dir
xvt_fsys_save_dir
xvt_fsys_set_dir_startup

The "Files" chapter in the *XVT Portability Toolkit Guide*

Example

```
FILE_SPEC fs_in;memset(&fs_in, 0, sizeof(FILE_SPEC));
fs_in.type[0] = '0'; /* want all types */
xvt_fsys_get_default_dir(&fs_in.dir);
switch (xvt_dm_post_file_open(&fs_in,
    "Select input file...")) {
case FL_BAD:
    return;
case FL_CANCEL:
    return;
case FL_OK:
    break;
}
xvt_fsys_set_dir(&fs_in.dir);
if ((in = fopen(fs_in.name, "rb")) == NULL) {
    xvt_dm_post_error("Can't open file "%s".",
        fs_in.name);
    return;
}
```

xvt_fsys_set_dir_startup

Change to Startup Directory

Summary

```
void xvt_fsys_set_dir_startup(void)
```

Description

This function changes the default directory to the directory that was current when `the xvt_app_create` was called.

See Also

`DIRECTORY`
`xvt_app_create`
`xvt_fsys_get_default_dir`
`xvt_fsys_save_dir`
`xvt_fsys_set_dir`

The "Files" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_fsys_save_dir`.

---

xvt_fsys_set_file_attr

Set the File Attribute

Summary

```
BOOLEAN xvt_fsys_set_file_attr
    (FILE_SPEC *file, long attr, long value)
```

`FILE_SPEC *file`

File for which to set the attribute.

`long attr`

Attribute to set.

`long value`

Value to be assigned to the attribute.

Description

This function sets the attributes for the file represented by the input `FILE_SPEC` structure. The following attribute values are supported for `set` operations:

`XVT_FILE_ATTR_TYPESTR`

Assumes the `value` parameter is a pointer to a `NULL`-terminated `char` array, which is copied into the `type` member of the `FILE_SPEC` pointed to by the `file` parameter. The string is

truncated to four bytes. On XVT/Mac, if the file exists, this also sets the native system file type attribute.

`XVT_FILE_ATTR_CREATORSTR`

Assumes the `value` parameter is a pointer to a `NULL`-terminated `char` array, which is copied into the `creator` member of the `FILE_SPEC` pointed to by the `file` parameter. The string is truncated to four bytes. On XVT/Mac, if the file exists, this also sets the native system file creator attribute.

Return Value

`TRUE` if successful; `FALSE` if an error condition or problem occurs that makes it impossible to correctly complete the attribute set.

Parameter Validity and Error Conditions

Calling this function with a pointer that cannot be de-referenced as the `file` parameter results in a serious runtime error and returns a value of `FALSE`. If the file specified by `file` does not exist, XVT issues a warning and returns `FALSE`. Calls with illegal values for the `attr` parameter generate a runtime warning and a return value of `FALSE`.

Implementation Note

If the input string is too large for the destination field in the `FILE_SPEC`, the string is copied and truncated, and the function returns `FALSE`.

See Also

`FILE_SPEC`
`xvt_fsys_get_file_attr`
`xvt_fsys_convert_str_to_dir`

The "Files" chapter in the *XVT Portability Toolkit Guide*

xvt_gmem_*

Global Memory Management (Mac relocatable)

`xvt_gmem_alloc`
`xvt_gmem_free`
`xvt_gmem_get_size`
`xvt_gmem_lock`
`xvt_gmem_realloc`
`xvt_gmem_unlock`

xvt_gmem_alloc

Allocate Global Memory Block

Summary

```
GHANDLE xvt_gmem_alloc(long size)
```

`long size`

Size of the memory block to be allocated from the "global" heap.

Description

This function allocates memory from the "global" heap. The global heap is a separate memory manager that has special characteristics that vary between platforms. You might want to consider using global memory to reduce the heap fragmentation that can occur on XVT/Mac.

This function returns a `GHANDLE` representing the memory allocated. However, this is *not* a pointer. To get a pointer to the memory, call `xvt_gmem_lock`. When you are not using the pointer, call `xvt_gmem_unlock` to allow the system to possibly move the memory block and defragment the heap.

Once a global memory block is allocated, you can get its size with `xvt_gmem_get_size`, resize it with `xvt_gmem_realloc`, or free it with `xvt_gmem_free`.

You must not assume that the portable use of XVT global memory supports any of the tricks available on XVT/Mac. In particular, global memory is *not* shared memory! Do not attempt to pass `GHANDLE`s from one application to another, any more than you would pass a pointer from one application to another.

Return Value

A `GHANDLE` that identifies the block if successful; `(GHANDLE)0` if no suitable block can be allocated.

Implementation Note

XVT/Mac

If you are planning to run your application on the Mac, then you can use `xvt_gmem_alloc` to allocate memory that can be moved by the Mac operating system to another location. Doing so

avoids heap fragmentation, and allows your application to use less memory. Typically, the memory saved is on the order of 20%. Of course, the trade-off is that your application requires more complexity to manage the locking and unlocking required to use global memory, and performance suffers due to the locking and unlocking overhead.

All other platforms

`xvt_gmem_alloc` is implemented in terms of `malloc`. In this case, a `GHANDLE` is the same as a character pointer, and `xvt_gmem_lock` is a macro that returns its argument. Because of this, the global heap is the same as the local heap on those platforms. Therefore, using global memory has no benefit.

See Also

`GHANDLE`
`xvt_gmem_free`
`xvt_gmem_get_size`
`xvt_gmem_lock`
`xvt_gmem_realloc`
`xvt_gmem_unlock`
`xvt_mem_alloc`

The "Memory Allocation" chapter in the *XVT Portability Toolkit Guide*

---

xvt_gmem_free

Free Global Memory Block

Summary

```
BOOLEAN xvt_gmem_free(GHANDLE h)

GHANDLE h
```

Handle to the global memory block.

Description

This function frees the global memory block identified by the `GHANDLE` given as its argument. The block must have been previously allocated by a call to `xvt_gmem_alloc` or `xvt_gmem_realloc`. If locked (via `xvt_gmem_lock`), then it must be unlocked (with `xvt_gmem_unlock`) before it can be freed.

The topic `xvt_gmem_alloc` includes a discussion of global memory.

Return Value

`TRUE` if successful; `FALSE` on error. A `FALSE` return is usually the result of an attempt to free a locked block.

See Also

```
GHANDLE
xvt_gmem_alloc
xvt_gmem_lock
xvt_gmem_realloc
xvt_gmem_unlock
```

Example

```
GHANDLE h;if ((h = xvt_gmem_alloc(sizeof(DOC))) == NULL)
    return(FALSE);
...
if (!xvt_gmem_unlock(h) || !xvt_gmem_free(h))
    xvt_errmsg_sig(NULLWIN, SEV_ERROR,
        ERR_FAIL, "UNLOCK", 295,
            "Failed to unlock and free memory");
```

xvt_gmem_get_size

Get Size of Global Memory Block

Summary

```
long xvt_gmem_get_size(GHANDLE h)

GHANDLE h
```

Handle to the global memory block.

Description

This function returns the size in bytes of a memory block identified by the `GHANDLE` given as its argument. The block can either be locked or unlocked, but it cannot have already been freed. The size returned can be larger than the size given as the argument to `xvt_gmem_alloc` or `xvt_gmem_realloc` when the block was allocated, but it can't be smaller.

See the topic `xvt_gmem_alloc` for a discussion of global memory.

Return Value

The size of the block in bytes if successful; zero if `h` is not a valid memory handle. Blocks of size zero are not allowed.

See Also

GHANDLE
xvt_gmem_alloc
xvt_gmem_realloc

Example

See xvt_gmem_alloc.

xvt_gmem_lock

Lock Global Memory Block

Summary

```
char *xvt_gmem_lock(GHANDLE h)

GHANDLE h
```

Handle to the global memory block.

Description

This function is intended to lock a global memory block and return a pointer to it. A locked piece of global memory can't be moved by the system while it's locked (which would invalidate the pointer), so you should unlock it with xvt_gmem_unlock as soon as you can. The next time you need to access the block, you can get a fresh pointer with another call to xvt_gmem_lock.

For a discussion of global memory, see xvt_gmem_alloc.

Return Value

A pointer to the block if successful; NULL if the block cannot be locked. A NULL return usually means that the block is already locked.

Parameter Validity and Error Conditions

It is important to remember that calls to xvt_gmem_lock do not stack. Therefore, never attempt to lock a block that has already been locked. The result of doing so is undefined.

Implementation Note

On XVT/Mac, the memory is physically locked. xvt_gmem_lock should still be used to convert the GHANDLE to a pointer. On other systems, this function has no effect.

See Also

GHANDLE
xvt_gmem_alloc
xvt_gmem_realloc
xvt_gmem_unlock

xvt_gmem_realloc

Reallocate Global Memory Block

Summary

```
GHANDLE xvt_gmem_realloc(GHANDLE h, long size)
```

`GHANDLE h`

Handle to the global memory block.

`long size`

New size of the memory block.

Description

This function reallocates the global memory block associated with handle h to be of `size` bytes. The returned handle should be used to refer to the resized block. Since a new block might have been allocated (the data having been moved automatically), the returned handle might be different from the argument `h`. In this case, the old block is freed before `xvt_gmem_realloc` returns.

Before calling `xvt_gmem_realloc`, make sure that the block is unlocked with `xvt_gmem_unlock`.

Return Value

New handle to block (not necessarily equal to `h`) if successful; `(GHANDLE)0` on error, indicating insufficient memory.

See Also

GHANDLE
xvt_gmem_alloc
xvt_gmem_get_size
xvt_gmem_unlock

Example

```
GHANDLE h;...
if (xvt_gmem_get_size(h) < TBLK_SZ)
    if ((h = xvt_gmem_realloc(h, TBLK_SZ)) == NULL)
        return(FALSE);
```

xvt_gmem_unlock

Unlock Global Memory Block

Summary

```
BOOLEAN xvt_gmem_unlock(GHANDLE h)
```

`GHANDLE h`

Handle to the global memory block.

Description

This function unlocks the global memory block identified by the `GHANDLE` given as its argument. The block must have previously been allocated with `xvt_gmem_alloc` or `xvt_gmem_realloc` and locked with `xvt_gmem_lock`.

Unlocking a block doesn't disturb its contents, but it does invalidate the pointer that the previous call to `xvt_gmem_lock` returned. Be careful to avoid using this pointer after calling `xvt_gmem_unlock`. If you need to re-access the block, call `xvt_gmem_lock` again. The result of calling `xvt_gmem_unlock` on an unlocked block is undefined.

Don't keep blocks locked for a long period. Instead, bracket your accesses with paired calls to `xvt_gmem_lock` and `xvt_gmem_unlock`.

See the topic `xvt_gmem_alloc` for a discussion of global memory.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful (on error).

Parameter Validity and Error Conditions

If `h` is invalid, or if `h` is not locked, XVT issues an error.

See Also

```
GHANDLE
xvt_gmem_alloc
xvt_gmem_lock
xvt_gmem_realloc
```

xvt_help_*

Help Functions

```
xvt_help_assoc_all
xvt_help_begin_objclick
xvt_help_close_helpfile
xvt_help_disassoc_all
xvt_help_display_topic
xvt_help_end_objclick
xvt_help_get_flavor
xvt_help_get_menu_assoc
xvt_help_get_win_assoc
xvt_help_open_helpfile
xvt_help_process_event
xvt_help_search_topic
xvt_help_set_menu_assoc
xvt_help_set_win_assoc
```

xvt_help_assoc_all

Associate Help Topics with all Objects

Summary

```
void xvt_help_assoc_all(XVT_HELP_INFO hi, WINDOW win,
     long rid, WIN_DEF *wdef)
```

`XVT_HELP_INFO hi`

Help file information handle.

`WINDOW win`

Container; top-level window or dialog.

`long rid`

Resource ID of container.

`WIN_DEF *wdef`

Window definition.

Description

This function associates help topics with all GUI objects (controls and menus) contained by a window or dialog. The association between topics (by topic identifier) and GUI objects is provided by a help association file.

`xvt_help_assoc_all` requires a `WIN_DEF` structure for associating help topics with a container window. If `wdef` in `NULL`, the resource ID of the window (`rid`) is obtains the `WIN_DEF` structure.

Parameter Validity and Error Conditions

XVT issues an error for any of the following conditions:

- `hi` is not a valid help file information handle
- `win` is not a valid container window
- `rid` is not a valid resource ID

Return Value

None.

See Also

`WIN_DEF`
`XVT_HELP_INFO`
`xvt_help_disassoc_all`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

---

xvt_help_begin_objclick

Begin Object-Click Help Mode

Summary

```
void xvt_help_begin_objclick(XVT_HELP_INFO hi,
     WINDOW win, unsigned long flags)
```

`XVT_HELP_INFO hi`

Help file information handle.

`WINDOW win`

Container window.

`unsigned long flags`

Not currently used; pass zero for this parameter.

Description

This function initiates the object-click help mode. In the object-click help mode, the cursor changes to a "help" cursor (often a question mark) to indicate the change in mode. When the user selects an object, the help engine displays the help topic associated with the selected object. The cursor reverts to its previous state, and object-click mode is terminated.

Pass your application's currently active container window as the `win` parameter of this function. While object-click mode is in effect, the mouse cursor is trapped to this window.

Your application won't usually need to call this function, since it is generally called automatically in response to a menu selection. This function has no effect if object-click mode is already active.

Return Value

None.

Parameter Validy and Error Conditions

XVT issues an error if:

- `hi` is not valid
- `win` is not a valid container window

See Also

`XVT_HELP_INFO`
`xvt_help_end_objclick`
`xvt_help_open_helpfile`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

xvt_help_close_helpfile

Close an Open Help File

Summary

```
void xvt_help_close_helpfile(XVT_HELP_INFO hi)

XVT_HELP_INFO hi
```

Help file information handle.

Description

This function closes and frees all information associated with the opened help file `hi`. Call this function when your application no longer needs access to the help file.

Caution: Do not use `free` or `xvt_mem_free` to deallocate help file information handles.

Return Value

None.

Parameter Validity and Error Conditions

If `hi` is not valid, XVT issues an error.

See Also

```
XVT_HELP_INFO
xvt_help_open_helpfile
```

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

Example

```
static XVT_HELP_INFO hi;hi = xvt_help_open_helpfile
(fileSpec, 0L);
if (hi == (XVT_HELP_INFO)0){
    /* display an informative error message, or try
        another directory */
}
...
xvt_help_close_helpfile(hi);
```

xvt_help_disassoc_all

Remove Help Topic Associations from all Objects

Summary

```
void xvt_help_disassoc_all(XVT_HELP_INFO hi,
    WINDOW parent_win)

XVT_HELP_INFO hi
```

Help file information handle.

```
WINDOW parent_win
```

Window whose help associations are to be removed.

Description

Removes all help topic associations for a window, and any controls, child windows, and menu items it contains. This function is called internally when the parent `WINDOW` is destroyed; your application does not need to call this function.

Return Value

None.

See Also

`XVT_HELP_INFO`
`xvt_help_open_helpfile`
`xvt_help_set_menu_assoc`
`xvt_help_set_win_assoc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_help_set_win_assoc`.

---

xvt_help_display_topic

Display a Help Topic

Summary

```
void xvt_help_display_topic(XVT_HELP_INFO hi,
    XVT_HELP_TID topic_id)
```

`XVT_HELP_INFO hi`

Help file information handle.

`XVT_HELP_TID topic_id`

Help topic identifier.

Description

This function displays the topic with reference identifier `topic_id` from the compiled help file `hi`. The topic will be displayed within the viewer window (either the help system viewer or native viewer). If the viewer window is already active for the application, then the

topic will replace the currently viewed topic; otherwise a new viewer window is invoked.

Return Value

None.

Parameter and Validity Conditions

XVT issues an error if:

- `hi` is not valid
- `topic_id` is `NULL_TID`

See Also

```
XVT_HELP_INFO
XVT_HELP_TID, NULL_TID
xvt_help_open_helpfile
```

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

Example

```
static XVT_HELP_INFO hi;hi =
xvt_help_open_helpfile(fileSpec, 0L);/* Display a topic,
and create new window */
xvt_help_display_topic(hi, XVT_TPC_INDEX);
.../* Display a different topic: */
xvt_help_display_topic(hi, XVT_TPC_KEYBOARD);
```

xvt_help_end_objclick

Cancel Object-Click Help Mode

Summary

```
void xvt_help_end_objclick(XVT_HELP_INFO hi)

XVT_HELP_INFO hi
```

Help file information handle.

Description

This function terminates the object-click help mode. Usually your application does not need to call this function, since the object-click mode is normally terminated automatically by the help system in response to the user’s selection.

See Also

```
XVT_HELP_INFO
xvt_help_begin_objclick
xvt_help_open_helpfile
```

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

xvt_help_get_flavor

Get Help Viewer Flavor to be Used by the Application

Summary

```
XVT_HELP_FLAVOR xvt_help_get_flavor(void)
```

Description

This function returns the flavor (configuration) of the help viewer that is to be used by the application. The configuration of the help viewer is established at application build (link) time. A help file must be opened prior to calling this function in order for the return value to be accurate. Otherwise, it returns `XVT_HELP_FLAVOR_NONE`.

Return Value

The flavor of the help viewer in use by the application, of type `XVT_HELP_FLAVOR`.

See Also

```
XVT_HELP_* Values for XVT_HELP_FLAVOR
xvt_help_open_helpfile
```

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

Example

```
/* do something based on the viewer configuration */
    switch (xvt_help_get_flavor())
{
    case XVT_HELP_FLAVOR_PORTBND:
        /* do something for the portable bound
            viewer */
        break;
    case XVT_HELP_FLAVOR_NTVSRV:
        /* do something else for the native
            stand-alone viewer */
        break;
}
```

xvt_help_get_menu_assoc

Get Help Topic Associated with a Menu Item

Summary

```
XVT_HELP_TID xvt_help_get_menu_assoc(XVT_HELP_INFO hi,
    WINDOW win, MENU_TAG tag)
```

`XVT_HELP_INFO hi`

Help file information handle.

`WINDOW win`

Window that contains the menu item.

`MENU_TAG tag`

Identifier tag of the menu item.

Description

This function returns the help topic identifier associated with the menu item `tag`. `win` is the window that contains the menu. The association must be previously created with a call to `xvt_help_set_menu_assoc`.

Return Value

The help topic identifier, of type `XVT_HELP_TID`, if successful; `NULL_TID` if no topic is associated with the menu item.

Parameter and Validity Conditions

XVT issues an error if:

- `hi` is not valid

- `win` is not a valid window

See Also

`XVT_HELP_INFO`
`xvt_help_open_helpfile`
`xvt_help_set_menu_assoc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

---

xvt_help_get_win_assoc

Retrieve the Help Topic Associated with an Object

Summary

```
XVT_HELP_TID xvt_help_get_win_assoc(XVT_HELP_INFO hi,
    WINDOW win)
```

`XVT_HELP_INFO hi`

Help file information handle.

`WINDOW win`

Window, dialog, or control.

Description

This function returns the identifier of the help topic associated with `win`, a window, dialog, or control.

Return Value

A help topic identifier if successful; `NULL_TID` if no topic is associated with `win`.

Parameter and Validity Conditions

If `hi` or `win` is not valid, XVT issues an error.

See Also

`XVT_HELP_INFO`
`xvt_help_get_menu_assoc`
`xvt_help_open_helpfile`
`xvt_help_set_win_assoc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

xvt_help_open_helpfile

Load a Help File

Summary

```
XVT_HELP_INFO xvt_help_open_helpfile(FILE_SPEC *fs,
    unsigned long flags)
```

`FILE_SPEC *fs`

Pointer to help file specification.

`unsigned long flags`

Help system options (see description below).

Description

This function opens the specified help file and returns a handle to the help file's information. The help file specification pointer should be to a valid, compiled help file; if not, `xvt_help_open_helpfile` returns an error.

`xvt_help_open_helpfile` uses the following search procedure to locate the help file:

1. If a full path is specified, look only there (e.g., **/usr/apps/help1.csc**, **d:apps elp1.csc** or **HD:Applications:help1.csc** indicates that the compiled help file named **help1.csc** must be in the specified directory).
2. Look in the current directory (current when this function is called).
3. Look in the directory given by the `HELP` environment variable.
4. Look in `XVTPATH` directories.
5. Look in `PATH` directories.

The following is a list of the help system `flags` that apply to the portable viewer. The `flags` information can contain zero or more of the following values, OR'd together:

`HSF_APPNAME_TITLE`

Normally, the help system displays the current topic in a help topic window title bar. If the `HSF_APPNAME_TITLE` flag option is

set, your application name, as defined in the XVT_CONFIG structure, is used instead.

HSF_INDEX_ON_DISK

If this flag option is used, the topic index is maintained on disk. By default, the index is maintained in application memory. This option is useful for low-memory environments.

HSF_NO_HELPMENU_ASSOC

Normally, the help system automatically associates help topics with the help menu items (e.g., "Help On Help," "Search"). If the this flag is set, the association is not performed.

HSF_NO_TOPIC_WARNING

If this flag option is used, the help system does not display any error messages when a requested topic is not found in the help file. If this flag is not set, a "topic not found" message is displayed if a topic cannot be found.

HSF_NO_BEEP_MODAL

Most native window systems do not allow the user to manipulate menus and windows when a modal dialog is active. Therefore, the user could not operate the help viewer if help was requested for a modal dialog. If HSF_NO_BEEP_MODAL is set, the system beeps if help is requested for a modal dialog. If this flag is not set, requests for help on modal dialogs are silently ignored.

Note: The HSF_NO_BEEP_MODAL flag only applies to the application-bound help viewer. It has no effect on native and standalone help viewers.

Return Value

A valid XVT_HELP_INFO help file descriptor if successful; zero on error.

Note: The returned XVT_HELP_INFO value is passed to every other help system function.

Implementation Note

The HSF_APPNAME_TITLE flag has no effect on the native MS-Windows help viewer.

The native help system (Win32 WinHelp) is passed a complete path name, according to the above search rules, to the specified **.hlp** file.

On XVT/Mac, the help file must be located in the directory where the application or help viewer (if using client-server help) is installed.

See Also

```
FILE_SPEC
XVT_CONFIG
XVT_HELP_INFO
xvt_help_close_helpfile
```

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

Example

```
FILE_SPEC fspec;
XVT_HELP_INFO hi;xvt_fsys_get_default_dir (&fspec);
fspec.type[0] = '0';
fspec.creator[0] = '0';
xvt_str_copy(fspec.name, xvt_config. base_appl_name);
hi = xvt_help_open_helpfile(&&fspec, 0L);
if(!hi){
     xvt_dm_post_note("Unable to open help file")
}
```

xvt_help_process_event

Pass Event to Help Event Handler

Summary

```
BOOLEAN xvt_help_process_event(XVT_HELP_INFO hi,
     WINDOW win, EVENT *ev)
```

`XVT_HELP_INFO hi`

Help file information handle.

`WINDOW win`

Window in which the event occurred.

`EVENT *ev`

XVT event pointer.

Description

This function processes any events intended for the help system. Call this function in your window and dialog event handlers before processing any events. If this call returns `TRUE`, then the event was consumed by the help system, and your event handler function should return without any further action.

You call this function only if you are customizing the behavior of the help system. Normally, it is called for you automatically.

Return Value

If `TRUE`, the event was processed by the help system, and no further action is necessary for this event. If `FALSE`, the event is not useful to the help system, and should be processed by your event handler.

Parameter and Validity Conditions

XVT issues an error if:

- `hi` is not valid
- `win` is not valid
- `ev` is `NULL`

See Also

```
ATTR_HELP_CONTEXT
ATTR_HELP_HOOK
XVT_HELP_INFO
xvt_help_open_helpfile
```

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

Example

See the example under `xvt_help_set_win_assoc`.

---

xvt_help_search_topic

Display a Help Topic and Begin a Keyword Search

Summary

```
void xvt_help_search_topic(XVT_HELP_INFO hi,
     XVT_HELP_TID topic_id, char *str)
```

`XVT_HELP_INFO hi`

Help file information handle.

`XVT_HELP_TID topic_id`

Help topic identifier.

`char *str`

Search keyword.

Description

This function displays the topic identified by `topic_id` with the help viewer, and starts a keyword search. `str` contains the search keyword.

Return Value

None.

Parameter and Validity Conditions

XVT issues an error if `hi` is not valid

See Also

`XVT_HELP_INFO`
`XVT_HELP_TID, NULL_TID`
`xvt_help_display_topic`
`xvt_help_open_helpfile`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

xvt_help_set_menu_assoc

Associate a Help Topic with a Menu Item

Summary

```
void xvt_help_set_menu_assoc(XVT_HELP_INFO hi,
     WINDOW win, MENU_TAG menu_tag,
    XVT_HELP_TID topic_id, unsigned long flags)
```

`XVT_HELP_INFO hi`

Help file information handle.

`WINDOW win`

Parent window of menu item.

`MENU_TAG menu_tag`

Menu item identifier.

`XVT_HELP_TID topic_id`

Help topic identifier.

`unsigned long flags`

Currently not used--pass zero for this parameter.

Description

This function associates the help topic with the given menu item. The association remains in effect until a subsequent call with a `topic_id` of `NULL_TID`.

The `parent_window` can be a valid top-level or task window, or `NULL_WIN`. If it is `NULL_WIN`, then any parent that has `menu_tag` in its menubar will use the topic identified by `topic_id`.

Call this function on each `MENU_TAG` with a `topic_id` of `NULL_TID` to remove the help topic associations.

Note: The help system automatically removes associations for all `MENU_TAG` values when the `parent_window` is being destroyed.

Return Value

None.

Parameter and Validity Conditions

XVT issues an error if:

- `hi` is not valid
- `win` is not `NULL_WIN` and not valid

See Also

```
MENU_TAG
XVT_HELP_INFO
XVT_HELP_TID, NULL_TID
xvt_help_disassoc_all
xvt_help_open_helpfile
xvt_help_set_win_assoc
```

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

Example

See the example under `xvt_help_set_win_assoc`.

xvt_help_set_win_assoc

Associate a Help Topic with a Window, Dialog, or Control

Summary

```
void xvt_help_set_win_assoc(XVT_HELP_INFO hi,
     WINDOW win, XVT_HELP_TID id, unsigned long flags)
```

`XVT_HELP_INFO hi`

Help file information handle.

`WINDOW win`

Window, dialog, or control.

`XVT_HELP_TID id`

Help topic identifier.

`unsigned long flags`

Currently not used; pass zero for this parameter.

Description

This function associates the help topic identifier `id` with the window object `win`. The association remains in effect until a subsequent call with a `topic_id` of `NULL_TID`.

When you no longer require associated help, call this function on each object with a `copic_id` of `NULL_TID`.

Note: When a parent window is being `destroyed`, the help system removes associations for its child `WINDOW`s and controls.

Return Value

None.

Parameter and Validity Conditions

If `hi` or `win` are not valid, XVT issues an error.

See Also

`XVT_HELP_INFO`
`XVT_HELP_TID, NULL_TID`
`xvt_help_assoc_all`
`xvt_help_disassoc_all`
`xvt_help_open_helpfile`
The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

Example

```
static XVT_HELP_INFO hi = NULL_HELP_INFO;long
XVT_CALLCONV1 task_eh(WINDOW win, EVENT *ep)
{
    switch (ep->type)
    {
    ...
    case E_CREATE:
        {
            FILE_SPEC fs;/* open help file */
            memset(&fs, 0, sizeof(FILE_SPEC));
            strcpy(fs.name, "MyHelpFile");
            hi = xvt_help_open_helpfile(&fs, 0L);
            if (!hi) xvt_dm_post_note ("Unable to open
                help file");
        }
        break;
    case E_DESTROY:
        /* close help file */
        if (hi != NULL_HELP_INFO)
            xvt_help_close_helpfile(hi);
        hi = NULL_HELP_INFO;
        break;
    ...
    }
    return 0L;
}
long XVT_CALLCONV1 win_eh(WINDOW win, EVENT *ep)
{
    WINDOW ctlWin;
    MENU_TAG my_menu_tag;switch (ep->type)
    {
    ...
    case E_CREATE:
        /* associate a help topic with a top level
            container and a menu tag */
        if (hi) {
        xvt_help_set_win_assoc(hi, win,
            MY_WIN_TOPIC_ID, 0L);
        xvt_help_set_menu_assoc(hi, win, my_menu_tag,
            MY_MENU_TOPIC_ID, 0L);
        /* associate a help topic with a control */
        ctlWin = xvt_win_get_ctl(win,
            MY_PUSHBUTTON_RES_ID);
        xvt_help_set_win_assoc(hi, ctlWin,
            MY_PUSHBUTTON_TOPIC_ID, 0L);
        }
        break;
    ...
    }
    return 0L;
}
```

xvt_html_*

HTML Control Functions

```
xvt_html_get_url
xvt_html_set_url
xvt_html_get_url_intercept
xvt_html_set_url_intercept
```

xvt_html_get_url

Get URL of HTML Control

Summary

```
char *xvt_html_get_url(WINDOW win, char *url, int sz_url)
```

`WINDOW win`

HTML control whose URL is to be retrieved.

`char *url`

Buffer to hold URL.

`int sz_url`

Maximum buffer capacity.

Description

This function gets the Universal Resource Locator (URL) of an HTML control and stores it in url. The maximum capacity of url (including the NULL-terminator) is sz_url. The URL is truncated as necessary to fit into url.

Calling xvt_html_get_url on other controls issues an error.

Return Value

Pointer to url if successful; NULL if the control type does not have a valid URL.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

- win must be a valid WINDOW of type WC_HTML
- url must not be NULL

- sz_url must be greater than zero

See Also

```
WC_* Values for WIN_TYPE
WINDOW
xvt_html_set_url
xvt_vobj_get_title
```

xvt_html_set_url

Set URL of HTML Control

Summary

```
void xvt_html_set_url(WINDOW win, char *url)
```

`WINDOW win`

HTML control whose URL is to be retrieved.

`char *url`

URL to be set.

Description

This function changes the Universal Resource Locator (URL) of an HTML control to the NULL-terminated string pointed to by url. The URL must be fully qualified and adhere to composition standards, i.e. 'file:///C:/mydoc.htm' or 'http://www.xvt.com'. Depending on the URL, some platforms might spawn the default Internet browser or application to complete the request.

Control update will not occur for an invalid URL.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

- win must be a valid WINDOW of type WC_HTML
- url must not be NULL

See Also

```
WC_* Values for WIN_TYPE
WINDOW
xvt_html_get_url
xvt_vobj_get_title
```

xvt_html_get_url_intercept

Retrieve URL Intercept Handler for HTML Control

Summary

```
XVT_HTML_URL_INTERCEPT_HANDLER
    xvt_html_get_url_intercept(WINDOW win)
```

`WINDOW win`

HTML control whose URL intercept handler is to be retrieved.

Description

This function gets the current Universal Resource Locator (URL) intercept handler for an HTML control. The following are typical uses for xvt_html_get_url_intercept:

- You can temporarily save the current intercept handler and restore it later with a call to xvt_html_set_url_intercept; you may want to do this if you temporarily override the intercept handler and wish to put it back later.
- You can retrieve the current intercept handler from one HTML control and assign it to another HTML control.
- You can save the intercept handler, reassign a new intercept handler, and "preprocess" URLs with the new intercept handler before it invokes the original intercept handler, effectively chaining together intercept handlers.

Return Value

The XVT_HTML_URL_INTERCEPT_HANDLER registered to the HTML control.

Parameter Validity and Error Conditions

XVT issues an error if the following condition is not met:

- win must be a valid WINDOW of type WC_HTML

See Also

```
XVT_HTML_URL_INTERCEPT_HANDLER
WC_* Values for WIN_TYPE
WINDOW
xvt_html_set_url_intercept
```

Example

This code uses xvt_html_get_url_intercept to get the URL intercept handler so that it can be assigned to another HTML control.

```
/* Get the URL intercept handler */
XVT_HTML_URL_INTERCEPT_HANDLER urlIH =
xvt_html_get_url_intercept(myHTMLCtl);
xvt_html_set_url_intercept(myNewHTMLCtl, urlIH);
```

xvt_html_set_url_intercept

Set URL Intercept Handler for HTML Control

Summary

```
void xvt_html_set_url_intercept(WINDOW win,
   XVT_HTML_URL_INTERCEPT_HANDLER fcn)
```

`WINDOW win`

HTML control whose URL intercept handler is to be set.

`XVT_HTML_URL_INTERCEPT_HANDLER fcn`

Application-defined URL intercept handler. Passing a value of NULL removes any previously set URL intercept handlers.

Description

This function sets the current Universal Resource Locator (URL) intercept handler for an HTML control. The URL intercept handler provides the ability to intercept the requested URL prior to processing. Possible uses of an URL intercept handler include:

- Implementation of a history list
- Redirection of URLs

Parameter Validity and Error Conditions

XVT issues an error if the following condition is not met:

- win must be a valid WINDOW of type WC_HTML

See Also

```
XVT_HTML_URL_INTERCEPT_HANDLER
WC_* Values for WIN_TYPE
WINDOW
xvt_html_get_url_intercept
```

Example

This code uses xvt_html_set_url_intercept to set an URL intercept handler to redirect URLs.

```
BOOLEAN myInterceptHdlr(WINDOW win, char **url)
{
    char localURL[] = "http://www.xvt.com";
    char errURL[] = "file://c:/my_app/errpage.htm";


    /* If URL is not local, redirect to error page. */
    if (strncmp(*url, localURL, sizeof(localURL)) == 0)
    {
        /* url was allocated using xvt_mem_alloc.
           According to documentation, if we want to
           change url, it must be freed using
           xvt_mem_free to avoid memory leaks. */
       xvt_mem_free(*url);

       /*  Allocate memory for url based on the length of
           our new URL */
        *url = xvt_mem_alloc(sizeof(errURL));

        /* Copy the new URL into url */
        strcpy(*url, errURL);
    }
   /* Returning TRUE notifies calling function to process
       the URL in url Returning FALSE notifies the
       calling function not to process the URL in url */
    return TRUE;
}


long XVT_CALLCONV1 myWindow_eh(WINDOW win, EVENT *ep)
{
    switch(ep->type)
    {
        case E_CREATE:
        ...
        xvt_html_set_url_intercept(xvt_win_get_ctl(win,
           HTML_CTL), myInterceptHdlr);
        ...
        break;
...
}
return (*save_eh)(win, ep);
}
```

xvt_image_*

Image Functions

```
xvt_image_create
xvt_image_duplicate
xvt_image_destroy
xvt_image_fill_rect
xvt_image_get_clut
xvt_image_get_dimensions
xvt_image_get_format
xvt_image_get_from_pmap
xvt_image_get_ncolors
xvt_image_get_pixel
xvt_image_get_resolution
xvt_image_get_scanline
xvt_image_read_*
xvt_image_read_bmp
xvt_image_read_bmp_from_iostr
xvt_image_read_gif
xvt_image_read_gif_from_iostr
xvt_image_read_jpg
xvt_image_read_jpg_from_iostr
xvt_image_read_macpict
xvt_image_read_macpict_from_iostr
xvt_image_read_xbm
xvt_image_read_xbm_from_iostr
xvt_image_read_xpm
xvt_image_read_xpm_from_iostr
xvt_image_set_clut
xvt_image_set_ncolors
xvt_image_set_pixel
xvt_image_set_resolution
xvt_image_transfer
xvt_image_write_bmp_to_iostr
xvt_image_write_macpict_to_iostr
```

xvt_image_create

Create a New Image

Summary

```
XVT_IMAGE xvt_image_create(XVT_IMAGE_FORMAT format,
    short width, short height,
    XVT_IMAGE_ATTR reserved)

XVT_IMAGE_FORMAT format
```

Color format of the new image.

`short width`

Width of the new image, in pixels.

`short height`

Height of the new image, in pixels.

`XVT_IMAGE_ATTR reserved`

Currently not used; pass `NULL` for this parameter.

Description

This function creates an `XVT_IMAGE` object of the specified format, width, and height. Use this function to allocate memory for images, rather than using `malloc` or `xvt_mem_alloc`.

The contents (pixel values) of the image are not defined after creation. Use `xvt_image_fill_rect` to fill the image with a solid color.

A newly created image with color format `XVT_IMAGE_CL8` or `XVT_IMAGE_MONO` has a color look-up table with two entries (the colors black and white). The number of colors in use by the image is set to two.

Return Value

The newly created `XVT_IMAGE` handle if successful; `NULL` if there is insufficient memory to create the image, or if `height` or `width` is less than zero.

Parameter Validity and Error Conditions

XVT issues an error if `format` is not one of the supported color format types.

See Also

```
XVT_IMAGE
XVT_IMAGE_ATTR
XVT_IMAGE_FORMAT
xvt_dwin_draw_image
xvt_image_destroy
xvt_image_fill_rect
xvt_image_get_pixel
xvt_image_set_clut
xvt_image_set_ncolors
xvt_image_set_pixel
xvt_image_read_*
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

```
/* rotates an image 180 degrees */
    void
image_rotate_180(XVT_IMAGE image)
{
    XVT_IMAGE new_image;
    short height, width;
    /* create new_image for rotated image */
    xvt_image_get_dimensions(image, &width, &height);
    new_image = xvt_image_create(
        xvt_image_get_format(image), width, height,
        (XVT_IMAGE_ATTR) NULL);
    if (new_image)
    {
        short h, v;
        RCT rect;
        /* set up colors for new_image */
        short ncolors, c;
        ncolors = xvt_image_get_ncolors(image);
        xvt_image_set_ncolors(new_image, ncolors);
        for (c = 0; c < ncolors; c++)
            xvt_image_set_clut(new_image, c,
                xvt_image_get_clut(image, c));
        /* rotate image pixels */
        for (v = 0 ; v < height; v++)
            for (h = 0; h < width; h++)
                xvt_image_set_pixel(new_image,
                    width-1-h, height-1-v,
        xvt_image_get_pixel(image, h, v));
        /* transfer new_image to image */
        xvt_rect_set(&rect, 0, 0, width, height);
        xvt_image_transfer(image, new_image, &rect,
            &rect);
        /* destroy new_image */
        xvt_image_destroy(new_image);
    }
}
```

xvt_image_duplicate

Make a copy of an image

Summary

```
XVT_IMAGE xvt_image_duplicate (XVT_IMAGE image);
```

```
XVT_IMAGE image
```

The image to duplicate.

Description

This function returns a copy of the image passed. The new image has all of the same attributes as the original image. The returned image is a true copy so `xvt_image_destroy` should be used to destroy it.

Return Value

The image iin an XVT_IMAGE variable if successful; `NULL` on error.

Parameter Validity and Error Conditions

XVT issues an error if the image is `NULL` or invalid.

See Also

`xvt_image_destroy`

---

xvt_image_destroy

Destroy an Image

Summary

```
void xvt_image_destroy(XVT_IMAGE image)
```

`XVT_IMAGE image`

Image to be destroyed.

Description

This function destroys an image, freeing the memory it occupies. The `image` variable is not valid after calling this function, and its value should not be used.

Caution: Do not use `free` or `xvt_mem_free` to deallocate the memory occupied by an image. Use this function instead.

Return Value

None.

Parameter Validity and Error Conditions

XVT issues an error if `image` is `NULL` or invalid.

See Also

`XVT_IMAGE`
`xvt_image_create`

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_image_create`.

xvt_image_fill_rect

Fill a Rectangular Area of an Image

Summary

```
void xvt_image_fill_rect(XVT_IMAGE image, COLOR color,
     RCT *rectp)
```

`XVT_IMAGE image`

Destination image.

`COLOR color`

Color with which to fill area.

`RCT *rectp`

Pointer to rectangle delimiting area to fill.

Description

This function fills a rectangular region in an image with one color value. If there is no color in the image that matches the fill color exactly, the nearest color is used.

If any portion of `rectp` lies outside the boundary of the image, the portion(s) will be clipped.

Return Value

None.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `image` is `NULL` or invalid
- `rectp` is `NULL`

See Also

```
COLOR
RCT
XVT_IMAGE
xvt_image_set_pixel
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

xvt_image_get_clut

Get a Color Entry Look-up Table

Summary

```
COLOR xvt_image_get_clut(XVT_IMAGE image, short index)
```

`XVT_IMAGE image`

Image from which to retrieve the look-up table.

`short index`

Index of color entry to retrieve.

Description

This function retrieves the color entry in an image's look-up table. `image` must have color format `XVT_IMAGE_CL8` or `XVT_IMAGE_MONO`.

Return Value

The `COLOR` value if successful; `COLOR_INVALID` if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if any of the following are true:

- `image` is `NULL` or invalid
- `index` is outside of the range [0, 255] for `XVT_IMAGE_CL8` or [0,1] for `XVT_IMAGE_MONO`.
- The image's color format is not `XVT_IMAGE_CL8` or `XVT_IMAGE_MONO`.

See Also

```
XVT_IMAGE
XVT_IMAGE_FORMAT
xvt_image_set_clut
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_image_create`.

xvt_image_get_dimensions

Get an Image's Width and Height

Summary

```
void xvt_image_get_dimensions(XVT_IMAGE image,
     short *width, short *height)
```

`XVT_IMAGE image`

Image from which to retrieve the dimensions.

`short *width`

Pointer to the image's width (in pixels).

`short *height`

Pointer to the image's height (in pixels).

Description

This function returns the dimensions of an image object, in the variables pointed to by `width` and `height`.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `image` is `NULL` or invalid
- `width` or `height` is `NULL`

See Also

`XVT_IMAGE`
`xvt_dwin_draw_image`
`xvt_image_create`

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

See the examples for `xvt_dwin_draw_image` and `xvt_image_create`.

xvt_image_get_format

Get an Image's Format Type

Summary

```
XVT_IMAGE_FORMAT xvt_image_get_format(XVT_IMAGE image)
```

`XVT_IMAGE image`

Image from which to retrieve the format.

Description

This function returns the color format of an image.

Parameter Validity and Error Conditions

XVT issues an error if `image` is `NULL` or invalid.

See Also

`XVT_IMAGE`
`XVT_IMAGE_* Values for XVT_IMAGE_FORMAT`
`xvt_image_create`

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_image_create`.

xvt_image_get_from_pmap

Transfer a Pixmap to an Image

Summary

```
void xvt_image_get_from_pmap(XVT_IMAGE dst_image,
     XVT_PIXMAP src_pmap, RCT *dst_rectp,
    RCT *src_rectp)
```

`XVT_IMAGE dst_image`

Destination image.

`XVT_PIXMAP src_pmap`

Source pixmap.

`RCT *dst_rectp`

Pointer to a rectangle that delimits the destination region, in the coordinates of `dst_image`. If this rectangle is empty, no image data is transferred.

`RCT *src_rectp`

Pointer to a rectangle that delimits the source region, in the coordinates of `src_pmap`.

Description

This function copies the contents of a rectangular region in the source pixmap into a rectangular region in the destination image. Colors in the source pixmap are mapped into the closest matching colors in the destination image.

If `*src_rctp` and `*dst_rctp` are not congruent, this function translates and scales the source region as necessary to fit it into the destination rectangle. Any parts of the source or destination rectangles that fall outside of the bounds of their respective containers are ignored.

To copy the entire source pixmap, use the rectangle (`0`, `0`, `width`, `height`) for the source rectangle, where width and height are the dimensions of the source pixmap. To fill the entire destination image, use a similar rectangle for the destination rectangle. In this case, width and height are the dimensions of the destination image.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `dst_image` is `NULL` or invalid
- `src_pmap` is `NULL` or invalid
- `src_rectp` is `NULL` or empty

See Also

```
RCT
XVT_IMAGE
XVT_PIXMAP
xvt_dwin_draw_image
xvt_dwin_draw_pmap
xvt_image_create
xvt_image_get_dimensions
xvt_image_transfer
xvt_pmap_create
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

xvt_image_get_ncolors

Get the Number of Colors in an Image

Summary

```
short xvt_image_get_ncolors(XVT_IMAGE image)
```

`XVT_IMAGE image`

Image from which to retrieve the number of colors.

Description

This function returns the number of colors in an image's look-up table and is useful only for images with `XVT_IMAGE_CL8` or `XVT_IMAGE_MONO` formats. (The value 2 is always returned for `XVT_IMAGE_MONO`.)

Return Value

If successful, the number of colors in the image's look-up table. If the image's color format is not `XVT_IMAGE_CL8` or `XVT_IMAGE_MONO`, this function returns zero.

Parameter Validity and Error Conditions

XVT issues an error if `image` is `NULL` or invalid.

See Also

```
XVT_IMAGE
XVT_IMAGE_* Values for XVT_IMAGE_FORMAT
xvt_image_create
xvt_image_set_ncolors
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_image_create`.

xvt_image_get_pixel

Get the Color of a Pixel in an Image

Summary

```
COLOR xvt_image_get_pixel(XVT_IMAGE image, short x,
     short y)
```

`XVT_IMAGE image`

Image containing the pixel.

`short x`

Horizontal coordinate of the pixel.

`short y`

Vertical coordinate of the pixel.

Description

This function returns the color value of a pixel in an image. Using this function is more convenient than retrieving the pixels directly (using `xvt_image_get_scanline`), since it handles the arithmetic for the array addressing and converting colors to different color formats.

Return Value

The `COLOR` value of the specified pixel if successful; `COLOR_INVALID` if unsuccessful (on error).

Parameter Validity and Error Conditions

If the `image` is invalid, or if the coordinate is outside of the range of dimensions, `COLOR_INVALID` is returned, and XVT issues an error.

See Also

```
XVT_IMAGE
xvt_image_create
xvt_image_get_scanline
xvt_image_set_pixel
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_image_create`.

xvt_image_get_resolution

Get the Horizontal and Vertical Resolution of an Image

Summary

```
void xvt_image_get_resolution(XVT_IMAGE image,
     long *hresp, long *vresp)
```

`XVT_IMAGE image`

Image whose resolution is to be received.

`long *hresp`

Pointer to the horizontal resolution value.

`long *vresp`

Pointer to the vertical resolution value.

Description

This function gets the horizontal and vertical resolution of an image in dots per inch (DPI).

Return value

Zero.

Parameter Validity and Error Conditions

XVT issues an error if `image` is not valid, or if the `hresp` or `vresp` pointers are `NULL`.

Implementation Note

Currently this function is not operational. It is included for future enhancement.

See Also

`XVT_IMAGE`
`xvt_image_set_resolution`

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

xvt_image_get_scanline

Get a Pointer to a Scanline in an Image

Summary

```
DATA_PTR xvt_image_get_scanline(XVT_IMAGE image,
     short linenum)
```

`XVT_IMAGE image`

Image containing the scanline.

`short linenum`

Row number of the scanline.

Description

This function returns a pointer to a row of pixel data in the image. Given this pointer, your application can directly manipulate the pixels in an image.

Note: Do not assume that consecutive scanlines are contiguous in memory. Also, do not make any assumptions about the presence or absence of padding at the ends of scanlines.

Caution: Do not attempt to free the memory pointer retrieved to by this function.

Return Value

The scanline data pointed to by the address returned by this function is interpreted differently, depending on the color format of the image:

`XVT_IMAGE_MONO`

Data consists of one-bit pixels, packed eight pixels per byte. The MSB of each byte is the leftmost of the eight pixels. Bytes of eight pixels are arranged from left to right in memory, with the leftmost byte first.

`XVT_IMAGE_CL8`

Data consists of 8-bit pixels, which are indices for the color look-up table. Pixels are arranged from left to right in memory, with the leftmost pixel first.

`XVT_IMAGE_RGB`

Data consists of 24-bit `COLOR` values, one per pixel. Cast the address returned by this function into (`COLOR *`). Pixels are arranged from left to right in memory, with the leftmost pixel first.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `image` is `NULL` or invalid
- `linenum` is less than zero or greater than (image height - 1)

See Also

```
DATA_PTR
XVT_IMAGE
XVT_IMAGE_FORMAT
XVT_IMAGE_* Values for XVT_IMAGE_FORMAT
xvt_image_get_pixel
xvt_image_set_pixel
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

xvt_image_read_*

Image Read Functions

```
xvt_image_read
xvt_image_read_bmp
xvt_image_read_bmp_from_iostr
xvt_image_read_gif
xvt_image_read_gif_from_iostr
xvt_image_read_jpg
xvt_image_read_jpg_from_iostr
xvt_image_read_macpict
xvt_image_read_macpict_from_iostr
xvt_image_read_xbm
xvt_image_read_xbm_from_iostr
xvt_image_read_xpm
xvt_image_read_xpm_from_iostr
```

xvt_image_read

Read an Image from a File

Summary

```
XVT_IMAGE xvt_image_read(char *filenamep)

char *filenamep
```

Image's filename.

Description

This function returns an image read from a file. `xvt_image_read` attempts to determine the type of image stored in the file by opening it and reading the first few bytes of the file. If a match with a known image type is found, the appropriate `xvt_image_read_*` function is called to read the image file.

Return Value

The image in a `XVT_IMAGE` variable if successful; `NULL` on error.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- The file type cannot be discerned from the contents of the file
- The file cannot be opened
- `filename` is `NULL`
- There is insufficient memory to contain the image

See Also

```
XVT_IMAGE
xvt_image_create
xvt_image_destroy
xvt_image_read_*
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

```
XVT_IMAGE image;
char *filename;
...
image = xvt_image_read(filename);
if (image == NULL_IMAGE) {
    /* image read error */
    ...
} else {
    /* valid image */
    ...
}
```

xvt_image_read_bmp

Create an Image Read from a Named BMP File

Summary

```
XVT_IMAGE xvt_image_read_bmp(char *filenamep)

char *filenamep
```

The **.BMP** filename.

Description

This function opens the file specified by `filenamep` and checks that it is a valid Win32 BMP file. If valid, it creates an `XVT_IMAGE` of the appropriate format and size and reads the contents of the BMP file into the `XVT_IMAGE`. It then closes the file and returns the handle to the `XVT_IMAGE`.

Return Value

The image, in a `XVT_IMAGE` variable if successful. The value of the `XVT_IMAGE` handle is `NULL_IMAGE` if there is an error during the procedure.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `filenamep` is `NULL`
- The file cannot be opened
- The file is not a valid Win32 BMP file
- There is insufficient memory to contain the image

Implementation Note

BMP file data is represented by the Win32 format. The XVT image-read BMP functions support this format. BMP image data is represented by one of four image formats based on the number of bits to store each pixel. The BMP image formats are 1-bit, 4-bits, 8-bits, and 24-bits per pixel. The mapping of BMP image format to `XVT_IMAGE` format is as follows:

BMP format	XVT_IMAGE_FORMAT
1-bit	`XVT_IMAGE_MONO`
4-bits	`XVT_IMAGE_CL8`
8-bits	`XVT_IMAGE_CL8`
24-bits	`XVT_IMAGE_RGB`

See Also

```
XVT_IMAGE
XVT_IMAGE_FORMAT
XVT_IMAGE_* Values for XVT_IMAGE_FORMAT
xvt_image_create
xvt_image_destroy
xvt_image_read_bmp_from_iostr
xvt_image_write_bmp_to_iostr
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

If the variable `filenamep` is a pathname to a valid BMP file, this code fragment creates an `XVT_IMAGE`. It calls the `xvt_image_read_bmp` function and checks the `XVT_IMAGE` handle for an error.

```
XVT_IMAGE image;
char *filenamep;
image = xvt_image_read_bmp(filenamep);
if (image == NULL_IMAGE) {
    /* image read error */
    ...
    return;
}
/* valid image */
```

xvt_image_read_bmp_from_iostr

Create an Image Read from an Input Stream of BMP Data

Summary

```
void xvt_image_read_bmp_from_iostr(XVT_IOSTREAM iostr)
```

`XVT_IOSTREAM iostr`

Input stream.

Description

This function checks that `XVT_IOSTREAM` is initialized for reading and positioned to the beginning of valid Win32 BMP data.

If valid, it creates an `XVT_IMAGE` of the appropriate format and size, and reads the BMP data from the `XVT_IOSTREAM` into the `XVT_IMAGE`. It then returns the handle to the `XVT_IMAGE`.

Return Value

An `XVT_IMAGE` handle if successful. The value of the `XVT_IMAGE` handle is `NULL_IMAGE` if there is an error during the procedure.

Parameter Validity and Error Conditions

`XVT issues an error if any of the following conditions are true:`

- iostr is `NULL`
- `iostr` does not reference valid Win32 BMP data
- There is insufficient memory to contain the image

Implementation Note

See `xvt_image_read_bmp`.

See Also

```
XVT_IMAGE
XVT_IOSTREAM
xvt_image_create
xvt_image_destroy
xvt_image_read_bmp
xvt_image_write_bmp_to_iostr
xvt_iostr_create_fread
xvt_iostr_create_read
xvt_iostr_destroy
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

If the variable `filenamep` is a pathname to a valid BMP file, this code fragment creates an `XVT_IMAGE`. It opens the specified file for reading, and then creates an `XVT_IOSTREAM` with default functions set up to read an open file. It then calls the `xvt_image_read_bmp_from_iostr` function, destroys the `XVT_IOSTREAM`, and closes the file. The `XVT_IMAGE` handle is then checked for an error; this error could be a file open error, an I/O stream create error, or an image read error.

```
XVT_IMAGE image;
char *filenamep;
FILE *filep;
XVT_IOSTREAM iostr;
image= NULL_IMAGE;
filep = fopen(filenamep, "rb");
if (filep != NULL) {
    iostr = xvt_iostr_create_fread(filep);
    if (iostr != NULL) {
    image = xvt_image_read_bmp_from_iostr(iostr);
    xvt_iostr_destroy(iostr);
    }
    fclose(filep);
}
if (image == NULL_IMAGE) {
    /* file open, iostr create or image read error */
    return;
}
/* valid image */
```

xvt_image_read_gif

Create an Image Read from a Named GIF File

Summary

```
XVT_IMAGE xvt_image_read_gif(char *filenamep)
```

`char *filenamep`

The **.GIF** filename.

Description

This function opens the file specified by `filenamep` and checks that it is a valid GIF file. If valid, it creates an `XVT_IMAGE` of the appropriate format and size and reads the contents of the GIF file

into the XVT_IMAGE. It then closes the file and returns the handle to the XVT_IMAGE.

Return Value

The image, in a XVT_IMAGE variable if successful. The value of the XVT_IMAGE handle is NULL_IMAGE if there is an error during the procedure.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- filenamep is NULL
- The file cannot be opened
- The file is not a valid GIF file
- There is insufficient memory to contain the image

Implementation Note

GIF image data is represented by one of three image formats based on the number of bits to store each pixel. The GIF image formats are 1-bit, 4-bits, and 8-bits per pixel. The mapping of GIF image format to XVT_IMAGE format is as follows:

BMP format	XVT_IMAGE_FORMAT
1-bit	XVT_IMAGE_MONO
4-bits	XVT_IMAGE_CL8
8-bits	XVT_IMAGE_CL8

See Also

```
XVT_IMAGE
XVT_IMAGE_FORMAT
XVT_IMAGE_* Values for XVT_IMAGE_FORMAT
xvt_image_create
xvt_image_destroy
xvt_image_read_gif_from_iostr
xvt_image_write_gif_to_iostr
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

If the variable filenamep is a pathname to a valid GIF file, this code fragment creates an XVT_IMAGE. It calls the xvt_image_read_gif function and checks the XVT_IMAGE handle for an error.

```
XVT_IMAGE image;
char *filenamep;
image = xvt_image_read_gif(filenamep);
if (image == NULL_IMAGE) {
    /* image read error */
    ...
    return;
}
/* valid image */
```

xvt_image_read_gif_from_iostr

Create an Image Read from an Input Stream of GIF Data

Summary

```
void xvt_image_read_gif_from_iostr(XVT_IOSTREAM iostr)
```

`XVT_IOSTREAM iostr`

Input stream.

Description

This function checks that `XVT_IOSTREAM` is initialized for reading and positioned to the beginning of valid GIF data.

If valid, it creates an `XVT_IMAGE` of the appropriate format and size, and reads the GIF data from the `XVT_IOSTREAM` into the `XVT_IMAGE`. It then returns the handle to the `XVT_IMAGE`.

Return Value

An `XVT_IMAGE` handle if successful. The value of the `XVT_IMAGE` handle is `NULL_IMAGE` if there is an error during the procedure.

Parameter Validity and Error Conditions

`XVT issues an error if any of the following conditions are true:`

- iostr is `NULL`
- `iostr` does not reference valid GIF data
- There is insufficient memory to contain the image

Implementation Note

See `xvt_image_read_gif`.

See Also

XVT_IMAGE
XVT_IOSTREAM
xvt_image_create
xvt_image_destroy
xvt_image_read_gif
xvt_image_write_gif_to_iostr
xvt_iostr_create_fread
xvt_iostr_create_read
xvt_iostr_destroy

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

If the variable `filenamep` is a pathname to a valid GIF file, this code fragment creates an `XVT_IMAGE`. It opens the specified file for reading, and then creates an `XVT_IOSTREAM` with default functions set up to read an open file. It then calls the `xvt_image_read_gif_from_iostr` function, destroys the `XVT_IOSTREAM`, and closes the file. The `XVT_IMAGE` handle is then checked for an error; this error could be a file open error, an I/O stream create error, or an image read error.

```
XVT_IMAGE image;
char *filenamep;
FILE *filep;
XVT_IOSTREAM iostr;
image= NULL_IMAGE;
filep = fopen(filenamep, "rb");
if (filep != NULL) {
    iostr = xvt_iostr_create_fread(filep);
    if (iostr != NULL) {
    image = xvt_image_read_bmp_from_iostr(iostr);
    xvt_iostr_destroy(iostr);
    }
    fclose(filep);
}
if (image == NULL_IMAGE) {
    /* file open, iostr create or image read error */
    return;
}
/* valid image */
```

xvt_image_read_jpg

Create an Image Read from a Named JPEG File

Summary

```
XVT_IMAGE xvt_image_read_jpg(char *filenamep)

char *filenamep
```

The **.JPG** filename.

Description

This function opens the file specified by `filenamep` and checks that it is a valid JPEG file. If valid, it creates an `XVT_IMAGE` of the appropriate format and size and reads the contents of the JPEG file into the `XVT_IMAGE`. It then closes the file and returns the handle to the `XVT_IMAGE`.

Return Value

The image, in a `XVT_IMAGE` variable if successful. The value of the `XVT_IMAGE` handle is `NULL_IMAGE` if there is an error during the procedure.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `filenamep` is `NULL`
- The file cannot be opened
- The file is not a valid JPEG file
- There is insufficient memory to contain the image

Implementation Note

JPEG image data is represented by one of four image formats based on the number of bits to store each pixel. The JPEG image formats are 1-bit, 4-bits, 8-bits, and 24-bits per pixel. The mapping of JPEG image format to `XVT_IMAGE` format is as follows:

BMP format	XVT_IMAGE_FORMAT
1-bit	XVT_IMAGE_MONO
4-bits	XVT_IMAGE_CL8
8-bits	XVT_IMAGE_CL8
24-bits	XVT_IMAGE_RGB

See Also

```
XVT_IMAGE
XVT_IMAGE_FORMAT
XVT_IMAGE_* Values for XVT_IMAGE_FORMAT
xvt_image_create
xvt_image_destroy
xvt_image_read_jpg_from_iostr
xvt_image_write_jpg_to_iostr
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

If the variable `filenamep` is a pathname to a valid JPEG file, this code fragment creates an `XVT_IMAGE`. It calls the `xvt_image_read_jpg` function and checks the `XVT_IMAGE` handle for an error.

```
XVT_IMAGE image;
char *filenamep;
image = xvt_image_read_jpg(filenamep);
if (image == NULL_IMAGE) {
    /* image read error */
    ...
    return;
}
/* valid image */
```

xvt_image_read_jpg_from_iostr

Create an Image Read from an Input Stream of JPEG Data

Summary

```
void xvt_image_read_jpg_from_iostr(XVT_IOSTREAM iostr)
```

`XVT_IOSTREAM iostr`

Input stream.

Description

This function checks that `XVT_IOSTREAM` is initialized for reading and positioned to the beginning of valid JPEG data.

If valid, it creates an `XVT_IMAGE` of the appropriate format and size, and reads the JPEG data from the `XVT_IOSTREAM` into the `XVT_IMAGE`. It then returns the handle to the `XVT_IMAGE`.

Return Value

An XVT_IMAGE handle if successful. The value of the XVT_IMAGE handle is NULL_IMAGE if there is an error during the procedure.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- iostr is NULL
- iostr does not reference valid JPEG data
- There is insufficient memory to contain the image

Implementation Note

See xvt_image_read_jpg.

See Also

XVT_IMAGE
XVT_IOSTREAM
xvt_image_create
xvt_image_destroy
xvt_image_read_jpg
xvt_image_write_jpg_to_iostr
xvt_iostr_create_fread
xvt_iostr_create_read
xvt_iostr_destroy

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

If the variable filenamep is a pathname to a valid JPEG file, this code fragment creates an XVT_IMAGE. It opens the specified file for reading, and then creates an XVT_IOSTREAM with default functions set up to read an open file. It then calls the xvt_image_read_jpg_from_iostr function, destroys the XVT_IOSTREAM, and closes the file. The XVT_IMAGE handle is then checked for an error; this error could be a file open error, an I/O stream create error, or an image read error.

```
XVT_IMAGE image;
char *filenamep;
FILE *filep;
XVT_IOSTREAM iostr;
image= NULL_IMAGE;
filep = fopen(filenamep, "rb");
if (filep != NULL) {
    iostr = xvt_iostr_create_fread(filep);
    if (iostr != NULL) {
    image = xvt_image_read_jpg_from_iostr(iostr);
    xvt_iostr_destroy(iostr);
    }
    fclose(filep);
}
if (image == NULL_IMAGE) {
    /* file open, iostr create or image read error */
    return;
}
/* valid image */
```

xvt_image_read_macpict

Create an Image Read from a Named Macintosh PICT File

Summary

```
XVT_IMAGE xvt_image_read_macpict(char *filenamep)

char *filenamep
```

PICT's filename.

Description

This function opens the file specified by `filenamep` and checks that it is a valid Macintosh PICT file. If valid, it creates an `XVT_IMAGE` of the appropriate format and size and reads the contents of the Macintosh PICT file into the `XVT_IMAGE`. It then closes the file and returns the handle to the `XVT_IMAGE`.

Return Value

An `XVT_IMAGE` handle. The value of the `XVT_IMAGE` handle is `NULL_IMAGE` if there is an error during the procedure.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `filenamep` is `NULL`
- The file cannot be opened

- The file is not a valid Macintosh PICT file
- There is insufficient memory to contain the image

Implementation Note

This function is available only in XVT/Mac. On other platforms, this function has no effect and returns `NULL`.

See Also

```
XVT_IMAGE
xvt_image_create
xvt_image_destroy
xvt_image_read_bmp
xvt_image_read_macpict_from_iostr
```

The "Portable Images" Chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_image_read_bmp`.

xvt_image_read_macpict_from_iostr

Create an Image Read from an Input Stream of Macintosh PICT Data

Summary

```
XVT_IMAGE xvt_image_read_macpict_from_iostr
     (XVT_IOSTREAM iostr)

XVT_IOSTREAM iostr
```

Input stream.

Description

This function checks that `XVT_IOSTREAM` is initialized for reading and positioned to the beginning of a valid Macintosh PICT data structure. If valid, it creates an `XVT_IMAGE` of the appropriate format and size and reads the Macintosh PICT data structure from the `XVT_IOSTREAM` into the `XVT_IMAGE`. It then returns the handle to the `XVT_IMAGE`.

Return Value

An `XVT_IMAGE` handle if successful. The value of the `XVT_IMAGE` handle is `NULL_IMAGE` if there is an error during the procedure.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `iostr` is `NULL`
- `iostr` does not reference valid Macintosh PICT data
- There is insufficient memory to contain the image

Implementation Note

This function is available only in XVT/Mac.

The input data stream must be in the following format (i.e., the Macintosh PICT format):

- A 512-byte header
- 2 bytes containing the picture size
- 8 bytes containing the picture frame (rectangle)
- `n` bytes containing the picture description data

See Also

```
XVT_IMAGE
XVT_IOSTREAM
xvt_image_create
xvt_image_destroy
xvt_image_read_bmp_from_iostr
xvt_image_read_macpict
xvt_iostr_create_fread
xvt_iostr_create_read
xvt_iostr_destroy
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_image_read_bmp_from_iostr`.

xvt_image_read_xbm

Create an Image Read from a Named XBM File

Summary

```
XVT_IMAGE xvt_image_read_xbm(char *filenamep)
```

`char *filenamep`

XBM’s filename.

Description

This function opens the file specified by `filenamep` and checks that it is a valid XBM file. If valid, it creates an `XVT_IMAGE` of the appropriate format and size and reads the contents of the XBM file into the `XVT_IMAGE`. It then closes the file and returns the handle to the `XVT_IMAGE`.

Return Value

An `XVT_IMAGE` handle if successful. The value of the `XVT_IMAGE` handle is `NULL_IMAGE` if there is an error during the procedure.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `filenamep` is `NULL`
- The file cannot be opened
- The file is not a valid XBM file
- There is insufficient memory to contain the image

See Also

```
XVT_IMAGE
xvt_image_create
xvt_image_destroy
xvt_image_read_bmp
xvt_image_read_xbm_from_iostr
xvt_image_read_xpm
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_image_read_bmp`.

---

xvt_image_read_xbm_from_iostr

Create an Image Read from an Input Stream of XBM Data

Summary

```
XVT_IMAGE xvt_image_read_xbm_from_iostr
     (XVT_IOSTREAM iostr)

XVT_IOSTREAM iostr
```

Input stream.

Description

This function checks that XVT_IOSTREAM is initialized for reading and positioned to the beginning of valid XBM data. If valid, it creates an XVT_IMAGE of the appropriate format and size and reads the XBM data from the XVT_IOSTREAM into the XVT_IMAGE. It then returns the handle to the XVT_IMAGE.

Return Value

An XVT_IMAGE handle if successful. The value of the XVT_IMAGE handle is NULL_IMAGE if there is an error during the procedure.

Parameter Validity and Error Conditions

XVT issues an error if any the following conditions are true:

- iostr is NULL
- iostr does not reference valid XBM data
- There is insufficient memory to contain the image

Implementation Note

See xvt_image_read_xbm.

See Also

XVT_IMAGE
XVT_IOSTREAM
xvt_image_create
xvt_image_destroy
xvt_image_read_bmp_from_iostr
xvt_image_read_xbm
xvt_image_read_xpm_from_iostr
xvt_iostr_create_fread
xvt_iostr_create_read
xvt_iostr_destroy

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for xvt_image_read_bmp_from_iostr.

xvt_image_read_xpm

Create an Image from a Named XPM File

Summary

```
XVT_IMAGE xvt_image_read_xpm(char *filenamep)

char *filenamep
```

XPM's filename.

Description

This function opens the file specified by `filenamep` and checks that it is a valid XPM file. If valid, it creates an `XVT_IMAGE` of the appropriate format and size, and reads the contents of the XPM file into the `XVT_IMAGE`. It then closes the file and returns the handle to the `XVT_IMAGE`.

Return Value

An `XVT_IMAGE` handle if successful. The value of the `XVT_IMAGE` handle is `NULL_IMAGE` if there is an error during the procedure.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `filenamep` is `NULL`
- The file cannot be opened
- The file is not a valid XPM file
- There is insufficient memory to contain the image

See Also

```
XVT_IMAGE
xvt_image_create
xvt_image_destroy
xvt_image_read_bmp
xvt_image_read_xbm
xvt_image_read_xpm_from_iostr
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_image_read_bmp`.

xvt_image_read_xpm_from_iostr

Create an Image Read from an Input Stream of XPM Data

Summary

```
XVT_IMAGE xvt_image_read_xpm_from_iostr
     (XVT_IOSTREAM iostr)

XVT_IOSTREAM iostr
```

Input stream.

Description

This function checks that `XVT_IOSTREAM` is initialized for reading and positioned to the beginning of valid XPM data. If valid, it creates an `XVT_IMAGE` of the appropriate format and size, and reads the XPM data from the `XVT_IOSTREAM` into the `XVT_IMAGE`. It then returns the handle to the `XVT_IMAGE`.

Return Value

An `XVT_IMAGE` handle if successful. The value of the `XVT_IMAGE` handle is `NULL_IMAGE` if there is an error during the procedure.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `iostr` is `NULL`
- `iostr` does not reference valid XPM data
- There is insufficient memory to contain the image

Implementation Note

See `xvt_image_read_xpm`.

See Also

```
XVT_IMAGE
XVT_IOSTREAM
xvt_image_create
xvt_image_destroy
xvt_image_read_bmp_from_iostr
xvt_image_read_xbm_from_iostr
xvt_image_read_xpm
xvt_iostr_create_fread
xvt_iostr_create_read
xvt_iostr_destroy
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for xvt_image_read_bmp_from_iostr.

xvt_image_set_clut

Set a Color Look-up Table Entry

Summary

```
void xvt_image_set_clut(XVT_IMAGE image, short index,
     COLOR color)
```

XVT_IMAGE image

Image that owns the look-up table to alter.

short index

Index of the look-up table entry to set.

COLOR color

Color value to set the look-up table entry.

Description

This function sets the COLOR value of an entry in a color look-up table. index must be between 0 and 255 (inclusive) for XVT_IMAGE_CL8 or between 0 and 1 (inclusive) for XVT_IMAGE_MONO. The function does not alter the number of colors used by the image, nor does it alter the image itself. To change the number of colors used by the image, use xvt_image_set_ncolors.

This function is useful only with images of format XVT_IMAGE_CL8 or XVT_IMAGE_MONO, since other formats do not have look-up tables.

Return Value

None.

Parameter Validity and Error Conditions

XVT issues an error if image is not valid. This function has no effect if the format of image is not XVT_IMAGE_CL8, or if index is not in the range [0, 255] for XVT_IMAGE_CL8 or [0, 1] for XVT_IMAGE_MONO.

See Also

```
COLOR
XVT_IMAGE
XVT_IMAGE_* Values for XVT_IMAGE_FORMAT
xvt_image_create
xvt_image_get_clut
xvt_image_get_pixel
xvt_image_set_ncolors
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_image_create`.

---

xvt_image_set_ncolors

Set the Number of Colors Used by an Image

Summary

```
void xvt_image_set_ncolors(XVT_IMAGE image,
     short ncolors)
```

`XVT_IMAGE image`

Image whose color look-up table is to be altered.

`short ncolors`

Value to set the number of colors.

Description

This function sets the number of colors used by an image with color format `XVT_IMAGE_CL8`. The number set by this function affects how colors are mapped when the image is transferred to a window or pixmap, or from a pixmap. The color-mapping process constructs color tables to map colors to/from the image's colors, but only for the number of colors set by this function.

Return Value

None.

Parameter Validity and Error Conditions

XVT issues an error if `image` is not valid. This function has no effect if the format of `image` is not `XVT_IMAGE_CL8`.

If `ncolors` is greater than 256, it will be treated as if it is 256. If `ncolors` is less than 2, it will be treated as if it is 2.

See Also

```
XVT_IMAGE
XVT_IMAGE_* Values for XVT_IMAGE_FORMAT
xvt_dwin_draw_image
xvt_image_create
xvt_image_get_ncolors
xvt_image_set_clut
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_image_create`.

xvt_image_set_pixel

Set the Value of a Pixel In an Image

Summary

```
void xvt_image_set_pixel(XVT_IMAGE image, short x,
     short y, COLOR color)
```

`XVT_IMAGE image`

Image containing the pixel to set.

`short x`

Horizontal coordinate of the pixel.

`short y`

Vertical coordinate of the pixel.

`COLOR color`

Color value to set the pixel.

Description

This function sets the color value of a pixel in an image. Using this function is more convenient than manipulating the pixels directly (using `xvt_image_get_scanline`), since it handles the arithmetic for the array addressing and converting colors to different color formats.

Return Value

None.

Parameter Validity and Error Conditions

XVT issues an error if `image` is not valid, or if the coordinate is outside the range of the dimensions.

Implementation Note

This function is slow when the image uses the `XVT_IMAGE_CL8` color format, since it performs a best-fit calculation comparing `color` to the entries in the image's color look-up table. For faster performance on images of this format, use `xvt_image_get_scanline`.

See Also

```
COLOR
XVT_IMAGE
XVT_IMAGE_* Values for XVT_IMAGE_FORMAT
xvt_image_create
xvt_image_get_pixel
xvt_image_get_scanline
xvt_image_set_clut
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_image_create`.

---

xvt_image_set_resolution

Set the Horizontal and Vertical Resolution of an Image

Summary

```
void xvt_image_set_resolution(XVT_IMAGE image,
     long hres, long vres)
```

`XVT_IMAGE image`

Image whose resolution is to be set.

`long hres`

Horizontal resolution value.

`long vres`

Vertical resolution value.

Description

This function sets the horizontal and vertical resolution of an image in dots per inch (dpi).

Return value

None.

Parameter Validity and Error Conditions

XVT issues an error if `image` is not valid.

Implementation Note

Currently this function is not operational. It is included for future enhancement.

See Also

`XVT_IMAGE`
`xvt_image_get_resolution`

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

xvt_image_transfer

Copy a Portion of One Image to Another Image

Summary

```
void xvt_image_transfer(XVT_IMAGE dstimage,
     XVT_IMAGE srcimage, RCT *dstrctp, RCT *srcrctp)
```

`XVT_IMAGE dstimage`

Destination image.

`XVT_IMAGE srcimage`

Source image.

`RCT *dstrctp`

Pointer to a rectangle that delimits the destination region. If this rectangle is empty, no image data is transferred.

`RCT *srcrctp`

Pointer to a rectangle that delimits the source region.

Description

This function copies the contents of a rectangular region in the source image into a rectangular region in the destination image.

If `*srcrctp` and `*dstrctp` are not congruent, this function translates and scales the source region as necessary to fit it into the destination rectangle. Any parts of the source or destination rectangles that fall outside of the bounds of their respective containers are ignored.

To copy the entire source image, use the rectangle (`0`, `0`, `width`, `height`) for the source rectangle, where width and height are the dimensions of the source image. To fill the entire destination image, use a similar rectangle for the destination rectangle. In this case, width and height are the dimensions of the destination image.

Return Value

None.

Parameter Validity and Error Conditions

If either `srcimage` or `dstimage` is not valid, if either is `NULL`, or if the source rectangle points to an empty rectangle, XVT issues an error. Any parts of the source or destination rectangles that fall outside of the bounds of their respective images are ignored.

See Also

```
RCT
XVT_IMAGE
xvt_dwin_draw_image
xvt_image_get_dimensions
xvt_image_get_from_pmap
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_image_create`.

xvt_image_write_bmp_to_iostr

Write an Image in MS-Windows BMP Format to an I/O Stream

Summary

```
BOOLEAN xvt_image_write_winbmp_to_iostr
     (XVT_IMAGE image, XVT_IOSTREAM iostr)
```

`XVT_IMAGE image`

A handle to the source image.

`XVT_IOSTREAM iostr`

The output stream.

Description

This function writes an `XVT_IMAGE` to an `XVT_IOSTREAM` in Win32 BMP format.

Prior to calling this function, the `XVT_IOSTREAM` must be correctly positioned and initialized for writing.

Return Value

A `BOOLEAN TRUE` if the procedure is successful; `FALSE` if there is an error during the procedure.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `iostr` is `NULL`
- `image` is `NULL` or invalid

See Also

```
XVT_IMAGE
XVT_IOSTREAM
xvt_image_create
xvt_image_destroy
xvt_image_read_bmp_from_iostr
xvt_image_read_xbm
xvt_iostr_create_fwrite
xvt_iostr_create_write
xvt_iostr_destroy
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

This code fragment writes an `XVT_IMAGE` to the file specified by the pathname `filenamep`. It opens the specified file for writing, and then creates an `XVT_IOSTREAM` with default functions setup up to write to an open file. It then calls the `xvt_image_write_bmp_to_iostr` function, destroys the `XVT_IOSTREAM`, and closes the file. The return value `ret` is then checked for an error; this error could be a file open error, an I/O stream create error, or an image write error.

```
XVT_IMAGE image;
char *filenamep;
BOOLEAN ret;
FILE * filep;
XVT_IOSTREAM iostr;ret = FALSE;filep = fopen(filenamep,
"wb");
if (filep != NULL) {
    iostr = xvt_iostr_create_fwrite(filep);
    if (iostr != NULL) {
        ret = xvt_image_write_winbmp_to_iostr(iostr,
            image);
        xvt_iostr_destroy(iostr);
    }
    fclose(filep);
}
if (!ret) {
    /* file open, iostr create or image write error */
    return;
}/* write succeeded */
```

xvt_image_write_macpict_to_iostr

Write an Image in Macintosh PICT Format to an Output Stream

Summary

```
BOOLEAN xvt_image_write_macpict_to_iostr
     (XVT_IMAGE image, XVT_IOSTREAM iostr)
```

`XVT_IMAGE image`

Handle to the source image.

`XVT_IOSTREAM iostr`

Output stream.

Description

This function writes an `XVT_IMAGE` to an `XVT_IOSTREAM` in Macintosh PICT format. Prior to calling this function, the `XVT_IOSTREAM` must have been initialized for writing and correctly positioned.

Return Value

A `BOOLEAN`. `TRUE` if the procedure is successful; `FALSE` if there was any error during the procedure.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `iostr` is `NULL`
- `image` is `NULL` or invalid

Implementation Note

This function is available only in XVT/Mac. The data is written in the following format (i.e., the Macintosh PICT format):

- A 512-byte header (containing all zeros)
- 2 bytes containing the picture size
- 8 bytes containing the picture frame (rectangle)
- n bytes containing the picture description data

See Also

```
XVT_IMAGE
XVT_IOSTREAM
xvt_image_create
xvt_image_destroy
xvt_image_read_macpict
xvt_image_read_macpict_from_iostr
xvt_image_write_bmp_to_iostr
xvt_iostr_create_fwrite
xvt_iostr_create_write
xvt_iostr_destroy
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_image_write_bmp_to_iostr`.

xvt_iostr_*

Input/Output Byte Stream Functions

```
xvt_iostr_create_fread
xvt_iostr_create_fwrite
xvt_iostr_create_read
xvt_iostr_create_write
xvt_iostr_destroy
xvt_iostr_get_context
```

xvt_iostr_create_fread

Create an I/O Stream for Reading Data from a File

Summary

```
XVT_IOSTREAM xvt_iostr_create_fread(FILE *fp)

FILE *fp
```

File pointer stream context for an open file from which to read.

Description

This function creates an I/O stream object for reading data from a file. `fp` must be an open file pointer. Data will be read starting at the current file position.

Return Value

The `XVT_IOSTREAM` object; `NULL` on insufficient memory or error (which invokes XVT error processing).

See Also

```
XVT_IOSTREAM
xvt_image_read_bmp_from_iostr
xvt_image_read_macpict_from_iostr
xvt_image_read_xbm_from_iostr
xvt_image_read_xpm_from_iostr
xvt_iostr_create_fwrite
xvt_iostr_create_read
```

Example

See the example for `xvt_image_read_bmp_from_iostr`.

xvt_iostr_create_fwrite

Create an I/O Stream for Writing Data to a File

Summary

```
XVT_IOSTREAM xvt_iostr_create_fwrite(FILE *fp)

FILE *fp
```

File pointer stream context for an open file to which to write.

Description

This function creates an I/O stream object for writing data to a file. `fp` must be an open file pointer. Data will be written starting at the current file position.

Return Value

The `XVT_IOSTREAM` object; `NULL` on insufficient memory or error (which invokes XVT error processing).

See Also

`XVT_IOSTREAM`
`xvt_image_read_bmp_from_iostr`
`xvt_image_read_macpict_from_iostr`
`xvt_image_read_xbm_from_iostr`
`xvt_image_read_xpm_from_iostr`
`xvt_image_write_bmp_to_iostr`
`xvt_iostr_create_read`
`xvt_iostr_create_write`

Example

See the example for `xvt_image_write_bmp_to_iostr`.

xvt_iostr_create_read

Create an I/O Stream for Reading Data

Summary

```
XVT_IOSTREAM xvt_iostr_create_read
     (XVT_IOSTR_CONTEXT context,
    XVT_IOSTR_RWFUNC get_bytes,
    XVT_IOSTR_SZFUNC num_bytes)

XVT_IOSTR_CONTEXT context
```

Context for the I/O stream object. Typically a file pointer or a pointer to application data memory representing the input stream.

```
XVT_IOSTR_RWFUNC get_bytes
```

User-supplied function for reading bytes.

```
XVT_IOSTR_SZFUNC num_bytes
```

User-supplied function for the size of the stream in bytes.

Description

This function creates an I/O stream object for reading data from an arbitrary source, such as a memory buffer. You must supply the functions `get_bytes` and `num_bytes`. The types are defined as follows:

```
    typedef short(* XVT_IOSTR_RWFUNC)
        (XVT_IOSTREAM iostr, unsigned short nbytes,
XVT_BYTE buf);
    typedef long(* XVT_IOSTR_SZFUNC)
        (XVT_IOSTREAM iostr);
```

Return Value

The `XVT_IOSTREAM` object; `NULL` on insufficient memory or error (which invokes XVT error processing).

See Also

XVT_BYTE
XVT_IOSTR_CONTEXT
XVT_IOSTR_RWFUNC
XVT_IOSTR_SZFUNC
xvt_image_read_bmp_from_iostr
xvt_image_read_macpict_from_iostr
xvt_image_read_xbm_from_iostr
xvt_image_read_xpm_from_iostr
xvt_iostr_create_write

xvt_iostr_create_write

Create an I/O Stream for Writing Data

Summary

```
XVT_IOSTREAM xvt_iostr_create_write
    (XVT_IOSTR_CONTEXT context,
    XVT_IOSTR_RWFUNC put_bytes)
```

`XVT_IOSTR_CONTEXT context`

Context for the I/O stream object. Typically, a file pointer, or a pointer to application data memory representing the output stream.

`XVT_IOSTR_RWFUNC put_bytes`

User-supplied function for writing bytes.

Description

This function creates an I/O stream object for writing data to an arbitrary destination, such as a memory buffer. You must supply the function `put_bytes`. The type is defined as follows:

```
typedef short(* XVT_IOSTR_RWFUNC)
    (XVT_IOSTREAM iostr, unsigned short nbytes,
        XVT_BYTE buf);
```

Return Value

The `XVT_IOSTREAM` object; `NULL` on insufficient memory or error (which invokes XVT error processing).

See Also

```
XVT_BYTE
XVT_IOSTR_CONTEXT
XVT_IOSTR_RWFUNC
XVT_IOSTR_SZFUNC
xvt_image_read_bmp_from_iostr
xvt_image_read_macpict_from_iostr
xvt_image_read_xbm_from_iostr
xvt_image_read_xpm_from_iostr
xvt_iostr_create_read
```

xvt_iostr_destroy

Destroy an I/O Stream Object

Summary

```
void xvt_iostr_destroy(XVT_IOSTREAM iostr)
```

`XVT_IOSTREAM iostr`

The I/O stream object to destroy.

Description

This function destroys an I/O stream object. To free the memory an object uses, call this function when you no longer need the object.

Return Value

None.

Parameter Validity and Error Conditions

XVT issues an error if `iostr` is not a valid `XVT_IOSTREAM`.

See Also

```
XVT_IOSTREAM
xvt_image_write_bmp_to_iostr
xvt_iostr_create_fread
xvt_iostr_create_fwrite
xvt_iostr_create_read
xvt_iostr_create_write
```

Example

See the example for `xvt_image_write_bmp_to_iostr`.

xvt_iostr_get_context

Returns the Context of a Stream Object

Summary

```
XVT_IOSTR_CONTEXT xvt_iostr_get_context(
    XVT_IOSTREAM iostream);
```

Description

This function returns the pointer to the data stream context of an `XVT_IOSTREAM` object. For file stream objects, the data context is a file pointer (`FILE *`).

See Also

```
XVT_IOSTR_CONTEXT
XVT_IOSTREAM
xvt_iostr_create_fread
xvt_iostr_create_fwrite
xvt_iostr_create_read
xvt_iostr_create_write
```

xvt_list_*

List Functions

```
xvt_list_add
xvt_list_clear
xvt_list_count_all
xvt_list_count_sel
xvt_list_get_all
xvt_list_get_elt
xvt_list_get_first_sel
xvt_list_get_sel
xvt_list_get_sel_index
xvt_list_is_sel
xvt_list_rem
xvt_list_resume
xvt_list_set_sel
xvt_list_suspend
```

xvt_list_add

Add String or Slist to a List Control

Summary

```
BOOLEAN xvt_list_add(WINDOW win, int index,
    char *sx)
```

`WINDOW win`

Window of control (`WC_LBOX`, `WC_LISTEDIT`, or `WC_LISTBUTTON`).

`int index`

Position to insert the string or `SLIST`.

`char *sx`

String or `SLIST` to add.

Description

This function adds a `NULL`-terminated string or `SLIST` specified by `sx` to the `WC_LBOX`, `WC_LISTEDIT`, or `WC_LISTBUTTON` control specified by `win`. For `WC_LISTEDIT` or `WC_LISTBUTTON` controls, only the "list" portion of the control is affected. The exception to this is that when the first item is added to an empty list for a `WC_LISTBUTTON`, the title of the button portion is set to the new item. The addition is placed before the item whose index is equal to `index` (origin 0). An `index` that's too large or -1 causes the addition to be at the end; for readability, a value of -1 is suggested.

`SLIST`s are linked lists of strings; they are explained in the "Utilities" chapter in the*XVT Portability Toolkit Guide*. When an `SLIST` is passed to `xvt_list_add`, the data is not used or saved by the list control.

If you're going to call `xvt_list_add` several times to add a group of strings, you should bracket the calls with `xvt_list_suspend` and `xvt_list_resume` to suppress unneeded updates.

If an application-defined format function is attached to the `WC_LISTBOX` control, calling `xvt_list_add` may result in inserting a different string or list of strings, as specified by the format function, into the list box contents or may cause `xvt_list_add` to return without taking any action.

Tip: List controls don't have a fixed limit on how many items they can hold, but they get sluggish when the number gets much over a hundred. Consider using a regular XVT window, which you can scroll yourself, for huge amounts of data.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are not met:

- `win` must be a `WC_LBOX`, `WC_LISTBUTTON`, or `WC_LISTEDIT` control
- `sx` is `NULL`
- You must *not* call this function during an `EUPDATE` event

Implementation Note

XVT/Mac has a system limitation of 32KB total data that can be stored in a `WC_LBOX` list.

See Also

```
SLIST
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
xvt_list_add
xvt_list_clear
xvt_list_resume
xvt_list_suspend
```

Example

This code adds available font families to a list box:

```
int f;
WINDOW listbox;
long num_families;
char *families[MAX_FAMILIES];
...
/* show available families in a listbox */
num_families = xvt_fmap_get_families(NULL, families,
     MAX_FAMILIES);
xvt_list_suspend(listbox);
xvt_list_clear(listbox);
for (f = 0; f < num_families; f++)
{
    xvt_list_add(listbox, f, families[f]);
    xvt_mem_free(families[f]);
}
xvt_list_resume(listbox);
```

xvt_list_clear

Clear List Control

Summary

```
BOOLEAN xvt_list_clear(WINDOW win)
```

`WINDOW win`

Window of control (`WC_LBOX`, `WC_LISTEDIT`, or `WC_LISTBUTTON`).

Description

This function clears all items from the `WC_LBOX`, `WC_LISTEDIT`, or `WC_LISTBUTTON` control specified by `win`. For `WC_LISTEDIT` controls, the edit portion of the control will also be cleared. For `WC_LISTBUTTON controls`, the button portion of the control will also be cleared.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are not met:

- `win` must be a `WC_LBOX`, `WC_LISTBUTTON`, or `WC_LISTEDIT` control
- You must *not* call this function during an `EUPDATE` event

See Also

```
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
xvt_list_add
```

Example

See the example for `xvt_list_add`.

xvt_list_count_all

Count Items in List Control

Summary

```
int xvt_list_count_all(WINDOW win)

WINDOW win
```

Window of control (WC_LBOX, WC_LISTEDIT, or WC_LISTBUTTON).

Description

This function counts the total number of items currently in the WC_LBOX, WC_LISTEDIT, or WC_LISTBUTTON control specified by win. For WC_LISTEDIT or WC_LISTBUTTON controls, only the items in the "list" portion of the control are counted.

Return Value

Number of items if successful; -1 if unsuccessful (on error).

Parameter Validity and Error Conditions

win must be a WC_LBOX, WC_LISTBUTTON, or WC_LISTEDIT control.

See Also

```
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
xvt_list_get_elt
```

Example

See the example for xvt_list_get_elt.

xvt_list_count_sel

Count Selected Items in List Control

Summary

```
int xvt_list_count_sel(WINDOW win)

WINDOW win
```

Window of control (WC_LBOX or WC_LISTBUTTON).

Description

This function counts the number of selected items in the WC_LBOX or WC_LISTBUTTON control specified by win. For WC_LISTBUTTON controls, only the items in the "list" portion of the control are counted.

WC_LISTEDIT doesn't allow any programmatic manipulation of the list selection, and cannot be used with this function.

Note: Always test the return result of this function to verify that there is a selection. This ensures that portable code is maintained.

Return Value

Number of items if successful; -1 if unsuccessful (on error).

Parameter Validity and Error Conditions

win must be a WC_LBOX or WC_LISTBUTTON control.

See Also

W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
xvt_list_get_sel

Example

See the example for xvt_list_get_sel.

---

xvt_list_get_all

Get All Items in List Control

Summary

```
SLIST xvt_list_get_all(WINDOW win)
```

WINDOW win

Window of control (WC_LBOX, WC_LISTEDIT, or WC_LISTBUTTON).

Description

This function gets all items, in the form of an SLIST, from the WC_LBOX, WC_LISTEDIT, or WC_LISTBUTTON control specified by win. For WC_LISTEDIT or WC_LISTBUTTON controls, only items in the "list" portion of the control are retreived.

The order of the elements in the SLIST is the same as the order of the items in the list control.

Each string in an SLIST is associated with a long data word; for this function each data word is the index of the corresponding item in the list box, not original data from an SLIST. Because this function gets all the items, the indices go from zero through the number of items minus one.

To retrieve the elements of the returned SLIST, you can call the xvt_slist_* functions. After you're done with the returned SLIST, it's your responsibility to free it with a call to xvt_slist_destroy.

Return Value

SLIST containing all items if successful; NULL if unsuccessful (on error).

Parameter Validity and Error Conditions

win must be a WC_LBOX, WC_LISTBUTTON, or WC_LISTEDIT control.

See Also

SLIST
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
xvt_list_get_all
xvt_slist_*
xvt_slist_destroy
xvt_slist_get
xvt_slist_get_elt
xvt_slist_get_first
xvt_slist_get_next

Example

This code writes the contents of a list box to a file. Note the call to xvt_slist_destroy at the end. For a similar example, see xvt_list_get_elt.

```
FILE *fp;
WINDOW lbox;
SLIST slist;
long dummy;
SLIST_ELT elt;
char *value;
...
slist = xvt_list_get_all(lbox);
for (elt = xvt_slist_get_first(slist); elt;
    elt = xvt_slist_get_next(slist, elt))
{
    value = xvt_slist_get(slist, elt, &dummy);
    if (value)
    fprintf(fp, "%s
    ", value);
}
xvt_slist_destroy(slist);
fclose(fp);
```

xvt_list_get_elt

Get Indexed Item in List Control

Summary

```
BOOLEAN xvt_list_get_elt(WINDOW win, int index,
     char *s, int sz_s)
```

`WINDOW win`

Window of control (`WC_LBOX`, `WC_LISTEDIT`, or `WC_LISTBUTTON`).

`int index`

Item to be gotten.

`char *s`

Buffer into which the text of the item is stored.

`int sz_s`

Maximum capacity of the buffer in bytes, including the `NULL`-terminator.

Description

This function gets the item whose index (origin 0) is equal to `index` from the `WC_LBOX`, `WC_LISTEDIT`, or `WC_LISTBUTTON` control specified by `win`. For `WC_LISTEDIT` or `WC_LISTBUTTON` controls, the item is retrieved from the "list" portion of the control.

The text of the item is stored in the buffer pointed to by `s`, whose capacity in bytes (including the `NULL`-terminator) is `sz_s`. Only as many bytes as will fit are copied to `s`.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are not met:

- `win` must be a `WC_LBOX`, `WC_LISTBUTTON`, or `WC_LISTEDIT` control
- `index` must be greater than or equal to than zero

See Also

`W_*, WC_*, WD_*, Values for WIN_TYPE`
`WINDOW`
`xvt_list_count_all`

Example

This code writes the contents of a list box to a file. Selected items are enclosed in brackets. For a similar example, see

```
xvt_list_get_all.
FILE *fp;
WINDOW lbox;
char value[MAX_SIZE];
int i, count;
...
count = xvt_list_count_all(lbox);
for (i = 0; i < count; i++)
    if (xvt_list_get_elt(lbox, i, value, MAX_SIZE)){
    if (xvt_list_is_sel(lbox, i))
    fprintf(fp, "[%s]
    ", value);
    else
    fprintf(fp, " %s
    ", value);
    }
fclose(fp);
```

xvt_list_get_first_sel

Get First Selected Item in List Control

Summary

```
BOOLEAN xvt_list_get_first_sel(WINDOW win, char *s,
     int sz_s)
```

`WINDOW win`

Window of control (`WC_LBOX` or `WC_LISTBUTTON`).

`char *s`

Buffer into which the text is stored.

`int sz_s`

Maximum capacity of the buffer in bytes.

Description

This function gets the first selected item from the `WC_LBOX` or `WC_LISTBUTTON` control specified by `win`. For `WC_LISTBUTTON` controls, the first selected item is retrieved from the "list" portion of the control.

The item's text is stored into the buffer pointed to by `s`, whose capacity in bytes (including `NULL`) is `sz_s`. Only as many bytes as will fit are copied to `s`.

If the list is a `WC_LBOX` that has been created with the "multiple select" flag, call the function `xvt_list_get_sel` instead.

`WC_LISTEDIT` doesn't allow any programmatic manipulation of the list selection, and cannot be used with this function.

Note: Always test the return result of this function to verify that there is a selection. This ensures that portablity is maintained.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful, on error, or if nothing is selected.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are not met:

- `win` must be a `WC_LBOX` or `WC_LISTBUTTON` control
- `s` must not be `NULL`

See Also

```
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
xvt_list_get_sel
```

xvt_list_get_sel

Get Selected Items in List Control

Summary

```
SLIST xvt_list_get_sel(WINDOW win)
```

`WINDOW win`

Window of control (`WC_LBOX` or `WC_LISTBUTTON`).

Description

This function gets all selected items from the `WC_LBOX` or `WC_LISTBUTTON` control specified by `win`. For `WC_LISTBUTTON` controls, the selected items are retreived from the "list" portion of the control.

The order of the elements of the `SLIST` is the same as the order of the items in the list control.

Each string in an `SLIST` is associated with a `long` data word. For this function, each data word is the index of the corresponding item in the control, not original data from an `SLIST`.

To retrieve the elements of the returned `SLIST`, you can call the `xvt_slist_*` functions. After you're done with the returned `SLIST`, it is your responsibility to free it with a call to `xvt_slist_destroy`.

`WC_LISTEDIT` doesn't allow any programmatic manipulation of the list selection, and cannot be used with this function.

Note: Always test the return result of this function to verify that there is a selection. This ensures that portablity is maintained.

Return Value

An `SLIST` containing items if successful; an empty `SLIST` if nothing is selected; `NULL` on error.

Parameter Validity and Error Conditions

`win` must be a `WC_LBOX` or `WC_LISTBUTTON` control.

See Also

SLIST
W_*, WC_*, WD_*, Values for WIN_TYPE
xvt_slist_*
xvt_slist_destroy
xvt_slist_get
xvt_slist_get_first
xvt_slist_get_next

Example

This code writes the selected items of a list box to a file. Note the call to `xvt_slist_destroy` at the end.

```
FILE *fp;
WINDOW lbox;
SLIST slist;
long dummy;
SLIST_ELT elt;
char *value;
...
slist = xvt_list_get_sel(lbox);
for (elt = xvt_slist_get_first(slist); elt;
    elt = xvt_slist_get_next(slist, elt))
{
    value = xvt_slist_get(slist, elt, &dummy);
    if (value)
    fprintf(fp, "%s
    ", value);
}
xvt_slist_destroy(slist);
fclose(fp);
```

xvt_list_get_sel_index

Get Index of First Selected Item in List

Summary

```
int xvt_list_get_sel_index(WINDOW win)

WINDOW win
```

Window of control (`WC_LBOX` or `WC_LISTBUTTON`).

Description

This function gets the index of the first selected item (origin 0) in the `WC_LBOX` or `WC_LISTBUTTON` control specified by `win`. For

WC_LISTBUTTON controls, the index is returned from the "list" portion of the control.

To get the text of the item, call xvt_list_get_elt or xvt_list_get_first_sel.

If the list is a WC_LBOX that has been created with the "multiple select" flag, then you should consider calling the function xvt_list_get_sel instead.

WC_LISTEDIT doesn't allow any programmatic manipulation of the list selection, and cannot be used with this function.

Note: Always test the return result of this function to verify that there is a selection. This ensures that portablity is maintained.

Return Value

The index if successful; -1 if an error occurs, or if nothing is selected.

Parameter Validity and Error Conditions

win must be a WC_LBOX or WC_LISTBUTTON control.

See Also

W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
xvt_list_get_elt
xvt_list_get_first_sel
xvt_list_rem
xvt_list_get_sel

Example

See the example for xvt_list_rem.

xvt_list_is_sel

Test if Item is Selected in List Control

Summary

```
BOOLEAN xvt_list_is_sel(WINDOW win, int index)
```

WINDOW win

Window of control (WC_LBOX or WC_LISTBUTTON).

int index

Item to be tested.

Description

This function tests whether the item is selected whose index (origin 0) is equal to `index` in the `WC_LBOX` or `WC_LISTBUTTON` control specified by `win`. For `WC_LISTBUTTON` controls, the function applies only to the "list" portion of the control.

`WC_LISTEDIT` doesn't allow any programmatic manipulation of the list selection, and cannot be used with this function.

Return Value

`TRUE` if an item is selected; `FALSE` if an item is not selected, or if an error occurs.

Parameter Validity and Error Conditions

`win` must be a `WC_LBOX` or `WC_LISTBUTTON` control.

See Also

`W_*, WC_*, WD_*, Values for WIN_TYPE`
`WINDOW`
`xvt_list_get_elt`

Example

See the example for `xvt_list_get_elt`.

---

xvt_list_rem

Delete Item in List Box

Summary

```
BOOLEAN xvt_list_rem(WINDOW win, int index)
```

`WINDOW win`

Window of control (`WC_LBOX`, `WC_LISTEDIT` or `WC_LISTBUTTON`).

`int index`

Item to delete.

Description

This function deletes the item whose index (origin 0) is equal to `index` from the `WC_LBOX`, `WC_LISTEDIT`, or `WC_LISTBUTTON` control

specified by win. For WC_LISTEDIT control, only the "list" portion of the control is affected. For a WC_LISTBUTTON, deleting a list item that is the current selection will result in an empty button label.

Note: To delete all items it is better to call xvt_list_clear.

Return Value

TRUE if successful; FALSE if unsuccessful (on error).

Parameter Validity and Error Conditions

win must be a WC_LBOX, WC_LISTBUTTON, or WC_LISTEDIT control.

See Also

W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
xvt_list_clear
xvt_list_get_sel_index
xvt_list_resume
xvt_list_suspend

Example

This code deletes selected items from a listbox:

```
WINDOW lbox;
int rip;
xvt_list_suspend(lbox);
rip = xvt_list_get_sel_index(lbox);
while (rip != -1)
{
    xvt_list_rem(lbox, rip);
    rip = xvt_list_get_sel_index(lbox);
}
xvt_list_resume(lbox);
```

xvt_list_resume

Resume List Control Updating

Summary

```
void xvt_list_resume(WINDOW win)

WINDOW win
```

Window of control (WC_LBOX, WC_LISTEDIT, or WC_LISTBUTTON).

Description

This function resumes updating of the WC_LBOX, WC_LISTEDIT, or WC_LISTBUTTON control specified by win. However, on WC_LISTEDIT or WC_LISTBUTTON controls, all programmatic changes are normally made while the list is not visible. Therefore, it is *not* essential to call xvt_list_resume for these two controls. However, if xvt_list_suspend was previously called (unnecessarily) for either of these two control types, your application should call xvt_list_resume to cause the WC_LISTEDIT or WC_LISTBUTTON to be updated. If your application does not do so, the results will be unpredictable on some platforms.

You should call this function to resume operation of a list control that has been suspended by a previous call to xvt_list_suspend. This function causes the list to be redrawn.

Parameter Validity and Error Conditions

win must be a WC_LBOX, WC_LISTBUTTON, or WC_LISTEDIT control.

See Also

```
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
xvt_list_add
xvt_list_rem
xvt_list_suspend
```

Example

See the examples for xvt_list_add and xvt_list_rem.

xvt_list_set_sel

Set Selection State of Item in List Control

Summary

```
BOOLEAN xvt_list_set_sel(WINDOW win, int index,
     BOOLEAN select)
```

`WINDOW win`

Window of control (WC_LBOX or WC_LISTBUTTON).

`int index`

Specified item to select or to unselect.

`BOOLEAN select`

Select or unselect the item.

Description

This function selects or unselects the item whose index (origin 0) is equal to `index` in the `WC_LBOX` or `WC_LISTBUTTON` control specified by `win`. For `WC_LISTBUTTON` controls, only the "list" portion of the control is affected. The selected item will be scrolled into view.

If `index` is equal to -1, this function selects or unselects all items, depending on `select`.

Don't attempt to select more items than allowed--some controls don't allow any selections, some allow only one, and some allow more than one. This is discussed in the "Controls" chapter in the*XVT Portability Toolkit Guide*.

`WC_LISTEDIT` doesn't allow any programmatic manipulation of the list selection, and cannot be used with this function.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful (on error).

Parameter Validity and Error Conditions

`win` must be a `WC_LBOX` or `WC_LISTBUTTON` control.

See Also

`W_*, WC_*, WD_*, Values for WIN_TYPE`
`WINDOW`

Example

This code unselects all items, and then selects item `n`:

```
WINDOW lbox;
int n;
if (xvt_list_set_sel(lbox, -1, FALSE) == FALSE
    || xvt_list_set_sel(lbox, n, TRUE) == FALSE)
xvt_dm_post_warning("Unable to select item %d", n);
```

xvt_list_suspend

Suspend Updating of List Control

Summary

```
void xvt_list_suspend(WINDOW win)
```

`WINDOW win`

Window of control (`WC_LBOX`, `WC_LISTEDIT`, or `WC_LISTBUTTON`).

Description

This function suspends updating of the `WC_LBOX`, `WC_LISTEDIT`, or `WC_LISTBUTTON` control specified by `win`, so that additions and deletions can be made quickly. However, on `WC_LISTEDIT` or `WC_LISTBUTTON` controls, all programmatic changes are normally made while the list is not visible. Therefore, it is *not* essential to call `xvt_list_suspend` for these two controls. However, if called, your application must also call `xvt_list_resume` to resume updating. If your application does not do so, the results will be unpredictable on some platforms.

The contents of the control are still changed while updating is suspended, but you will not see any changes until updating is resumed by a call to `xvt_list_resume`. Suspension of updating is optional, but usually it significantly increases speed and reduces gratuitous display activity when you change a list control's contents.

Parameter Validity and Error Conditions

`win` must be a `WC_LBOX`, `WC_LISTBUTTON`, or `WC_LISTEDIT` control.

See Also

```
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
xvt_list_add
xvt_list_rem
xvt_list_resume
```

Example

See the examples for `xvt_list_add` and `xvt_list_rem`.

xvt_mem_*

Memory Allocation Functions

```
xvt_mem_alloc
xvt_mem_free
xvt_mem_realloc
xvt_mem_rep
xvt_mem_zalloc
```

xvt_mem_alloc

Allocate Memory

Summary

```
DATA_PTR xvt_mem_alloc(size_t size)
```

`size_t size`

Size of the memory to be allocated, in bytes.

Description

XVT uses this function internally to allocate memory; it is analogous to the standard C function `malloc`. We recommend that you always use `xvt_mem_alloc` instead of using `malloc` for these reasons:

- XVT uses `xvt_mem_alloc` to allocate all memory, and might hand some of those pointers to your application. To avoid confusion, it is best to have all pointers allocated the same way.
- If you consistently use `xvt_mem_alloc` for your memory allocation, then you are free to change the implementation of your memory manager (using `ATTR_MEMORY_MANAGER`), whereas you would not be able to if you used `malloc`.

When you allocate any memory with `xvt_mem_alloc`, you must free it with `xvt_mem_free`, or resize it with `xvt_mem_realloc`. You must not free it with `free`, or reallocate it with `realloc`.

Return Value

A pointer to a block of at least "`size`" bytes if successful; `NULL` if out of memory.

See Also

ATTR_MEMORY_MANAGER
DATA_PTR
xvt_mem_free
xvt_mem_realloc
xvt_mem_rep
xvt_mem_zalloc

xvt_mem_free

Free Memory

Summary

```
void xvt_mem_free(DATA_PTR p)

DATA_PTR p
```

Pointer to memory.

Description

This function is used internally by XVT to free memory allocated with the `xvt_mem_*alloc` functions; it is analogous to the standard C function `free`. All memory allocated by XVT internally will be allocated by `xvt_mem_alloc`. Therefore, you must use this function to free any memory that is allocated for you by XVT. The `xvt_res_get_*_data` functions, which allocate strings, are examples of functions that allocate memory for you. You may override the default implementation of this function by using the `ATTR_MEMORY_MANGER` attribute.

Parameter and Validity Conditions

If `p` is `NULL`, XVT issues an error.

See Also

ATTR_MEMORY_MANAGER
DATA_PTR
xvt_mem_alloc
xvt_mem_realloc
xvt_mem_rep
xvt_mem_zalloc

xvt_mem_realloc

Resize Memory

Summary

```
DATA_PTR xvt_mem_realloc(DATA_PTR p, size_t size)
```

`DATA_PTR p`

Old pointer to memory.

`size_t size`

New size of memory in bytes.

Description

This function is used internally by XVT to resize memory allocated with `xvt_mem_alloc`; it is analogous to the standard C function `realloc`. You must use this function both to resize any memory allocated for you by XVT functions and to resize memory that was originally allocated with `xvt_mem_alloc`.

If the pointer to the old memory (`p`) is `NULL`, then `xvt_mem_realloc` behaves as `xvt_mem_alloc`. You may override the default implementation of this function by using the `ATTR_MEMORY_MANGER` attribute.

Return Value

A pointer to a block of at least "`size`" bytes if successful; `NULL` if out of memory.

See Also

```
ATTR_MEMORY_MANAGER
DATA_PTR
xvt_mem_alloc
xvt_mem_free
xvt_mem_rep
xvt_mem_zalloc
```

Example

See the example for `xvt_menu_set_tree`.

xvt_mem_rep

Repeat Block of Data

Summary

```
DATA_PTR xvt_mem_rep(DATA_PTR dst, DATA_PTR src,
     UINT srclen, long reps)
```

`DATA_PTR dst`

Pointer to destination buffer.

`DATA_PTR src`

Source block of data.

`UINT srclen`

Number of bytes in the source block.

`long reps`

Number of times the block of data is to be repeated.

Description

This function copies `reps` consecutive instances of data to memory pointed to by `dst`. The data is pointed to by `src`, and it is of length `srclen`.

Parameter and Validity Conditions

If either `dst` or `src` is `NULL`, XVT issues an error.

Return Value

Value of `dst` argument.

See Also

```
ATTR_MEMORY_MANAGER
DATA_PTR
xvt_mem_alloc
xvt_mem_free
xvt_mem_realloc
xvt_mem_zalloc
```

Example

See the example for `xvt_menu_set_tree`.

xvt_mem_zalloc

Allocate Zeroed Memory

Summary

```
DATA_PTR xvt_mem_zalloc(size_t size)

size_t size
```

Size of memory in bytes.

Description

This function allocates `size` bytes of memory via `xvt_mem_alloc`, and sets the contents to all zeros before returning a pointer to the memory. You must free the memory allocated with this function with `xvt_mem_free`, and resize it with `xvt_mem_realloc`. You may override the default implementation of this function by using the `ATTR_MEMORY_MANGER` attribute.

Return Value

A pointer to a block of at least "`size`" zeroed-out bytes if successful; `NULL` if out of memory.

See Also

```
ATTR_MEMORY_MANAGER
DATA_PTR
xvt_mem_alloc
xvt_mem_free
xvt_mem_realloc
xvt_mem_rep
xvt_menu_set_tree
```

Example

See the example for `xvt_menu_set_tree`.

xvt_menu_*

Menu Functions

```
xvt_menu_get_font_sel
xvt_menu_get_tree
xvt_menu_popup
xvt_menu_set_font_sel
xvt_menu_set_item_checked
xvt_menu_set_item_enabled
xvt_menu_set_item_title
xvt_menu_set_tree
xvt_menu_update
```

xvt_menu_get_font_sel

Get the State of the Font/Style Selection Menu or Dialog

Summary

```
XVT_FNTID xvt_menu_get_font_sel(WINDOW win)
```

`WINDOW win`

Window whose Font/Style menu information is being queried.

Description

This function returns an `XVT_FNTID` that represents either the state of check marks on the Font and Style submenus of the menu associated with `win`, or the current state of the Font Selection dialog (on platforms that do not provide a menu).

Return Value

`XVT_FNTID.`

Parameter Validity and Error Conditions

You must not call this function during an `E_UPDATE`.

`win` must be a window (possibly `TASK_WIN`) that has a menubar that contains XVT's Font/Style menu (e.g., one that uses **curl**'s `DEFAULT_FONT_MENU` in the menubar's specification); otherwise, XVT could issue an error.

Implementation Note

Only XVT/XM and XVT/Mac supply a full Font/Style menu. All other platforms provide a menu that invokes the Font Selection dialog. On XVT/Mac, the point sizes on the Style menu that correspond to available physical fonts are shown in Outline font.

See Also

```
DEFAULT_*_MENU Values
E_FONT
E_UPDATE
TASK_WIN
WINDOW
XVT_FNTID
xvt_menu_set_font_sel
```

The "Fonts and Text" and the "Menus" chapters in the*XVT Portability Toolkit Guide*

xvt_menu_get_tree

Get Entire Menu

Summary

```
MENU_ITEM* xvt_menu_get_tree(WINDOW win)
```

`WINDOW win`

Window whose menubar information is being queried.

Description

This function allocates memory for an appropriate `MENU_ITEM` tree and fills it with data that reflects `win`'s menubar.

For details on how `MENU_ITEM` arrays are connected together to form trees representing an entire menu hierarchy, see `MENU_ITEM`.

To load a `MENU_ITEM` tree from a resource file, use `xvt_res_get_menu`.

Return Value

A pointer to the `MENU_ITEM` tree. You can use `xvt_res_free_menu_tree` to release the memory allocated.

Parameter Validity and Error Conditions

XVT issues an error if `win` is not a top-level window that contains a menubar (i.e., was created without `WSF_NO_MENUBAR` specified). `win` can be a `TASK_WIN`.

See Also

```
MENU_ITEM
TASK_WIN
WINDOW
WSF_* Options Flags
xvt_font_map_using_default
xvt_menu_set_item_title
xvt_menu_set_tree
xvt_res_free_menu_tree
xvt_res_get_menu
```

The "Menus" chapter in the*XVT Portability Toolkit Guide*

Example

See the example for `xvt_menu_set_tree`.

xvt_menu_popup

Display Popup Menu Over a Window

Summary

```
BOOLEAN xvt_menu_popup(MENU_ITEM *menu_p,
    WINDOW win,
    PNT pos,
    XVT_POPUP_ALIGNMENT alignment,
    MENU_TAG item);
```

`MENU_ITEM *menu_p`

Tree structure of items for the popup menu.

`WINDOW win`

Window for popup display. Identifies the event handler to which the popup menu `E_COMMAND` event is delivered.

`PNT pos`

Position of display popup (in `win`'s coordinate system). Actual display position is also affected by the specified alignment.

`XVT_POPUP_ALIGNMENT alignment`

Changes the display positioning of the popup menu.

`MENU_ITEM item`

Tag identifying the "default" top-level menu item. This argument is used only when alignment is set to `XVT_POPUP_OVER_ITEM`.

Description

This function displays a popup menu (based on the provided `MENU_ITEM` tree structure) at a specified location over a window. The exact placement of the popup menu may vary if it is displayed too close to the edge of the screen (depending on the native toolkit). For example, the XVT/XM portability toolkit automatically adjusts the menu display position so that it will be completely visible.

You can obtain a `MENU_ITEM` tree appropriate for this function by calling `xvt_res_get_menu` and using any of the top-level `SUBMENU` items as a popup menu tree. In this case, `xvt_res_get_menu` can be viewed as retrieving an array of popup menu trees. You can also construct individual popup menu trees and pass them into this function. For more information on how `MENU_ITEM` trees are constructed, see the `MENU_ITEM`.

Return Value

`TRUE` if a valid menu was created; `FALSE` otherwise.

Parameter Validity and Error Conditions

XVT issues an error if any of the following parameter conditions are not met:

- `win` must be `W_DOC`, `W_PLAIN`, `W_DBL`, `W_NO_BORDER`, or `W_MODAL`
- `menu_p` must not be `NULL`
- if the `XVT_POPUP_OVER_ITEM` flag is set, the specified item must be a top-level menu item of the `menu_p` tree

Implementation Note

For portability across all native toolkits supported by XVT, you may *only* call this function in response to an `E_MOUSE_DOWN` event. This is because on XVT/Mac, popup menus are not displayed when invoked in response to any other type of event.

On all platforms normal event processing is blocked. This may result in recursive calls to `win`'s event handler. This means that an `E_COMMAND` may be received sometime after `xvt_menu_popup` has returned. You should take this into account when coding your application. In either case, if the user dismisses the popup menu without making a selection, an `E_COMMAND` is *not* sent.

On XVT/Win32, `win` must be a drawable task window when the attribute `ATTR_WIN_PM_DRAWABLE_TASKWIN` has been set to `TRUE`.

See Also

```
E_COMMAND
E_MOUSE_DOWN
MENU_ITEM
MENU_TAG
PNT
WINDOW
XVT_POPUP_ALIGNMENT
xvt_res_free_menu_tree
xvt_res_get_menu
```

The "Menus" chapter in the*XVT Portability Toolkit Guide*

Example

```
/* global application data... */MENU_ITEM *popup_menus;/
* in the event handler of your applications task window...
*/case E_CREATE:
        /* get an array of popup menus from resources,
            this code */
        /* assumes that POPUP_MENUS is a valid MENUBAR
            resource id */
    popup_menus = xvt_res_get_menu (POPUP_MENUS);
    break;
        /* in one of your applications top-level window
            event handlers... */case E_MOUSE_DOWN:
        /* see if mouse is in any popup region and popup
            one of the menus, */
        /* this code assumes that DEFAULT_MENU_TAG is a
            valid menu item tag */
    pnt = event->v.mouse.where;
    if (pnt.h > 100 && pnt.h < 150 && pnt.v > 100
        && pnt.v < 150)
        xvt_popup_menu (popup_menus[0]->child, win, pnt,
            XVT_POPUP_LEFT_ALIGNED, 0);
    else if (pnt.h > 150 && pnt.h < 200 && pnt.v > 150
        && pnt.v < 200)
         xvt_popup_menu (popup_menus[1]->child, win, pnt,
            XVT_POPUP_CENTERED, 0);
    else if (pnt.h > 200 && pnt.h < 250 && pnt.v > 200
        && pnt.v < 250)
        xvt_popup_menu (popup_menus[2]->child, win, pnt,
            XVT_POPUP_RIGHT_ALIGNED, 0);
    else
        xvt_popup_menu (popup_menus[3]->child, win, pnt,
            XVT_POPUP_OVER_ITEM, DEFAULT_ITEM_TAG);
    break;
```

xvt_menu_set_font_sel

Set the State of the Font/Style Selection Menu or Dialog

Summary

```
void xvt_menu_set_font_sel(WINDOW win,
     XVT_FNTID font_id)
```

`WINDOW win`

Window whose Font/Style menu is being changed.

`XVT_FNTID font_id`

Handle of a logical font, or `NULL_FNTID`.

Description

This function sets the Font Selection dialog or the Font/Style menu to match the `XVT_FNTID` specified by `font_id`.

On the Font/Style submenus of the menu associated with `win`, this function sets the check marks. Users see the appropriate check marks when they drop the Font or Style menus. If `font_id` is `NULL_FNTID`, all check marks are removed. On systems where the native look-and-feel implementation for font selection is a dialog, users will also see the appropriate default physical font if they bring up the Font Selection dialog.

Tip: If your application uses a single physical font throughout the entire window (for example a text editor), you should set the Font menu to the `XVT_FNTID` displayed in the window. However, if your application allows the display and selection of different text objects drawn with different physical fonts, then it should set the menu check marks to match the `XVT_FNTID` used in drawing the currently selected item. If there is no currently selected item, the Font menu should either be completely unchecked or set to the `XVT_FNTID` that would be used if a new text item were created.

Return Value

None.

Parameter Validity and Error Conditions

If the following parameter conditions are not met, XVT issues an error:

- `win` must be a window (possibly `TASK_WIN`) with a menubar that contains XVT's Font/Style menu (e.g., one that uses **curl**'s `DEFAULT_FONT_MENU` in the menubar's specification)
- `XVT_FNTID` must be a valid logical font

Implementation Note

Only XVT/XM and XVT/Mac supply a full Font/Style menu. All other platforms provide a menu that invokes the Font Selection dialog. On the Macintosh, the point sizes on the Style menu that correspond to available physical fonts are shown in Outline font.

See Also

DEFAULT_*_MENU Values
E_FONT
NULL_FNTID
TASK_WIN
WINDOW
xvt_menu_get_font_sel

The "Fonts and Text" and the "Menus" chapters in the*XVT Portability Toolkit Guide*

xvt_menu_set_item_checked

Check Menu Item

Summary

```
void xvt_menu_set_item_checked(WINDOW win,
     MENU_TAG tag, BOOLEAN check)
```

`WINDOW win`

Window whose menu item is to be checked.

`MENU_TAG tag`

Menu item in the menu associated with `win`.

`BOOLEAN check`

If `TRUE`, a check mark is placed next to the `tag`; if `FALSE`, a check mark is removed.

Description

If `check` is `TRUE`, this function places a check mark next to the tag menu item in the menu associated with `win`. If `check` is `FALSE`, this function removes the check mark.

To place or remove check marks on the Font or Style menus, use `xvt_menu_set_font_sel`.

Parameter Validity and Error Conditions

XVT issues an error if any of the following parameter conditions are not met:

- `win` must be a top-level window that was created without `WSF_NO_MENUBAR` specified; or, it can be `TASK_WIN`

- `tag` must match a menu item in the menubar; however, if `tag` refers to one of XVT's predefined menu items, no error is issued
- The `tag` menu item must be `checkable`; the checkability of a menu item is set when the menu is created, by setting the `checkable` bit in the item's `MENU_ITEM` structure, or by specifying the `checkable` keyword in the menu's URL statement

See Also

```
MENU_ITEM
MENU_TAG
WINDOW
WSF_* Options Flags
xvt_menu_set_font_sel
```

The "Menus" chapter in the*XVT Portability Toolkit Guide*

xvt_menu_set_item_enabled

Enable Menu Item

Summary

```
void xvt_menu_set_item_enabled(WINDOW win,
     MENU_TAG tag, BOOLEAN enable)
```

`WINDOW win`

Window whose menu item is to be enabled.

`MENU_TAG tag`

Menu item in the menu associated with `win`.

`BOOLEAN enable`

If `TRUE`, the specified menu item is enabled; if `FALSE`, the item is disabled.

Description

If `enable` is `TRUE`, this function enables the `tag` menu item in the menu associated with `win`. If `enable` is `FALSE`, this function disables (grays) the `tag` menu item. If `tag` refers to an entire submenu, then the entire submenu and its name on the menubar are disabled. However, the previous states (disabled or enabled) of the individual

items are remembered, so that re-enabling the submenu restores the state of each menu item to the way it was.

When enabling or disabling top-level items, the change cannot be shown on the menubar until you call `xvt_menu_update`. This call is only needed when enabling or disabling top-level items.

Individual items on the Font or Style menus can't be disabled or enabled, but the menus can be disabled or enabled in their entirety by calling `xvt_menu_set_item_enabled` with a `tag` of type `FONT_MENU_TAG`.

Parameter Validity and Error Conditions

XVT issues an error if any of the following parameter conditions are not met:

- `win` must be a top-level window that was created without `WSF_NO_MENUBAR` specified, or it may be `TASK_WIN`
- `tag` must match a menu item in the menubar; however, if `tag` refers to one of XVT's predefined menu items, no error is issued

Implementation Note

When an entire submenu is disabled, whether the user can still pull it down and look at it depends on the platform.

See Also

```
FONT_MENU_TAG
MAX_MENU_TAG
MENU_TAG
TASK_WIN
WINDOW
WSF_* Options Flags
xvt_menu_set_item_checked
xvt_menu_set_item_title
xvt_menu_update
```

The "Menus" chapter in the*XVT Portability Toolkit Guide*

xvt_menu_set_item_title

Set Text of Menu Item

Summary

```
void xvt_menu_set_item_title(WINDOW win,
     MENU_TAG tag, char *text)
```

`WINDOW win`

Window whose menu item is to be set.

`MENU_TAG tag`

Menu item in the menu associated with `win`.

`char *text`

Pointer to the `NULL`-terminated text string.

Description

This function changes the text of the menu item designated by `tag`, on the menu associated with `win`. The item's text is changed to the `NULL`-terminated string pointed to by `text`. You can use `xvt_menu_set_item_title` to change the text of items on either the top-level menubar, or items located in submenus.

`tag` cannot be set to `FONT_MENU_TAG`.

If you need to change a menubar extensively, such as adding and deleting items or menus, then use `xvt_menu_get_tree` and `xvt_menu_set_tree`.

Parameter Validity and Error Conditions

XVT issues an error if any of the following parameter conditions are not met:

- `win` must be a top-level window that was created without `WSF_NO_MENUBAR` specified, or it may be `TASK_WIN`.
- `tag` must match a menu item in the menubar. However, if `tag` refers to one of XVT's predefined menu items, no error is issued.

See Also

```
FONT_MENU_TAG
MENU_TAG
TASK_WIN
WINDOW
WSF_* Options Flags
xvt_menu_get_tree
xvt_menu_set_tree
```

The "Menus" chapter in the*XVT Portability Toolkit Guide*

xvt_menu_set_tree

Set Entire Menu

Summary

```
void xvt_menu_set_tree(WINDOW win, MENU_ITEM *menu_p
```

`WINDOW win`

Window whose entire menu is to be set.

`MENU_ITEM *menu_p`

Tree structure for a new menubar.

Description

This function synthesizes a menubar for the given window based on the provided `MENU_ITEM` tree structure.

You can obtain a `MENU_ITEM` tree appropriate for this function by calling `xvt_res_get_menu` or `xvt_menu_get_tree`. You can also construct one yourself. For details of how `MENU_ITEM` trees are linked together, see `MENU_ITEM`.

Parameter Validity and Error Conditions

XVT issues an error if any of the following parameter conditions are not met:

- `win` must be a top-level window that was created without `WSF_NO_MENUBAR` specified; or, it can be `TASK_WIN`
- `menu_p` must not be `NULL`

See Also

```
MENU_ITEM
TASK_WIN
WINDOW
WSF_* Options Flags
xvt_menu_get_tree
xvt_res_free_menu_tree
xvt_res_get_menu
```

The "Menus" chapter in the*XVT Portability Toolkit Guide*

Example

This code dynamically adds a menu to the current menubar. The new "Display" menu contains "Show" and "Hide All" items; "Show" has a pull-right menu containing "Primary" and "Secondary."

```
MENU_ITEM *menubar;
int num_menus;
    /* get current menubar */
menubar = xvt_menu_get_tree(window);
    /* count current menus */
num_menus = 0;
while (menubar[num_menus].tag)
    num_menus++;
    /* allocate space for new menus */
menubar = (MENU_ITEM *) xvt_mem_realloc(
    (DATA_PTR) menubar,
    sizeof(MENU_ITEM) * (num_menus + 1 + 1));
    /* +1 for new menus +1 for termination */
    /* zero memory for new menus */
xvt_mem_rep((DATA_PTR) &menubar[num_menus], "0",
    1, sizeof(MENU_ITEM) * (1 + 1));
    /* add new menu to menubar */
menubar[num_menus].tag = DISPLAY_MENU_TAG;
menubar[num_menus].text = xvt_str_duplicate ("Display");
menubar[num_menus].enabled = 1;
    /* allocate memory for menu items */
menubar[num_menus].child = (MENU_ITEM *)
    xvt_mem_zalloc(sizeof(MENU_ITEM)*(2+1));
    /* +2 for new menu items +1 for termination */
    /* add new menu items "Show" and "Hide All" */
menubar[num_menus].child[0].tag = SHOW_MENU_TAG;
menubar[num_menus].child[0].text =
    xvt_str_duplicate ("Show");
menubar[num_menus].child[0].enabled = 1;
menubar[num_menus].child[1].tag = HIDE_MENU_TAG;
menubar[num_menus].child[1].text =
    xvt_str_duplicate ("Hide All");
menubar[num_menus].child[1].enabled = 1;
    /* add pull right menu for "Show" */
menubar[num_menus].child[0].child = (MENU_ITEM *)
    xvt_mem_zalloc(sizeof(MENU_ITEM)*(2+1));
    /* +2 for new menu items +1 for termination */
menubar[num_menus].child[0].child[0].tag = SHOW_PRIMARY;
menubar[num_menus].child[0].child[0].text =
    xvt_str_duplicate ("Primary");
menubar[num_menus].child[0].child[0].enabled = 1;
menubar[num_menus].child[0].child[1].tag = SHOW_SECOND;
menubar[num_menus].child[0].child[1].text =
    xvt_str_duplicate ("Secondary");
menubar[num_menus].child[0].child[1].enabled = 1;
    /* change menu to new menubar */
xvt_menu_set_tree(window, menubar);
    /* free resources (but do not free string constants)*/
xvt_res_free_menu_tree(menubar);
```

xvt_menu_update

Display Menubar Changes

Summary

```
void xvt_menu_update(WINDOW win)
```

`WINDOW win`

Window whose menubar is being changed.

Description

This function changes the `win` menubar to reflect any changes due to a call to `xvt_menu_set_item_enabled`. To make the changes visible on some platforms, it is necessary to call `xvt_menu_update`.

Do not confuse `xvt_menu_update` with the processing of `E_UPDATE` events. Menubars, like other controls, take care of all their own updating. Therefore, calling `xvt_menu_update` is unnecessary and illegal during the processing of `E_UPDATE` events.

If you make wholesale changes to a menubar by calling `xvt_menu_set_tree`, then calling this function is unnecessary because the changes due to `xvt_menu_set_tree` are visible immediately.

Parameter Validity and Error Conditions

XVT issues an error if `win` is not a top-level window that contains a menubar (i.e., was created without `WSF_NO_MENUBAR` specified). `win` can be `TASK_WIN`.

Implementation Note

On some platforms, all changes due to `xvt_menu_set_item_enabled` are immediately visible, and `xvt_menu_update` is ignored.

See Also

```
E_UPDATE
TASK_WIN
WINDOW
WSF_* Options Flags
xvt_menu_set_item_enabled
xvt_menu_set_tree
```

The "Menus" chapter in the*XVT Portability Toolkit Guide*

xvt_nav_*

Navigation Functions

```
xvt_nav_add_win
xvt_nav_create
xvt_nav_destroy
xvt_nav_list_wins
xvt_nav_rem_win
```

xvt_nav_add_win

Adds a Control or Child Window to a Navigation Object

Summary

```
BOOLEAN xvt_nav_add_win(XVT_NAV nav, WINDOW win,
    WINDOW refwin, XVT_NAV_INSERTION where);
```

`XVT_NAV nav`

Navigation object in which to insert a control or child window.

`WINDOW win`

Control or child window to be added to the navigation order. It must be of type `WC_*`, `W_PLAIN`, or `W_NO_BORDER`.

`WINDOW refwin where`

A control or child window in the existing navigation object that serves a reference point for inserting the new control or child window.

`XVT_NAV_INSERTION`

The insertion position for `win`.

Description

`xvt_nav_add_win` inserts controls or child windows into a navigation order at the position specified by `where`.

If your application passes a value of `XVT_NAV_POS_BEFORE` or `XVT_NAV_POS_AFTER` to `where`, then XVT inserts `win` into the navigation order before or after the control or child window denoted by `refwin`.

If your application passes a value of XVT_NAV_POS_FIRST or XVT_NAV_POS_LAST to where, then XVT ignores refwin and inserts the control or child window at the beginning or end of the navigation order.

Return Value

TRUE if the window was added successfully, otherwise FALSE.

Parameter Validity and Error Conditions

XVT issues an error if your application does not meet the following conditions for parameters passed to xvt_nav_add_win:

- nav is a valid XVT_NAV object
- win is a valid WINDOW
- win must be an immediate descendent of the window that nav is associated with
- The win passed must contain valid control or child windows for navigation (types WC_*, W_PLAIN, or W_NO_BORDER)
- refwin is in the navigation object list

See Also

XVT_NAV
XVT_NAV_INSERTION
xvt_nav_rem_win
xvt_win_get_nav

Example

This code sample shows the addition of a push button to a navigation object. win is an XVT WINDOW that contains a navigation object.

```
#define BUTTON_ID 101
...
RCT rect;
WINDOW new_ctl;
XVT_NAV nav;
...
xvt_rect_set(&rect, BUTTON_X, BUTTON_Y, BUTTON_X + 100,


    BUTTON_Y + (int)xvt_vobj_get_attr(win,
    ATTR_CTL_BUTTON_HEIGHT));
new_ctl = xvt_ctl_create(WC_PUSHBUTTON, &rect, "Push Me",
    parent, 0L, 0L, BUTTON_ID);
nav = xvt_win_get_nav(win);
xvt_nav_add_win(nav, new_ctl, NULL_WIN,
    XVT_NAV_POS_LAST);
...
```

xvt_nav_create

Create a Navigation Object

Summary

```
XVT_NAV xvt_nav_create(WINDOW win, SLIST win_list);
```

`WINDOW win`

Container window in which navigation order is set. `win` must be of type `W_DOC`, `W_PLAIN`, `W_DBL`, `W_MODAL`, or `W_NO_BORDER`.

`SLIST`

An ordered list of control or child windows used by the navigation object in the container window.

Description

`xvt_nav_create` creates a navigation object in the specified window, `win`. If `win_list` is `NULL`, then the navigation object uses the immediate children of `win`.

The order of items in the `win_list` list determines the order that a user may navigate through controls or child windows. If `win_list` is `NULL`, then the order of navigation follows the creation order of the controls and child windows.

`SLIST` string items must be set to `NULL` for navigation objects (this data is reserved for future use).

`XVT_NAV` objects allow you to set default and escape pushbuttons. To specify a default button, set the control ID for a `WC_PUSHBUTTON` in the navigation order to `DLG_OK`. To specify an escape button, set the control ID to `DLG_CANCEL`.

Some GUI objects may be associated for special navigation within a group. Currently *only* `WC_RADIOBUTTON` controls may be grouped for radiobutton selection. A navigation group is one or more (groupable) objects listed within a navigation object list.

The beginning of a navigation group meets one of the following criteria:

- The first object of a group is a groupable object with its `CTL_GROUP_FLAG` creation flag set and it immediately follows a groupable object from another group.

-OR-

- The first object of a group is a groupable object immediately following any non-groupable object.

The end of a navigation group meets one of the following criteria:

- The last object of a group is a groupable object which immediately precedes a groupable object with its `CTL_FLAG_GROUP` creation flag set.

-OR-

- The last object of a group is a groupable object which immediately precedes any non-groupable object.
- To remove a navigation object from `win`, call `xvt_nav_destroy`.

Return Value

A valid `XVT_NAV` object if successful; otherwise `NULL`.

Parameter Validity and Error Conditions

XVT issues an error if your application does not meet the following conditions for parameters passed to `xvt_nav_create`:

- `win` is a valid window
- `win` is of type `W_DOC`, `W_PLAIN`, `W_DBL`, `W_MODAL`, or `W_NO_BORDER`
- The `SLIST` passed must contain valid control or child windows for navigation (types `WC_*`, `W_PLAIN`, or `W_NO_BORDER`)
- There is no current navigation defined for the window
- `SLIST` strings are `NULL`

See Also

`W_*, WC_*, WD_*, Values for WIN_TYPE`
`WINDOW`
`SLIST`
`XVT_NAV`
`xvt_nav_add_win`
`xvt_nav_destroy`
`xvt_win_get_nav`

Example

This code creates and destroys a navigation object:

```
long XVT_CALLCONV1 win_eh(WINDOW win, EVENT *ep) {
    SLIST win_list;
    XVT_NAV nav;
    ...
    switch(ep->type) {
        case E_CREATE:
            ...
            win_list = xvt_win_list_wins(win, 0L);
            xvt_nav_create(win, win_list);
            break;
            ...
        case E_DESTROY:
            ...
            nav = xvt_win_get_nav(win);
            if(nav)
                xvt_nav_destroy(nav);
            break;
    }
    ....
}
```

xvt_nav_destroy

Destroys a Navigation Object

Summary

```
void xvt_nav_destroy(XVT_NAV nav);
```

`XVT_NAV nav`

Navigation object to be destroyed.

Description

`xvt_nav_destroy` destroys the specified navigation object.

Return Value

None.

Parameter Validity and Error Conditions

If `nav` is not a valid `XVT_NAV` object, XVT issues an error.

See Also

`XVT_NAV`
`xvt_nav_create`

Example

See `xvt_nav_create` for an example of this function.

xvt_nav_list_wins

Retrieves the List of Controls or Child Windows from a Navigation Object

Summary

```
SLIST xvt_nav_list_wins(XVT_NAV nav);
```

`XVT_NAV nav`

Navigation object from which to obtain navigation list.

Description

`xvt_nav_list_wins` returns an `SLIST` of controls and child windows that are in the navigation order of the navigation object. The returned list of object is in the order of navigation.

Return Value

A valid `SLIST` containing the navigation controls and child windows; `NULL` on error.

Parameter Validity and Error Conditions

If `nav` is not a valid `XVT_NAV` object, XVT issues an error.

See Also

`SLIST`
`XVT_NAV`
`xvt_nav_add_win`
`xvt_nav_create`

xvt_nav_rem_win

Removes a Control or Child Window from the Navigation Object

Summary

```
BOOLEAN xvt_nav_rem_win(XVT_NAV nav, WINDOW win);
```

`XVT_NAV nav`

The navigation object from which to remove a control or child window.

`WINDOW win`

The control or child window to remove from the navigation order.

Description

`xvt_nav_rem_win` removes a control or child window from the navigation object list.

Return Value

`TRUE` if the window was removed; otherwise `FALSE`.

Parameter Validity and Error Conditions

XVT issues an error if your application does not meet the following conditions for parameters passed to `xvt_nav_rem_win`:

- `nav` is a valid `XVT_NAV` object
- `win` is a valid `WINDOW`
- `win` must be in the navigation object list
- The `win` passed must contain valid control or child windows for navigation (types `WC_*`, `W_PLAIN`, or `W_NO_BORDER`)

See Also

`WINDOW`
`XVT_NAV`
`xvt_nav_add_win`

Example

The following code removes the push button added in the example code for `xvt_nav_add_win`:

```
 ...
WINDOW win, ctl_win;
XVT_NAV nav;
...
ctl_win = xvt_win_get_ctl(win, BUTTON_ID);
nav = xvt_win_get_nav(win);
xvt_nav_rem_win(nav, ctl_win);
...
```

xvt_notebk_*

Notebook Functions

```
xvt_notebk_add_page
xvt_notebk_add_tab
xvt_notebk_create_face
xvt_notebk_create_face_def
xvt_notebk_create_face_res
xvt_notebk_enum_pages
xvt_notebk_get_face
xvt_notebk_get_front_page
xvt_notebk_get_num_pages
xvt_notebk_get_num_tabs
xvt_notebk_get_page_data
xvt_notebk_get_page_from_face
xvt_notebk_get_page_title
xvt_notebk_get_tab_image
xvt_notebk_get_tab_title
xvt_notebk_rem_page
xvt_notebk_rem_tab
xvt_notebk_set_page_data
xvt_notebk_set_page_title
xvt_notebk_set_front_page
xvt_notebk_set_tab_image
xvt_notebk_set_tab_title
```

xvt_notebk_add_page

Add a Page to a Specific Tab in a Notebook Control

Summary

```
void xvt_notebk_add_page (WINDOW notebk, short tab_no,
    short page_no, char * title, long page_data)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of tab to which to add a page.

`short page_no`

Position at which to add new page. If 0, it will be the first page. The page is placed before the page whose index is equal to

`page_no` (origin `0`). A `page_no` that is too large causes the page to be added at the end.

`char * title`

Title of page.

`long page_data`

Contains any application data you wish to attach to a page. Typically, this will be a pointer to some structure allocated from the heap, cast into a `long` so that, later, your application can retrieve the structure and look at it.

Description

This function adds a page to a notebook tab.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- notebk is `NULL` or not a valid notebook control
- `tab_no` is less than `0`
- `page_no` is less than `0`

Implementation Note

The multiple page mechanism allows one tab to have many pages. Each page has a `Face (XVT WINDOW)`. You must come up with a way for the user to change pages for tabs with more than one page. For example, you may provide "Next" and "Prev" buttons on each face for tabs with multiple pages.

See Also

```
xvt_notebk_add_tab
xvt_notebk_create_face
xvt_notebk_create_face_def
xvt_notebk_create_face_res
xvt_notebk_get_page_data
xvt_notebk_set_page_data
```

xvt_notebk_add_tab

Add a Tab to a Notebook Control

Summary

```
void xvt_notebk_add_tab (WINDOW notebk, short tab_no,
    char * title, XVT_IMAGE image)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of tab to add;Position at which to add new tab. If 0, this will be the first tab. The tab is placed before the tab whose index is equal to `tab_no` (origin `0`). A `tab_no` that is too large causes the tab to be added at the end.

`char * title`

Title of tab. If it is a `NULL_STRING`, the tab will have no title.

`XVT_IMAGE image`

Image to display in tab. If it is a `NULL_IMAGE`, the tab will have no image.

Description

This function adds a tab to a notebook control. Each tab is analogous to a divider in a notebook.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`

Implementation Note

A tab may have multiple pages. The image is duplicated. The application owns image and must destroy it when finished with it.

See Also

```
XVT_IMAGE
xvt_image_destroy
xvt_notebk_add_page
```

Example

```
anImage =  xvt_image_read_bmp("window.bmp");
xvt_notebk_add_tab (aNotebk, 0, "Window", anImage);
xvt_notebk_add_page(aNotebk, 0, 0, "Page0",0L);
aFace = xvt_res_get_win_def(WINDOW_FACE);
xvt_notebk_create_face_def(aNotebk, 0, 0, aFace, EM_ALL,
    WINDOW_FACE_eh, PTR_LONG(&aData->winFace));
xvt_res_free_win_def(aFace);
xvt_image_destroy(anImage);
```

xvt_notebk_create_face

Create a Face for a Page

Summary

```
WINDOW xvt_notebk_create_face (WINDOW notebk,
    short tab_no, short page_no, EVENT_MASK mask,
    EVENT_HANDLER face_eh, long app_data)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of page to which face is being added.

`short page_no`

Page number for which to create face.

`EVENT_MASK mask`

Specifies which events should be sent to the window handler. This is an `OR`'d combination of any of the `EM_*` constants. You usually set `EVENT_MASK mask` to `EM_ALL` indicating that all events should be sent to the window (no restriction). In some conditions, you can restrict the events sent to the window. For more details, see the "Event Masking" section of the "Events" chapter in the *XVT Portability Toolkit Guide*.

`EVENT_HANDLER face_eh`

The event handler function; it receives all of the events for the window.

`long app_data`

Contains any application data you wish to attach to the window when it is created. Normally, it is a pointer to a data structure cast into a `long`.

Description

This function creates a face for the page identified by `page_no`. A face is simply an XVT child WINDOW. There is a one to one relationship between a page and a face. The face is displayed when the page it is associated with is selected.

Return Value

A WINDOW if successful; `NULL_WIN` if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`
- `page_no` is less than `0`

See Also

`xvt_notebk_create_face_def`
`xvt_notebk_create_face_res`

---

xvt_notebk_create_face_def

Create a Face with Controls from an Array of Data Structures

Summary

```
WINDOW xvt_notebk_create_face_def (WINDOW notebk,
    short tab_no, short page_no, WIN_DEF * win_def_p,
    EVENT_MASK  mask, EVENT_HANDLER face_eh,
    long app_data)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of page to which face is being added.

`short page_no`

Page number for which to create face.

`WIN_DEF * win_def_p`

Points to an array of data structures. The first element in the array defines the window itself. Subsequent elements of the array define the controls or text edit objects contained within the window. The last element of the array is a terminator whose `wtype` field is set to `W_NONE`. `EVENT_MASK mask` specifies which events are sent to the window event handler. This is an `OR`'d combination of any of the `EM_*` constants. You usually set this to `EM_ALL` indicating that all events would be sent to the window. For more details, see the "Event Masking" section of the "Events" chapter in the *XVT Portability Toolkit Guide*.

`EVENT_HANDLER face_eh`

The event handler function; it receives all of the events for the window.

`long app_data`

Contains any application data you wish to attach to the window when it is created. Normally, it is a pointer to a data structure cast into a `long`.

Description

This function creates a face for the page identified by `page_no`. A face is simply an XVT child WINDOW. There is a one to one relationship between a page and a face. The face is displayed when the page it is associated with is selected. For more information on `WIN_DEF` structures see `xvt_win_create_def`.

Return Value

A WINDOW if successful; `NULL_WIN` if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control.
- `tab_no` is less than `0`.
- `page_no` is less than `0`.

See Also

`xvt_notebk_create_face`
`xvt_notebk_create_face_res`
`xvt_win_create_def`

xvt_notebk_create_face_res

Create a Face from a Resource File

Summary

```
WINDOW xvt_notebk_create_face_res (WINDOW notebk,
    short tab_no, short page_no, int rid, EVENT_MASK mask,
    EVENT_HANDLER face_eh, long app_data)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of page to which face is being added.

`short page_no`

Page number for which to create face.

`int rid`

Resource ID corresponding to a window statement in your **URL** resource file. The face is created as if this resource were loaded via `xvt_res_get_win_def`, and then instantiated via `xvt_notebk_create_face_def`.

`EVENT_MASK mask`

Specifies which events are sent to the window event handler. This is an `OR`'d combination of any of the `EM_*` constants. You usually set this to `EM_ALL` indicating that all events would be sent to the window. For more details, see the "Event Masking" section of the "Events" chapter in the *XVT Portability Toolkit Guide*.

`EVENT_HANDLER face_eh`

The event handler function; it receives all of the events for the window.

`long app_data`

Contains any application data you wish to attach to the window when it is created. Normally, it is a pointer to a data structure cast into a `long`.

Description

This function creates a face for the page identified by `page_no`. A face is simply an XVT child WINDOW. There is a one to one

relationship between a page and a face. The face is displayed when the page it is associated with is selected.

Return Value

A WINDOW if successful; `NULL_WIN` if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control.
- `tab_no` is less than `0`.
- `page_no` is less than `0`.
- `face_eh` must be set to a valid function pointer.
- `rid` specifies a window resource in the **URL** file.

See Also

`xvt_notebk_create_face`
`xvt_notebk_create_face_def`

xvt_notebk_enum_pages

Enumerate through All Pages and Apply the Function to Each Page

Summary

```
BOOLEAN xvt_notebk_enum_pages (WINDOW notebk,
    XVT_NOTEBK_ENUM_PAGES func, long data)
```

`WINDOW notebk`

Window ID of notebook control.

`XVT_NOTEBK_ENUM_PAGES func`

Address of function to be called for each page.

`long data`

Application-defined data to pass to callback function.

Description

This function enumerates (i.e., invokes an application-supplied callback function) the pages contained in the specified tab. It passes the `notebk`, `tab_no`, `page_no` and `page_data` of each page, in sequential order, to an application-defined callback function. It

continues until the last page is enumerated or until the callback function returns `FALSE`.

Return Value

`TRUE` if successful; `FALSE` otherwise.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control.
- `tab_no` is less than `0`.
- `page_no` is less than `0`.

See Also

`XVT_NOTEBK_ENUM_PAGES`

---

xvt_notebk_get_face

Get the Face in the Notebk at Tab and Page

Summary

```
WINDOW xvt_notebk_get_face (WINDOW notebk, short tab_no,
    short page_no)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of face.

`short page_no`

Page number of face.

Description

This function gets the face (`XVT WINDOW`) in the notebk at tab and page.

Return Value

The WINDOW of the face if successful, or `NULL_WIN` if no such face exists.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`
- `page_no` is less than `0`

See Also

`xvt_notebk_get_front_page`

---

xvt_notebk_get_front_page

Get the Current Front Page

Summary

```
WINDOW xvt_notebk_get_front_page (WINDOW  notebk,
    short * tab_no, short * page_no)
```

`WINDOW  notebk`

Window ID of notebook control.

`short * tab_no`

Tab number of tab with the front page.

`short * page_no`

Page number of front page.

Description

This function gets the current front page. This is the page that is currently on top and showing. It also returns the face (`WINDOW`) of the front page.

Return Value

The WINDOW of the face if successful, or `NULL_WIN` unsuccessful. `tab_no` and/or `page_no` may be set to `-1` on error.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is a `NULL` pointer
- `page_no` is a `NULL` pointer

See Also

`xvt_notebk_get_face`

xvt_notebk_get_num_pages

Get the Number of Pages in the Specified Tab

Summary

```
short xvt_notebk_get_num_pages (WINDOW notebk,
    short tab_no)
```

`WINDOW  notebk`

Window ID of notebook control.

`short * tab_no`

Tab number for which to get the number of pages.

Description

This function gets the number of pages in a notebk at the tab specified.

Return Value

The number of pages in the tab if successful, or `0` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control.
- `tab_no` is less than `0`.

See Also

`xvt_notebk_get_num_tabs`

xvt_notebk_get_num_tabs

Get the Number of Tabs in a Notebk

Summary

```
short xvt_notebk_get_num_tabs (WINDOW notebk)
```

`WINDOW  notebk`

Window ID of notebook control.

Description

This function gets the number of tabs in a notebk.

Return Value

The number of pages in the tab if successful, or `0` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control

See Also

`xvt_notebk_get_num_pages`

xvt_notebk_get_page_data

Get the Data Associated with a Page and Tab in a Notebk

Summary

```
long xvt_notebk_get_page_data (WINDOW notebk,
    short tab_no, short page_no)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number.

`short page_no`

Page number.

Description

This function gets the data associated with a page and tab in a notebk. Frequently the page data is a pointer to a structure of your own design. In this case, your application should cast the return value from `xvt_notebk_get_page_data` into a pointer of the correct type.

Return Value

`long` integer for application data associated with the page.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control

See Also

`xvt_notebk_set_page_data`
`xvt_notebk_add_page`

---

xvt_notebk_get_page_from_face

Get the Page, Tab, and Notebk Associated with a Specific Face

Summary

```
void xvt_notebk_get_page_from_face (WINDOW face,
    WINDOW * notebk, short * tab_no, short * page_no)
```

`WINDOW face`

Face whose page, tab, and notebk are to be retrieved.

`WINDOW * notebk`

Notebk that face is in.

`short * tab_no`

Tab that face is in.

`short * page_no`

Page that face is in.

Description

This function gets the page, tab, and notebk associated with a specific face.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `face` is `NULL`
- `notebk` is a `NULL` pointer
- `tab_no` is a `NULL` pointer
- `page_no` is a `NULL` pointer

See Also

`xvt_notebk_get_face`

---

xvt_notebk_get_page_title

Get the Page Title in a Notebk for Tab and Page

Summary

```
char * xvt_notebk_get_page_title (WINDOW notebk,
    short tab_no, short  page_no, char * buf, size_t size)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of page.

`short page_no`

Page number for which to get title.

`char * buf`

Buffer to hold title.

`size_t size`

Maximum buffer capacity.

Description

This function gets the page title in a notebk for tab and page. The maximum capacity (including the `NULL`-terminator) is size. The title is truncated as needed to fit into `buf`.

Return Value

Pointer to `buf` if successful; `NULL` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`
- `page_no` is less than `0`
- `buf` is a `NULL` pointer
- `size` is less than or equal to `0`

See Also

`xvt_notebk_get_tab_title`
`xvt_notebk_set_page_title`
`xvt_vobj_get_title`

---

xvt_notebk_get_tab_image

Get the Image for a Tab in a Notebk

Summary

```
XVT_IMAGE xvt_notebk_get_tab_image (WINDOW notebk,
    short tab_no)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number.

Description

This function gets the image for a tab in a notebk.

Return Value

Image displayed in tab if successful; `NULL` if unsuccessful or if tab has no image.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`

Implementation Note

This image belongs to the notebk control. Do not destroy it with `xvt_image_destroy`.

See Also

`xvt_notebk_set_tab_image`

xvt_notebk_get_tab_title

Get the Title for a Tab in a Notebk

Summary

```
char * xvt_notebk_get_tab_title (WINDOW notebk,
    short tab_no, char * buf, size_t size)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number.

`char * buf`

Buffer to hold title.

`size_t size`

Maximum buffer capacity.

Description

This function gets the title for a tab in a notebk. The maximum capacity (including the `NULL`-terminator) is size. The title is truncated as needed to fit into buf.

Return Value

Pointer to `buf` if successful; `NULL` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`
- `buf` is a `NULL` pointer

- `size` is less than or equal to `0`

See Also

```
xvt_notebk_get_page_title
xvt_notebk_set_tab_title
xvt_vobj_get_title
```

xvt_notebk_rem_page

Remove a Page Attached to a Tab from the Notebk

Summary

```
void xvt_notebk_rem_page (WINDOW notebk, short tab_no,
    short page_no)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of page.

`short page_no`

Page number to remove.

Description

This function removes a page attached to a tab from the notebk. The associated face will be destroyed.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`
- `page_no` is less than `0`

See Also

```
xvt_notebk_rem_tab
```

xvt_notebk_rem_tab

Remove a Tab in a Notebk

Summary

```
void xvt_notebk_rem_tab (WINDOW notebk, short tab_no)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number.

Description

This function removes a page from a tab in a notebk. It then removes the tab specified. Each of the faces for the pages attached to the tab will be destroyed.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`

See Also

`xvt_notebk_rem_page`

xvt_notebk_set_page_data

Sets the Data for a Page

Summary

```
void xvt_notebk_set_page_data (WINDOW notebk,
    short tab_no, short page_no, long data)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of page.

`short page_no`

Page number for which to set data.

`long data`

Data to associate with the page.

Description

This function sets the data for a page under the specified tab for a notebk.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control.
- `tab_no` is less than `0`.
- `page_no` is less than `0`.

See Also

`xvt_notebk_get_page_data`

---

xvt_notebk_set_page_title

Set the Title for a Page

Summary

```
void xvt_notebk_set_page_title (WINDOW notebk,
    short tab_no, short page_no, char * title)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of page.

`short page_no`

Page number for which to set title.

`char * title`

Title to be set.

Description

This function sets the title for a page under the specified tab for a notebk.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control.
- `tab_no` is less than `0`.
- `page_no` is less than `0`.

See Also

`xvt_notebk_set_tab_title`

---

xvt_notebk_set_front_page

Set the Front Page

Summary

```
void xvt_notebk_set_front_page (WINDOW notebk,
    short tab_no, short page_no)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of page.

`short page_no`

Page number to set to the front.

Description

This function sets the front page associated with a tab in a notebk. The face associated with the page will have the keyboard input focus.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control.
- `tab_no` is less than `0`.
- `page_no` is less than `0`.

xvt_notebk_set_tab_image

Set the Tab Image

Summary

```
void xvt_notebk_set_tab_image (WINDOW notebk,
    short tab_no, XVT_IMAGE image)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number for which to set image.

`XVT_IMAGE image`

Image to display in tab.

Description

This function duplicates image and replaces the existing tab image for the tab specified in a notebk. The previously set image is destroyed. The application owns image and must destroy it when finished with it.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`
- image is not `NULL`

See Also

`XVT_IMAGE`
`xvt_image_destroy`
`xvt_notebk_add_tab`

xvt_notebk_set_tab_title

Set the Tab Title

Summary

```
void xvt_notebk_set_tab_title (WINDOW notebk,
    short tab_no, char * title)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number for which to set image.

`char * title`

Title of tab. If it is a `NULL_STRING`, the tab will have no title.

Description

This function sets the tab title for the tab specified in a notebk.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`

See Also

`xvt_notebk_add_tab`
`xvt_notebk_set_page_title`

xvt_palet_*

Palette Functions

```
xvt_palet_add_colors
xvt_palet_add_colors_from_image
xvt_palet_create
xvt_palet_default
xvt_palet_destroy
xvt_palet_get_colors
xvt_palet_get_ncolors
xvt_palet_get_size
xvt_palet_get_tolerance
xvt_palet_get_type
xvt_palet_set_tolerance
```

xvt_palet_add_colors

Add Colors to a Palette

Summary

```
short xvt_palet_add_colors(XVT_PALETTE palet,
     COLOR *colorsp, short numcolors)
```

`XVT_PALETTE palet`

Palette to which colors are being added.

`COLOR *colorsp`

Pointer to an array of `COLOR` variables.

`short numcolors`

Number of colors in the array pointed to by `colorsp`.

Description

This function adds colors to a palette of type `XVT_PALETTE_USER`. Any attempt to use this function with other palette types is an error. Only the new color RGB values that are unique (within tolerance) are added to the palette. This function returns the actual number of colors added to the palette. Note that this value is less than or equal to the `numcolors` specified.

Return Value

The actual number of colors added to the palette.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `palet` is `NULL` or invalid
- `palet` is an invalid type (i.e., a type other than `XVT_PALETTE_USER`)
- `colorsp` is `NULL`

See Also

```
COLOR
XVT_PALETTE
XVT_PALLETE_* Values
xvt_palet_set_tolerance
```

The "Portable Images" chapter in the*XVT Portability Toolkit Guide*

Example

See the example for `xvt_palet_create`.

xvt_palet_add_colors_from_image

Add Colors from an Image to a Palette

Summary

```
short xvt_palet_add_colors_from_image
     (XVT_PALETTE palet, XVT_IMAGE image)
```

`XVT_PALETTE palet`

Palette to which colors are being added.

`XVT_IMAGE image`

Image from which to add the colors.

Description

This function adds colors to a palette of type `XVT_PALETTE_USER` that match the colors in a portable image object. Use this function to create a palette that produces the best screen appearance for the image.

Return Value

The number of colors (that were unique within tolerance) added to the palette.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `palet` is `NULL`
- `palet` is an invalid type (i.e. a type other than `XVT_PALETTE_USER`)
- `image` is `NULL` or invalid

See Also

```
XVT_IMAGE
XVT_PALETTE
XVT_PALLETE_* Values
xvt_palet_add_colors
xvt_palet_set_tolerance
```

The "Portable Images" chapter in the*XVT Portability Toolkit Guide*

xvt_palet_create

Create a New Palette

Summary

```
XVT_PALETTE xvt_palet_create(XVT_PALETTE_TYPE type,
     XVT_PALETTE_ATTR reserved)
```

`XVT_PALETTE_TYPE type`

Color type of new palette.

`XVT_PALETTE_ATTR reserved`

Currently not used; pass `NULL` for this parameter.

Description

This function creates a new palette of the specified type.

Return Value

A handle for the created palette if successful; `NULL` if unsuccessful (on error).

See Also

```
XVT_PALETTE_ATTR
XVT_PALLET_TYPE
XVT_PALLETE_* Values
xvt_pallet_*
xvt_vobj_get_palet
xvt_vobj_set_palet
```

The "Portable Images" chapter in the*XVT Portability Toolkit Guide*

Example

This code changes entry `n` of the window's palette to a new color by creating a new palette for the window:

```
WINDOW window;
COLOR colors[MAX_COLORS];
COLOR new_color;
short n;
short num_colors;
XVT_PALETTE palette;
XVT_PALETTE new_palette;
...
/* get colors from window palette */
palette = xvt_vobj_get_palet(window);
num_colors = xvt_palet_get_colors(palette,
     colors, MAX_COLORS);
/* create new palette with changed colors */
new_palette = xvt_palet_create(XVT_PALETTE_USER,
    (XVT_PALETTE_ATTR*)NULL);
if (new_palette)
{
    colors[n] = new_color;
    xvt_palet_set_tolerance(new_palette,
        xvt_palet_get_tolerance(palette));
    xvt_palet_add_colors(new_palette, colors,
        num_colors);
    /* set window palette to new_palette */
    xvt_vobj_set_palet(window, new_palette);
    xvt_palet_destroy(palette);
}
```

xvt_palet_default

Get the Default Palette

Summary

```
XVT_PALETTE xvt_palet_default(void)
```

Description

This function returns a handle to the default palette. The default palette is created for the screen window at application startup time.The palette is of type `XVT_PALETTE_STOCK`.

Return Value

The default palette.

See Also

`XVT_PALETTE`
`XVT_PALLETE_* Values`

The "Portable Images" chapter in the*XVT Portability Toolkit Guide*

---

xvt_palet_destroy

Destroy a Palette

Summary

```
void xvt_palet_destroy(XVT_PALETTE palet)
```

`XVT_PALETTE palet`

Palette to destroy.

Description

This function destroys a palette. The palette is not actually destroyed until its reference count (the number of windows or pixmaps it is associated with) is zero. A palette is retained in memory until it is explicitly destroyed, and until it is no longer associated with any windows or pixmaps.

Note: The default palette cannot be destroyed.

Return Value

None.

Parameter Validity and Error Conditions

XVT issues an error if `palet` is `NULL` or invalid.

See Also

`XVT_PALETTE`
`xvt_palet_create`
`xvt_palet_default`

The "Portable Images" chapter in the*XVT Portability Toolkit Guide*

Example

See the example for `xvt_palet_create`.

xvt_palet_get_colors

Get the Colors in a Palette

Summary

```
short xvt_palet_get_colors(XVT_PALETTE palet,
     COLOR *colorsp, short maxcolors)
```

`XVT_PALETTE palet`

Palette from which to retrieve the colors.

`COLOR *colorsp`

Pointer to an array of `COLOR` variables.

`short maxcolors`

Maximum number of colors in the array pointed to by `colorsp`.

Description

This function gets the colors currently defined in a palette object. The `COLOR` values are returned in the `colorsp` array.

Return Value

The number of colors actually set in the `colorsp` array, which may be less than `maxcolors`.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `palet` is `NULL` or invalid
- `colorsp` is `NULL`

See Also

```
COLOR
XVT_PALETTE
xvt_palet_add_colors
xvt_palet_add_colors_from_image
xvt_palet_create
```

The "Portable Images" chapter in the*XVT Portability Toolkit Guide*

Example

See the example for `xvt_palet_create`.

xvt_palet_get_ncolors

Get the Number of Colors in a Palette

Summary

```
short xvt_palet_get_ncolors(XVT_PALETTE palet)
```

`XVT_PALETTE palet`

Palette from which to retrieve the type.

Description

This function returns the number of colors defined in a palette. For `XVT_PALETTE_USER` type palettes, this number is less than or equal to the size of the palette. For all other palette types, this number is equal to the size of the palette.

Return Value

The number of colors in the palette.

Parameter Validity and Error Conditions

XVT issues an error if `palet` is `NULL` or invalid.

See Also

```
XVT_PALETTE
XVT_PALLETE_* Values
xvt_palet_get_size
```

The "Portable Images" chapter in the*XVT Portability Toolkit Guide*

xvt_palet_get_size

Get the Size of a Palette

Summary

```
short xvt_palet_get_size(XVT_PALETTE palet)
```

`XVT_PALETTE palet`

Palette from which to retrieve the size.

Description

This function returns the "size" of a palette, the maximum number of colors that can be defined in a palette. This number depends on the hardware in use--typical values are 16 and 256.

Return Value

The maximum number of colors that can be defined in the palette.

Parameter Validity and Error Conditions

XVT issues an error if `palet` is `NULL` or invalid.

See Also

`XVT_PALETTE`
`xvt_palet_get_ncolors`

The "Portable Images" chapter in the*XVT Portability Toolkit Guide*

xvt_palet_get_tolerance

Get the Color-Match Tolerance of a Palette

Summary

```
long xvt_palet_get_tolerance(XVT_PALETTE palet)
```

`XVT_PALETTE palet`

Palette from which to retrieve the color-match tolerance.

Description

This function returns the color-match tolerance for a palette object. For more details, see `xvt_palet_set_tolerance`.

Return Value

The color-match tolerance of the palette.

Parameter Validity and Error Conditions

XVT issues an error if `palet` is `NULL` or invalid.

See Also

```
XVT_PALETTE
xvt_palet_create
xvt_palet_set_tolerance
```

The "Portable Images" chapter in the*XVT Portability Toolkit Guide*

Example

See the example for `xvt_palet_create`.

---

xvt_palet_get_type

Get the Type of a Palette

Summary

```
XVT_PALETTE_TYPE xvt_palet_get_type(XVT_PALETTE palet)
```

`XVT_PALETTE palet`

Palette from which to retrieve the type.

Description

This function returns the type of a palette.

Return Value

The palette type.

Parameter Validity and Error Conditions

XVT issues an error if `palet` is `NULL` or invalid.

See Also

```
XVT_PALETTE
XVT_PALLET_TYPE
XVT_PALLETE_* Values
```

The "Portable Images" chapter in the*XVT Portability Toolkit Guide*

xvt_palet_set_tolerance

Set the Color-Match Tolerance of a Palette

Summary

```
void xvt_palet_set_tolerance(XVT_PALETTE palet,
     long tolerance)
```

`XVT_PALETTE palet`

Palette whose tolerance is to be set.

`long tolerance`

The color-matching tolerance.

Description

Sets the color-match tolerance for a palette object. Color-match tolerance is defined as the maximum of the differences between the corresponding RGB components of a given color value and an actual value in the palette.

For example, using the default tolerance of five, any two colors whose RGB components differ by at most five are considered equal within tolerance.

Color tolerance is used to control the "closeness" of colors that are added to `XVT_PALETTE_USER` type palettes. Only colors that are unique (within tolerance) are added to user palettes. Setting tolerance to zero disables any color tolerance checks when new colors are added to a palette.

Return Value

None.

Parameter Validity and Error Conditions

XVT issues an error if `palet` is `NULL` or invalid.

See Also

```
XVT_PALETTE
XVT_PALLETE_* Values
xvt_palet_create
```

The "Portable Images" chapter in the*XVT Portability Toolkit Guide*

Example

See the example for `xvt_palet_create`.

xvt_pattern_*

Complex String Pattern Facility

```
xvt_pattern_create
xvt_pattern_destroy
xvt_pattern_match
xvt_pattern_format_string
```

xvt_pattern_create

Creates an XVT_PATTERN From a Pattern String

Summary

```
XVT_PATTERN xvt_pattern_create (const char *patstr)
```

`const char *patstr`

String describing a Regular Expression pattern.

Description

This function takes a pattern string which defines a Regular Expression pattern, compiles it into a pattern parse tree and returns an `XVT_PATTERN`.

Patterns can be composed of any literal character (single or multibyte), plus the following special symbols.

Character	Meaning
`?`	Match any single character
`#`	Match any digit character
`X`	Match only an alphabetic character
`A`	Match and auto-uppercase alphabetic characters
`a`	Match and auto-lowercase alphabetic characters

Character	Meaning
*	Match 0 or more instances of the previous expression
+	Match 1 or more instances of the previous expression
[...]	Match an optional expression
(...)	Match one of any single character contained in the set
{...}	Match one of the contained, comma-separated strings with auto-casing and optional auto-completion
<...>	Complex expression - treat the contained expression as a single element
\	Literal escape of one character (allows the above characters to be treated as literals)
<all others>	Literal formatting characters to be inserted automatically in the output

The pattern language grammar shows how the symbols in the above table can be combined. The vertical bar '|' signifies "or" and '...' signifies multiple entries. The language grammar follows:

CHAR	Any non-NULL character (international or ASCII)
STRING	Any series of characters except ',' or '}'
LITERAL	CHAR | \CHAR
MATCH	? | # | X | A | a
COMPLEX	<EXPRESSION>
OPTIONAL	[EXPRESSION]
PICKONE	(CHAR...CHAR)
COMPLETE	{STRING,...STRING}
ZEROPLUS	EXPRESSION*
ONEPLUS	EXPRESSION+
EXPRESSION	LITERAL | MATCH | COMPLEX | OPTIONAL |PICKONE | COMPLETE | ZEROPLUS |ONEPLUS [EXPRESSION]

Caveats and Limitations

- () expressions may only contain single character elements such as literals and single-character match elements such as A and #.
- {} expressions contain enumerations of strings, separated by commas.

- All complex expressions like `()`, `[]`, `<>`, and `{}` must be ended with the appropriate matching character.
- All complex expressions must contain at least one element.
- Expressions may be of arbitrary length and complexity, but the strings they filter and match against are limited to a maximum of 256 characters.
- Care must be taken in building expressions. For example, the expression "`?*A`" will never match anything to the "`A`" element because the "`?*`" expression will 'consume' all of the characters in the input string by itself.

Return Value

An `XVT_PATTERN` object is returned if successful; `NULL` if an error occurred.

Parameter and Validity Conditions

XVT returns `NULL` if any of the following errors occur:

- If the `patstr` parameter is `NULL`.
- If `patstr` contains an invalid pattern description.
- There is insufficient memory to create the `XVT_PATTERN` object.

See Also

```
XVT_FORMAT_HANDLER
XVT_PATTERN
xvt_pattern_create
xvt_pattern_match
xvt_pattern_format_string
xvt_vobj_get_formatter
xvt_vobj_set_formatter
```

Examples

Some representative patterns used to accomplish certain described tasks are listed below:

Match any string of arbitrary length:

```
?*
```

These strings will match: "", "a", "any string"

Match any non-null string:

```
?+
```

These strings will match: “a”, “any string”
This string will not match: ““

Match a minimum of 3 characters and a maximum of 8 characters:

???[?][?][?][?][?]

These strings will match: “abc”, “abcd”, ..., “abcdefgh”
These strings will not match: ““, “a”, “ab”, “abcdefghi”

Match an optionally signed integer (notice that the + sign has to be escaped so it’s not treated as an operator):

[(\+-)]#+

These strings will match: “12”, “-1”, “+1”, “-1234”
These strings will not match: ““, “a”, “+a”, “0xFF”

Match negative numbers only:

-#+

These strings will match: “-1”, “-1234”
These strings will not match: ““, “a”, “+a”, “0xFF”, “+1”, “1234”

Match any number of instances of automatically upper-cased letters, each followed by a digit:

<A#>*

These strings will match: ““, “a1”, “b2”, “C3”, “D4”
Note that “a1” will resolve to “A1”
These strings will not match: “a”, “abcde”

Match a 10-digit telephone number with automatically added literals:

“\(###\) ###-####”

These strings will match: “(303) 443-4223”, “3034434223”, “(303)4434223”
Note that “3034434223” will resolve to “(303) 443-4223”

Match a US postal code with optional “Plus four” digits:

#####[-####]

These strings will match: “80301”, “80301-8750”, “803018750”
Note that “803018750” will resolve to “80301-8750”

Match British postal codes with automatically upper-cased letters:

"A[A]#[#] #AA"

Match an optionally signed float with optional 1-3 digit exponent:

[(\+-)]#+[.#+][(\+-)(eE)#[#][#]]

These strings will match: "12", "-12.03", "12.03-e10", "+12.03+E10"

Match a full proper name with an optional middle name and automatically upper-cased where appropriate:

"AX+ A(<X*>.) AX+"

These strings will match: "john t. doe", "john thomas doe", "Jane t. Doe"
Note that "john t doe" will resolve to "John T. Doe"

Match the day-of-the-week abbreviations with automatic completion and casing:

"{Sun,Mon,Tue,Wed,Thu,Fri,Sat}"

These strings will match: "Su", "M", "Wed"
Note that "Su" will resolve (auto-complete) to "Sun" and "M" will resolve to "Mon"

Match a day of the week, with automatic completion and casing (do NOT add unnecessary spaces after commas):

"{Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,
Saturday}"

These strings will match: "Su", "M", "Wed", "Thursday"
Note that "Su" will resolve (auto-complete) to "Sunday", "M" will resolve to "Monday", and "Wed" will resolve to "Wednesday"

Match the time of day:

"{1,2,3,4,5,6,7,8,9,10,11,12}:(012345)# {AM,PM}"

These strings will match: "2:29 AM", "12:08 PM"

Match dates:

"{Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec}
[(123)]#, ####"

These strings will match: "Feb 29, 1996", "Jan 1, 2000"
Note that the date pattern does not assure that the date is correct,

only that the date matches the specified format.
For example, “Feb 31, 1996” would match this pattern.

Match dates in American short format:

“{1,2,3,4,5,6,7,8,9,10,11,12}/(0123)#/##”

These strings will match: “1/23/96”, “11/30/00”

xvt_pattern_destroy

Destroys an XVT_PATTERN and Frees Associated Memory

Summary

```
void xvt_pattern_destroy(XVT_PATTERN pat)
```

`XVT_PATTERN pat`

The `XVT_PATTERN` object to destroy.

Description

This function destroys an `XVT_PATTERN` object and frees all associated memory.

Parameter and Validity Conditions

XVT issues an error if `pat` is `NULL` or is not a valid `XVT_PATTERN`.

See Also

`XVT_FORMAT_HANDLER`
`XVT_PATTERN`
`xvt_pattern_create`
`xvt_pattern_match`
`xvt_pattern_format_string`
`xvt_vobj_get_formatter`
`xvt_vobj_set_formatter`

xvt_pattern_match

Matches a String Against an XVT_PATTERN

Summary

```
BOOLEAN xvt_pattern_match(XVT_PATTERN pat,
    const char *str, const char** endstr)
```

`XVT_PATTERN pat`

Pattern parse tree object.

`const char *str`

String to match against pattern.

`const char** endst`

Set to return a pointer into the string `str` to the character immediately following the last matched character.

Description

This function matches the string `str` to the pattern described by the `XVT_PATTERN`, `pat`, and sets `endstr` to point to the character immediately following the last used character in the match.

Return Value

The function returns:

`TRUE` and `*endstr == '\0'`
: if the string matched the pattern exactly. The end of both the string and the pattern was reached.

`FALSE` and `*endstr == '\0'`
: if the string matched the pattern but not the complete pattern. The end of the string was reached but not the end of the pattern.

`FALSE` and `*endstr != '\0'`
: if the string did not match the pattern. Neither the end of the string nor the end of the pattern was reached.

Parameter and Validity Conditions

XVT returns `FALSE` and issues an error if any of the following conditions occur:

- If any of the parameters is `NULL`.

- If `pat` is not a valid `XVT_PATTERN`.
- There is insufficient memory to process the pattern match.

See Also

```
XVT_FORMAT_HANDLER
XVT_PATTERN
xvt_pattern_create
xvt_pattern_match
xvt_pattern_format_string
xvt_vobj_get_formatter
xvt_vobj_set_formatter
```

xvt_pattern_format_string

Matches and Transforms a String According to an XVT_PATTERN

Summary

```
char *xvt_pattern_format_string(XVT_PATTERN pat,
    const char *str, char *buf, size_t buflen,
    BOOLEAN complete_string, int *start, int *end)
```

`XVT_PATTERN pat`

Pattern parse tree object.

`const char *str`

String to match against pattern.

`char *buf`

Return buffer for completed string.

`size_t buflen`

Length of string buffer `buf` in bytes.

`BOOLEAN complete_string`

Determines if auto-completion clauses in the pattern are filled out.

`int *start`

Set to the start of the selection range for the completion clause.

`int *end`

Set to the end of the selection range for the completion clause.

Description

This function traverses the string `str` and matches and transforms the string into the pattern into the string buffer `buf` of length `buflen`. If `complete_string` is `TRUE`, the string is matched to uniqueness and the string filled out completely wherever there is an auto-completion clause "`{}`". In this case, `*start` and `*end` will be set to the range of characters that the formatter automatically inserted for the user (which should be used to set the selection range in an edit control). When `complete_string` is `FALSE`, `*start` and `*end` reflect the cursor's position (placed at the end of the formatted string), and auto-completion clauses will only restrict the characters entered instead of automatically completing the string.

Return Value

The function returns a pointer into the string `str` at the point at which it stopped formatting or `NULL` if an error occurred.

Parameter and Validity Conditions

XVT returns `NULL` and issues an error if any of the following conditions occur:

- If any of the parameters is `NULL`.
- If `pat` is not a valid `XVT_PATTERN`.
- There is insufficient memory to process the pattern match.

See Also

```
XVT_FORMAT_HANDLER
XVT_PATTERN
xvt_pattern_create
xvt_pattern_match
xvt_pattern_format_string
xvt_vobj_get_formatter
xvt_vobj_set_formatter
```

xvt_pict_*

Picture Objects

```
xvt_pict_create
xvt_pict_destroy
xvt_pict_lock
xvt_pict_unlock
```

xvt_pict_create

Make Encapsulated Picture from Data

Summary

```
PICTURE xvt_pict_create(char *buf, long nbytes,
     RCT *rctp)
```

`char *buf`

Picture data.

`long nbytes`

Number of bytes in the picture data.

`RCT *rctp`

Returned pointer to the bounding rectangle.

Description

This function recreates a `PICTURE` from a group of sequential bytes of length `nbytes` that were originally formed via a call to `xvt_pict_lock`. The original frame rectangle is returned through the pointer `rctp`.

You can also use `xvt_cb_get_data` to get data from the clipboard for creating a `PICTURE`.

When you are finished using it, you should free the returned `PICTURE` with a call to `xvt_pict_destroy`.

Return Value

A `PICTURE` if successful; `NULL_PICTURE` if unsuccessful (on error).

See Also

`xvt_cb_get_data`
`xvt_pict_destroy`
`xvt_pict_lock`

The "Drawing and Pictures" chapter in the*XVT Portability Toolkit Guide*

Example

This code gets a `PICTURE` from the clipboard and draws it in the upper-left corner of the window:

```
WINDOW window;
...
/* open clipboard */
if (xvt_cb_open(FALSE)) {
     char *data;
    long size;
    /* get PICTURE from clipboard and close */
    data = xvt_cb_get_data(CB_PICT, NULL, &size);
    xvt_cb_close();
    if (data) {
        PICTURE pict;
        RCT rect;
        /* create PICTURE from clipboard data */
        pict = xvt_pict_create(data, size, &rect);
        if (pict) {
            /* draw picture to window */
            xvt_dwin_draw_pict(window, pict, &rect);
            xvt_pict_destroy(pict);
        }
} else /* xvt_cb_get_data failed */
        xvt_dm_post_warning(
            "No PICTURE data in clipboard");
}
```

xvt_pict_destroy

Free Encapsulated Picture

Summary

```
void xvt_pict_destroy(PICTURE pic)
```

`PICTURE pic`

Picture to be freed.

Description

This function frees the memory occupied by a `PICTURE` object. The `PICTURE` may have been obtained from a call to `xvt_dwin_close_pict` or `xvt_pict_create`.

`PICTURE`s are not necessarily freed automatically when your application terminates. It is best to free them explicitly (in your `E_DESTROY` case of the task event handler if necessary).

See Also

`PICTURE`
`xvt_dwin_close_pict`
`xvt_pict_create`

The "Drawing and Pictures" chapter in the*XVT Portability Toolkit Guide*

Example

See the example for `xvt_pict_create`.

xvt_pict_lock

Get Pointer to Encapsulated Picture

Summary

```
char *xvt_pict_lock(PICTURE pic, long *sizep)
```

`PICTURE pic`

Picture that is to be transformed into bytes.

`long *sizep`

The size of the picture in bytes.

Description

This function transforms a `PICTURE` into an unstructured sequence of bytes, the length of which is stored into the `long` pointed to by `sizep`. The bytes *cannot* be interpreted or manipulated by your application, but they can be written to a file, read back in later, and transformed back into a `PICTURE` with `xvt_pict_create`. The returned character stream is owned by this function, and the application should `not` attempt to free it.

When you are done with the pointer returned by `xvt_pict_lock`, you must call `xvt_pict_unlock`. Don't keep the `PICTURE` locked for longer than you have to. If you need it for a long time, copy it to a block of memory that your application has allocated (with the `xvt_*_alloc` functions).

Implementation Note

The sequence of bytes returned from `xvt_pict_lock` is not in a portable format. Therefore, even if you write it to a file, you cannot transfer it to a platform different from its creator. You might not be able to move the file to a different computer running the same window system because this sequence of bytes is display-driver dependent.

Return Value

A character pointer to the bytes if successful; `NULL` on error (usually out of memory).

See Also

`xvt_gmem_alloc`
`xvt_mem_alloc`
`xvt_pict_create`
`xvt_pict_unlock`

The "Drawing and Pictures" chapter in the*XVT Portability Toolkit Guide*

xvt_pict_unlock

Unlock Picture

Summary

```
void xvt_pict_unlock(PICTURE pic)
```

`PICTURE pic`

Picture to be unlocked.

Description

This function unlocks a `PICTURE` that was previously locked with a call to `xvt_pict_lock`. After you have accessed its data, you should unlock a locked `PICTURE` as soon as possible.

See Also

`xvt_pict_lock`

The "Drawing and Pictures" chapter in the*XVT Portability Toolkit Guide*

xvt_pmap_*

Pixmap Objects

`xvt_pmap_create`
`xvt_pmap_destroy`

xvt_pmap_create

Create a New Pixmap

Summary

```
XVT_PIXMAP xvt_pmap_create(WINDOW parent,
     XVT_PIXMAP_FORMAT format, short width,
    short height, XVT_PIXMAP_ATTR reserved)
```

`WINDOW parent`

Parent window.

`XVT_PIXMAP_FORMAT format`

Format for the new pixmap. Currently, this must be `XVT_PIXMAP_DEFAULT`.

`short width, short height`

Width and height of the new pixmap, in pixels.

`XVT_PIXMAP_ATTR reserved`

Not currently used; pass `NULL` for this parameter.

Description

This function allocates memory and creates a pixmap. The pixmap's contents are not initialized; if you need to initialize the contents, call `xvt_dwin_clear`.

The pixmap's format matches the screen, and has a default color palette. The following XVT functions accept a pixmap in addition to a window:

```
xvt_dwin_clear
xvt_dwin_draw_aline
xvt_dwin_draw_arc
xvt_dwin_draw_icon
xvt_dwin_draw_image
xvt_dwin_draw_line
xvt_dwin_draw_oval
xvt_dwin_draw_pic
xvt_dwin_draw_pie
xvt_dwin_draw_pmap
xvt_dwin_draw_polygon
xvt_dwin_draw_polyline
xvt_dwin_draw_rect
xvt_dwin_draw_roundrect
xvt_dwin_draw_set_pos
xvt_dwin_draw_text
xvt_dwin_get_draw_ctools
xvt_dwin_get_font_metrics
xvt_dwin_get_text_width
xvt_dwin_scroll_rect
xvt_dwin_set_back_color
xvt_dwin_set_cbrush
xvt_dwin_set_clip
xvt_dwin_set_cpen
xvt_dwin_set_draw_ctools
xvt_dwin_set_draw_mode
xvt_dwin_set_font
xvt_dwin_set_fore_color
xvt_dwin_set_std_cbrush
xvt_dwin_set_std_cpen
xvt_vobj_get_client_rect
xvt_vobj_get_data
xvt_vobj_get_outer_rect
xvt_vobj_get_parent
xvt_vobj_get_type
xvt_vobj_set_data
```

Return Value

A valid `XVT_PIXMAP` if successful; `NULL_WIN` if unsuccessful (on error).

Parameter Validity and Error Conditions

This function returns `NULL_WIN` if any of the following conditions are true:

- `width` or `height` is less than zero
- The pixmap would exceed system limitations

See Also

XVT_PIXMAP_ATTR
XVT_PIXMAP
XVT_PIXMAP_FORMAT
xvt_dwin_*
xvt_image_create
xvt_pmap_destroy
xvt_vobj_get_client_rect
xvt_vobj_get_data
xvt_vobj_get_outer_rect
xvt_vobj_get_parent
xvt_vobj_get_type
xvt_vobj_set_data

The "Portable Images" chapter in the*XVT Portability Toolkit Guide*

Example

This code performs drawing operations on an off-screen XVT_PIXMAP, then copies them to the WINDOW:

```
WINDOW window;
XVT_PIXMAP pmap;
CBRUSH brush;
RCT &rect;
...
/* create pixmap for off-screen drawing */
pmap = xvt_pmap_create(window, XVT_PIXMAP_DEFAULT,
     50, 50, (XVT_PIXMAP_ATTR) NULL);
/* perform drawing operations on pixmap */
xvt_vobj_get_client_rect(pmap, &rect);
brush.pat = PAT_SOLID;
brush.color = COLOR_RED;
xvt_dwin_set_cbrush(pmap, &brush);
xvt_dwin_draw_rect(pmap, &rect);
brush.color = COLOR_GREEN;
xvt_dwin_set_cbrush(pmap, &brush);
xvt_dwin_draw_oval(pmap, &rect);
...
/* copy pixmap contents to window */
xvt_dwin_draw_pmap(window, pmap, &rect, &rect);
/* destroy pixmap */
xvt_pmap_destroy(pmap);
```

xvt_pmap_destroy

Destroy a Pixmap

Summary

```
void xvt_pmap_destroy(XVT_PIXMAP pmap)
```

`XVT_PIXMAP pmap`

Pixmap to destroy.

Description

This function destroys a pixmap and frees all memory used by it. Once this function returns, you should not attempt to use the window any longer (not even to call `xvt_vobj_get_data`).

Note: Do not use `xvt_mem_free` to free memory used by a pixmap; instead, use `xvt_pmap_destroy`.

See Also

`XVT_PIXMAP`
`xvt_pmap_create`

The "Portable Images" chapter in the*XVT Portability Toolkit Guide*

Example

See the example for `xvt_pmap_create`.

xvt_notebk_*

Notebook Functions

```
xvt_notebk_add_page
xvt_notebk_add_tab
xvt_notebk_create_face
xvt_notebk_create_face_def
xvt_notebk_create_face_res
xvt_notebk_enum_pages
xvt_notebk_get_face
xvt_notebk_get_front_page
xvt_notebk_get_num_pages
xvt_notebk_get_num_tabs
xvt_notebk_get_page_data
xvt_notebk_get_page_from_face
xvt_notebk_get_page_title
xvt_notebk_get_tab_image
xvt_notebk_get_tab_title
xvt_notebk_rem_page
xvt_notebk_rem_tab
xvt_notebk_set_page_data
xvt_notebk_set_page_title
xvt_notebk_set_front_page
xvt_notebk_set_tab_image
xvt_notebk_set_tab_title
```

xvt_notebk_add_page

Add a Page to a Specific Tab in a Notebook Control

Summary

```
void xvt_notebk_add_page (WINDOW notebk, short tab_no,
    short page_no, char * title, long page_data)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of tab to which to add a page.

`short page_no`

Position at which to add new page. If 0, it will be the first page. The page is placed before the page whose index is equal to

`page_no` (origin `0`). A `page_no` that is too large causes the page to be added at the end.

`char * title`

Title of page.

`long page_data`

Contains any application data you wish to attach to a page. Typically, this will be a pointer to some structure allocated from the heap, cast into a `long` so that, later, your application can retrieve the structure and look at it.

Description

This function adds a page to a notebook tab.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- notebk is `NULL` or not a valid notebook control
- `tab_no` is less than `0`
- `page_no` is less than `0`

Implementation Note

The multiple page mechanism allows one tab to have many pages. Each page has a `Face (XVT WINDOW)`. You must come up with a way for the user to change pages for tabs with more than one page. For example, you may provide "Next" and "Prev" buttons on each face for tabs with multiple pages.

See Also

```
xvt_notebk_add_tab
xvt_notebk_create_face
xvt_notebk_create_face_def
xvt_notebk_create_face_res
xvt_notebk_get_page_data
xvt_notebk_set_page_data
```

xvt_notebk_add_tab

Add a Tab to a Notebook Control

Summary

```
void xvt_notebk_add_tab (WINDOW notebk, short tab_no,
    char * title, XVT_IMAGE image)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of tab to add;Position at which to add new tab. If 0, this will be the first tab. The tab is placed before the tab whose index is equal to `tab_no` (origin `0`). A `tab_no` that is too large causes the tab to be added at the end.

`char * title`

Title of tab. If it is a `NULL_STRING`, the tab will have no title.

`XVT_IMAGE image`

Image to display in tab. If it is a `NULL_IMAGE`, the tab will have no image.

Description

This function adds a tab to a notebook control. Each tab is analogous to a divider in a notebook.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`

Implementation Note

A tab may have multiple pages. The image is duplicated. The application owns image and must destroy it when finished with it.

See Also

```
XVT_IMAGE
xvt_image_destroy
xvt_notebk_add_page
```

Example

```
anImage =  xvt_image_read_bmp("window.bmp");
xvt_notebk_add_tab (aNotebk, 0, "Window", anImage);
xvt_notebk_add_page(aNotebk, 0, 0, "Page0",0L);
aFace = xvt_res_get_win_def(WINDOW_FACE);
xvt_notebk_create_face_def(aNotebk, 0, 0, aFace, EM_ALL,
    WINDOW_FACE_eh, PTR_LONG(&aData->winFace));
xvt_res_free_win_def(aFace);
xvt_image_destroy(anImage);
```

xvt_notebk_create_face

Create a Face for a Page

Summary

```
WINDOW xvt_notebk_create_face (WINDOW notebk,
    short tab_no, short page_no, EVENT_MASK mask,
    EVENT_HANDLER face_eh, long app_data)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of page to which face is being added.

`short page_no`

Page number for which to create face.

`EVENT_MASK mask`

Specifies which events should be sent to the window handler. This is an `OR`'d combination of any of the `EM_*` constants. You usually set `EVENT_MASK mask` to `EM_ALL` indicating that all events should be sent to the window (no restriction). In some conditions, you can restrict the events sent to the window. For more details, see the "Event Masking" section of the "Events" chapter in the *XVT Portability Toolkit Guide*.

`EVENT_HANDLER face_eh`

The event handler function; it receives all of the events for the window.

`long app_data`

Contains any application data you wish to attach to the window when it is created. Normally, it is a pointer to a data structure cast into a `long`.

Description

This function creates a face for the page identified by `page_no`. A face is simply an XVT child WINDOW. There is a one to one relationship between a page and a face. The face is displayed when the page it is associated with is selected.

Return Value

A WINDOW if successful; `NULL_WIN` if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`
- `page_no` is less than `0`

See Also

`xvt_notebk_create_face_def`
`xvt_notebk_create_face_res`

---

xvt_notebk_create_face_def

Create a Face with Controls from an Array of Data Structures

Summary

```
WINDOW xvt_notebk_create_face_def (WINDOW notebk,
    short tab_no, short page_no, WIN_DEF * win_def_p,
    EVENT_MASK  mask, EVENT_HANDLER face_eh,
    long app_data)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of page to which face is being added.

`short page_no`

Page number for which to create face.

`WIN_DEF * win_def_p`

Points to an array of data structures. The first element in the array defines the window itself. Subsequent elements of the array define the controls or text edit objects contained within the window. The last element of the array is a terminator whose `wtype` field is set to `W_NONE`. `EVENT_MASK mask` specifies which events are sent to the window event handler. This is an `OR`'d combination of any of the `EM_*` constants. You usually set this to `EM_ALL` indicating that all events would be sent to the window. For more details, see the "Event Masking" section of the "Events" chapter in the *XVT Portability Toolkit Guide*.

`EVENT_HANDLER face_eh`

The event handler function; it receives all of the events for the window.

`long app_data`

Contains any application data you wish to attach to the window when it is created. Normally, it is a pointer to a data structure cast into a `long`.

Description

This function creates a face for the page identified by `page_no`. A face is simply an XVT child WINDOW. There is a one to one relationship between a page and a face. The face is displayed when the page it is associated with is selected. For more information on `WIN_DEF` structures see `xvt_win_create_def`.

Return Value

A WINDOW if successful; `NULL_WIN` if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control.
- `tab_no` is less than `0`.
- `page_no` is less than `0`.

See Also

`xvt_notebk_create_face`
`xvt_notebk_create_face_res`
`xvt_win_create_def`

xvt_notebk_create_face_res

Create a Face from a Resource File

Summary

```
WINDOW xvt_notebk_create_face_res (WINDOW notebk,
    short tab_no, short page_no, int rid, EVENT_MASK mask,
    EVENT_HANDLER face_eh, long app_data)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of page to which face is being added.

`short page_no`

Page number for which to create face.

`int rid`

Resource ID corresponding to a window statement in your **URL** resource file. The face is created as if this resource were loaded via `xvt_res_get_win_def`, and then instantiated via `xvt_notebk_create_face_def`.

`EVENT_MASK mask`

Specifies which events are sent to the window event handler. This is an `OR`'d combination of any of the `EM_*` constants. You usually set this to `EM_ALL` indicating that all events would be sent to the window. For more details, see the "Event Masking" section of the "Events" chapter in the *XVT Portability Toolkit Guide*.

`EVENT_HANDLER face_eh`

The event handler function; it receives all of the events for the window.

`long app_data`

Contains any application data you wish to attach to the window when it is created. Normally, it is a pointer to a data structure cast into a `long`.

Description

This function creates a face for the page identified by `page_no`. A face is simply an XVT child WINDOW. There is a one to one

relationship between a page and a face. The face is displayed when the page it is associated with is selected.

Return Value

A WINDOW if successful; `NULL_WIN` if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control.
- `tab_no` is less than `0`.
- `page_no` is less than `0`.
- `face_eh` must be set to a valid function pointer.
- `rid` specifies a window resource in the **URL** file.

See Also

`xvt_notebk_create_face`
`xvt_notebk_create_face_def`

xvt_notebk_enum_pages

Enumerate through All Pages and Apply the Function to Each Page

Summary

```
BOOLEAN xvt_notebk_enum_pages (WINDOW notebk,
    XVT_NOTEBK_ENUM_PAGES func, long data)
```

`WINDOW notebk`

Window ID of notebook control.

`XVT_NOTEBK_ENUM_PAGES func`

Address of function to be called for each page.

`long data`

Application-defined data to pass to callback function.

Description

This function enumerates (i.e., invokes an application-supplied callback function) the pages contained in the specified tab. It passes the `notebk`, `tab_no`, `page_no` and `page_data` of each page, in sequential order, to an application-defined callback function. It

continues until the last page is enumerated or until the callback function returns `FALSE`.

Return Value

`TRUE` if successful; `FALSE` otherwise.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control.
- `tab_no` is less than `0`.
- `page_no` is less than `0`.

See Also

`XVT_NOTEBK_ENUM_PAGES`

---

xvt_notebk_get_face

Get the Face in the Notebk at Tab and Page

Summary

```
WINDOW xvt_notebk_get_face (WINDOW notebk, short tab_no,
    short page_no)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of face.

`short page_no`

Page number of face.

Description

This function gets the face (`XVT WINDOW`) in the notebk at tab and page.

Return Value

The WINDOW of the face if successful, or `NULL_WIN` if no such face exists.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`
- `page_no` is less than `0`

See Also

`xvt_notebk_get_front_page`

---

xvt_notebk_get_front_page

Get the Current Front Page

Summary

```
WINDOW xvt_notebk_get_front_page (WINDOW  notebk,
    short * tab_no, short * page_no)
```

`WINDOW  notebk`

Window ID of notebook control.

`short * tab_no`

Tab number of tab with the front page.

`short * page_no`

Page number of front page.

Description

This function gets the current front page. This is the page that is currently on top and showing. It also returns the face (`WINDOW`) of the front page.

Return Value

The WINDOW of the face if successful, or `NULL_WIN` unsuccessful. `tab_no` and/or `page_no` may be set to `-1` on error.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is a `NULL` pointer
- `page_no` is a `NULL` pointer

See Also

xvt_notebk_get_face

xvt_notebk_get_num_pages

Get the Number of Pages in the Specified Tab

Summary

```
short xvt_notebk_get_num_pages (WINDOW notebk, short
tab_no)
```

`WINDOW  notebk`

Window ID of notebook control.

`short * tab_no`

Tab number for which to get the number of pages.

Description

This function gets the number of pages in a notebk at the tab specified.

Return Value

The number of pages in the tab if successful, or `0` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control.
- `tab_no` is less than `0`.

See Also

xvt_notebk_get_num_tabs

xvt_notebk_get_num_tabs

Get the Number of Tabs in a Notebk

Summary

```
short xvt_notebk_get_num_tabs (WINDOW notebk)
```

`WINDOW  notebk`

Window ID of notebook control.

Description

This function gets the number of tabs in a notebk.

Return Value

The number of pages in the tab if successful, or `0` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control

See Also

`xvt_notebk_get_num_pages`

xvt_notebk_get_page_data

Get the Data Associated with a Page and Tab in a Notebk

Summary

```
long xvt_notebk_get_page_data (WINDOW notebk,
    short tab_no, short page_no)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number.

`short page_no`

Page number.

Description

This function gets the data associated with a page and tab in a notebk. Frequently the page data is a pointer to a structure of your own design. In this case, your application should cast the return value from `xvt_notebk_get_page_data` into a pointer of the correct type.

Return Value

`long` integer for application data associated with the page.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control

See Also

`xvt_notebk_set_page_data`
`xvt_notebk_add_page`

---

xvt_notebk_get_page_from_face

Get the Page, Tab, and Notebk Associated with a Specific Face

Summary

```
void xvt_notebk_get_page_from_face (WINDOW face, WINDOW *
notebk, short * tab_no, short * page_no)
```

`WINDOW face`

Face whose page, tab, and notebk are to be retrieved.

`WINDOW * notebk`

Notebk that face is in.

`short * tab_no`

Tab that face is in.

`short * page_no`

Page that face is in.

Description

This function gets the page, tab, and notebk associated with a specific face.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `face` is `NULL`
- `notebk` is a `NULL` pointer
- `tab_no` is a `NULL` pointer
- `page_no` is a `NULL` pointer

See Also

`xvt_notebk_get_face`

---

xvt_notebk_get_page_title

Get the Page Title in a Notebk for Tab and Page

Summary

```
char * xvt_notebk_get_page_title (WINDOW notebk, short
tab_no, short  page_no, char * buf, size_t size)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of page.

`short page_no`

Page number for which to get title.

`char * buf`

Buffer to hold title.

`size_t size`

Maximum buffer capacity.

Description

This function gets the page title in a notebk for tab and page. The maximum capacity (including the `NULL`-terminator) is size. The title is truncated as needed to fit into `buf`.

Return Value

Pointer to `buf` if successful; `NULL` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`
- `page_no` is less than `0`
- `buf` is a `NULL` pointer
- `size` is less than or equal to `0`

See Also

`xvt_notebk_get_tab_title`
`xvt_notebk_set_page_title`
`xvt_vobj_get_title`

---

xvt_notebk_get_tab_image

Get the Image for a Tab in a Notebk

Summary

```
XVT_IMAGE xvt_notebk_get_tab_image (WINDOW notebk,
    short  tab_no)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number.

Description

This function gets the image for a tab in a notebk.

Return Value

Image displayed in tab if successful; `NULL` if unsuccessful or if tab has no image.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`

Implementation Note

This image belongs to the notebk control. Do not destroy it with `xvt_image_destroy`.

See Also

`xvt_notebk_set_tab_image`

xvt_notebk_get_tab_title

Get the Title for a Tab in a Notebk

Summary

```
char * xvt_notebk_get_tab_title (WINDOW notebk,
    short  tab_no, char * buf, size_t size)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number.

`char * buf`

Buffer to hold title.

`size_t size`

Maximum buffer capacity.

Description

This function gets the title for a tab in a notebk. The maximum capacity (including the `NULL`-terminator) is size. The title is truncated as needed to fit into buf.

Return Value

Pointer to `buf` if successful; `NULL` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`
- `buf` is a `NULL` pointer

- `size` is less than or equal to `0`

See Also

```
xvt_notebk_get_page_title
xvt_notebk_set_tab_title
xvt_vobj_get_title
```

xvt_notebk_rem_page

Remove a Page Attached to a Tab from the Notebk

Summary

```
void xvt_notebk_rem_page (WINDOW notebk, short  tab_no,
short  page_no)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of page.

`short page_no`

Page number to remove.

Description

This function removes a page attached to a tab from the notebk. The associated face will be destroyed.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`
- `page_no` is less than `0`

See Also

```
xvt_notebk_rem_tab
```

xvt_notebk_rem_tab

Remove a Tab in a Notebk

Summary

```
void xvt_notebk_rem_tab (WINDOW notebk, short tab_no)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number.

Description

This function removes a page from a tab in a notebk. It then removes the tab specified. Each of the faces for the pages attached to the tab will be destroyed.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`

See Also

`xvt_notebk_rem_page`

xvt_notebk_set_page_data

Sets the Data for a Page

Summary

```
void xvt_notebk_set_page_data (WINDOW notebk, short
tab_no, short page_no, long data)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of page.

`short page_no`

Page number for which to set data.

`long data`

Data to associate with the page.

Description

This function sets the data for a page under the specified tab for a notebk.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control.
- `tab_no` is less than `0`.
- `page_no` is less than `0`.

See Also

`xvt_notebk_get_page_data`

---

xvt_notebk_set_page_title

Set the Title for a Page

Summary

```
void xvt_notebk_set_page_title (WINDOW notebk, short
tab_no, short page_no, char * title)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of page.

`short page_no`

Page number for which to set title.

`char * title`

Title to be set.

Description

This function sets the title for a page under the specified tab for a notebk.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control.
- `tab_no` is less than `0`.
- `page_no` is less than `0`.

See Also

`xvt_notebk_set_tab_title`

---

xvt_notebk_set_front_page

Set the Front Page

Summary

```
void xvt_notebk_set_front_page (WINDOW notebk, short
tab_no, short page_no)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number of page.

`short page_no`

Page number to set to the front.

Description

This function sets the front page associated with a tab in a notebk. The face associated with the page will have the keyboard input focus.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control.
- `tab_no` is less than `0`.
- `page_no` is less than `0`.

xvt_notebk_set_tab_image

Set the Tab Image

Summary

```
void xvt_notebk_set_tab_image (WINDOW notebk, short
tab_no, XVT_IMAGE image)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number for which to set image.

`XVT_IMAGE image`

Image to display in tab.

Description

This function duplicates image and replaces the existing tab image for the tab specified in a notebk. The previously set image is destroyed. The application owns image and must destroy it when finished with it.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`
- image is not `NULL`

See Also

`XVT_IMAGE`
`xvt_image_destroy`
`xvt_notebk_add_tab`

xvt_notebk_set_tab_title

Set the Tab Title

Summary

```
void xvt_notebk_set_tab_title (WINDOW notebk, short
tab_no, char * title)
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number for which to set image.

`char * title`

Title of tab. If it is a `NULL_STRING`, the tab will have no title.

Description

This function sets the tab title for the tab specified in a notebk.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `notebk` is `NULL` or not a valid notebook control
- `tab_no` is less than `0`

See Also

`xvt_notebk_add_tab`
`xvt_notebk_set_page_title`

xvt_print_*

Printing Functions

```
xvt_print_close
xvt_print_close_page
xvt_print_create
xvt_print_create_win
xvt_print_destroy
xvt_print_get_next_band
xvt_print_is_valid
xvt_print_open
xvt_print_open_page
xvt_print_set_page_orient
xvt_print_set_page_size
xvt_print_start_thread
```

xvt_print_close

Terminate Printing Manager

Summary

```
void xvt_print_close(void)
```

Description

This function, called after `xvt_print_open`, closes the printing manager, allowing other applications to print. Both `xvt_print_open` and `xvt_print_close` are normally called automatically by XVT's printing functions. However, if your application has successfully called `xvt_print_open`, it should also call `xvt_print_close`.

See Also

`xvt_print_open`

"Printing" chapter in the*XVT Portability Toolkit Guide*

Example

See the example for `xvt_print_open`.

xvt_print_close_page

Finish Printer Page

Summary

```
BOOLEAN xvt_print_close_page(PRINT_RCD *precp)
```

`PRINT_RCD *precp`

Pointer to the print record.

Description

This function must be called in a printing loop to finish printing the current page. In a typical XVT printing loop, this function is called when drawing to all bands of the page is complete. Normally, you should call this function when `xvt_print_get_next_band` returns `NULL`, then call `xvt_print_open_page` to start printing the next page.

`precp` points to a `PRINT_RCD`, which holds information about the current print job, such as the page margins. Before the printing loop begins, you must initialize the `PRINT_RCD` with `xvt_print_create`. If you have an existing print record, you can check its validity with `xvt_print_is_valid`.

If `xvt_print_close_page` returns `FALSE`, the print job has been aborted, either by the user or because an error occurred. If this happens, you should not abandon the printing loop, but should end it gracefully by immediately calling `xvt_vobj_destroy(print_win)`. Be careful to call `xvt_vobj_destroy` only within the print function whose address was passed to `xvt_print_start_thread`.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful (on error).

See Also

```
PRINT_RCD
xvt_print_get_next_band
xvt_print_open_page
xvt_print_start_thread
xvt_vobj_destroy
```

Example

See the example for `xvt_print_start_thread`.

xvt_print_create

Get Printing Record

Summary

```
PRINT_RCD *xvt_print_create(int *sizep)
```

`int *sizep`

Size of the print record.

Description

This function allocates a "print record" of type `PRINT_RCD`. A print record is an object that XVT uses to hold information about a page setup and a print job. The internals of a print record are not available to applications.

`sizep` is a pointer to an integer variable used to store the size of the print record in bytes. When `xvt_print_create` returns, your application should use this size in operations such as writing the print record to disk or reading it back. Do not use the standard C `sizeof` operator to determine the size of a print record because the `PRINT_RCD` definition that appears in the XVT header file is a fictitious data type.

After using the `PRINT_RCD` returned by this function, you must free it with a call to `xvt_print_destroy`.

Once you have a print record, you can allow the user to change its internal settings by calling `xvt_dm_post_page_setup`. In addition, you can save the print record in a disk file and read it back later. If you choose to do this, you must call `xvt_print_is_valid` to check if the stored print record is valid for the current system configuration.

Return Value

A pointer to a `PRINT_RCD` if successful; `NULL` if `PRINT_RCD` cannot be allocated.

Parameter Validity and Error Conditions

If the `PRINT_RCD` cannot be allocated, XVT issues an error alert for the user’s benefit.

Implementation Note

The internal format and size of print records varies between XVT platforms, so a PRINT_RCD object cannot be passed between them (in a document file, for example).

See Also

PRINT_RCD
xvt_dm_post_page_setup
xvt_print_destroy
xvt_print_is_valid

xvt_print_create_win

Create Printing Window

Summary

```
WINDOW xvt_print_create_win(PRINT_RCD *precp,
     char *title)
```

PRINT_RCD *precp

Print record.

char *title

Optional print job title.

Description

This function begins a print job by returning the WINDOW onto which printed output should be drawn. Conceptually, a print window represents the printable area of a page of paper. No events are ever generated for it. It has a complete set of drawing tools, but no cursor or caret. The only drawing mode that's supported is M_COPY.

precp must point to a PRINT_RCD that's valid for the current printer. For more information, see the topics PRINT_RCD, xvt_print_create, and xvt_print_is_valid.

If there is an appropriate name for the print job, set title to point to the NULL-terminated name. title can be used in dialog boxes and print-spooler displays to identify the job. If there is no appropriate name, set title to the empty string (""). title must not be NULL.

After the print window is successfully opened, use the functions xvt_print_open_page, xvt_print_get_next_band, and xvt_print_close_page to control the banding and paging of the

print job. When the print job is completed (perhaps after several pages are output), the print window should be closed with a call to `xvt_vobj_destroy`.

These are the limited set of functions that accept print windows:

the drawing and tool-manipulation functions
`xvt_dwin_is_update_needed`
`xvt_vobj_destroy`
`xvt_vobj_get_client_rect`
`xvt_vobj_get_data`
`xvt_vobj_get_outer_rect`
`xvt_vobj_get_type`
`xvt_vobj_set_data`

Note: You should only call `xvt_print_create_win` within the print function whose address was passed to `xvt_print_start_thread`.

Return Value

A `WINDOW` object that identifies the print window if successful; `NULL_WIN` if the window can't be created.

Parameter Validity and Error Conditions

XVT returns `NULL_WIN` in one of two cases:

- An error could have occurred in the printing driver. In that case, XVT will have already put up an error dialog. Your application should not put up its own error dialog because doing so is illegal in the print thread.
- The user might have cancelled the printing operation before the print window was created. In that case, your application should not put up an error dialog because the user already knows that he or she cancelled the operation.

See Also

`NULL_WIN`
`PRINT_RCD`
`xvt_print_close_page`
`xvt_print_close`
`xvt_print_create`
`xvt_print_get_next_band`
`xvt_print_is_valid`
`xvt_print_open_page`
`xvt_print_start_thread`
`xvt_vobj_destroy`

The "Printing" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_print_start_thread`.

xvt_print_destroy

Free Print Record

Summary

```
void xvt_print_destroy(PRINT_RCD *precp)
```

`PRINT_RCD *precp`

Pointer to print record.

Description

This function frees the storage occupied by a `PRINT_RCD` allocated with a call to `xvt_print_create`.

Usually, each document that your application may print should be associated with a pointer to a `PRINT_RCD`. When the document is no longer needed (e.g., when its window is destroyed), you should free the `PRINT_RCD` along with the other memory associated with the document. Of course, the document itself, which could contain a copy of the `PRINT_RCD`, can continue to reside on disk.

Under no circumstances should you use a pointer to a `PRINT_RCD` as an argument to the standard function `free`. Also, you should *not* call `xvt_print_destroy` with a pointer that you got from `malloc`.

See Also

```
E_DESTROY
PRINT_RCD
XVT_CALLCONV*
xvt_print_create
```

Example

In this code fragment a document's `PRINT_RCD` is freed when an `E_DESTROY` event is received. Note that for safety's sake, the pointer is checked to ensure that it is non-`NULL` before it's freed. It is then set to `NULL` to avoid a dangling pointer.

```
long XVT_CALLCONV1 win_eh (WINDOW win, EVENT* ep)
{
static PRINT_RCD *print_rcd = NULL;
...
    switch (ep->type) {
    ...
    case E_DESTROY:
    if (print_rcdp != NULL) {
    xvt_print_destroy(print_rcdp);
    print_rcdp = NULL;
    }
    ...
    break;
    ...
    }
}
```

xvt_print_get_next_band

Get Coordinates of next Printing Band

Summary

```
RCT *xvt_print_get_next_band(void)
```

Description

This function is called repeatedly for a single print page, and it returns successive rectangular regions (or "bands") of the page to be drawn by your application. The native print driver then takes care of concatenating all of the bands into a page and sending them to the printer.

`xvt_print_get_next_band` returns `NULL` to indicate that all of the bands for a particular page have been processed. When drawing into the print band for a page, your application acts as if it were drawing the entire page. The window system takes care of clipping all drawing to the current band.

However, if your application wishes to optimize its drawing, it may examine the rectangle returned by `xvt_print_get_next_band`, or alternatively, it may call `xvt_dwin_is_update_needed` to determine which parts of the page need to be drawn for the current band. This is analogous to the optimization that can be performed while processing an `E_UPDATE` event.

You must call `xvt_print_open_page` before the first call to `xvt_print_get_next_band` for a page. You must call

`xvt_print_close_page` after `xvt_print_get_next_band` returns `NULL`.

Note: You should only call `xvt_print_get_next_band` within the print function whose address was passed to `xvt_print_start_thread`.

Return Value

Pointer to a `RCT` if the band is to be drawn; `NULL` if no bands remain.

Implementation Note

Some platform printing drivers do not implement banding. In these cases, `xvt_print_get_next_band` returns exactly one band, which represents the entire page. In any case, your application should be coded identically.

On XVT/Win32, banding printers can either print the page by returning several small bands, or return a single band representing the entire page in which text is to be drawn and multiple bands in which graphics are to be drawn. Your application need not be concerned about this except in that it must realize that all graphics appear to be drawn on top of the text. For this reason, your application should not attempt to clear the background of the print window because that would obscure all of the text.

See Also

```
E_UPDATE
RCT
xvt_dwin_is_update_needed
xvt_print_close_page
xvt_print_create_win
xvt_print_open_page
xvt_print_start_thread
```

The "Printing" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_print_start_thread`.

xvt_print_is_valid

Check Print Record

Summary

```
BOOLEAN xvt_print_is_valid(PRINT_RCD *precp)
```

`PRINT_RCD *precp`

Pointer to print record.

Description

This function checks if the `PRINT_RCD` pointed to by `precp` is valid for the current system configuration. You use this function after you have read a `PRINT_RCD` that was saved with a document file, since the current system configuration may not be the same as when the `PRINT_RCD` was saved. Calling `xvt_print_is_valid` does not alter the `PRINT_RCD` or transform an invalid print record to a valid one.

Note: You should call `xvt_print_is_valid` on a `PRINT_RCD` loaded from a file before using `PRINT_RCD` either for printing or for calling the function `xvt_dm_post_page_setup`.

Return Value

`TRUE` if the `PRINT_RCD` is valid; `FALSE` otherwise.

Implementation Note

On XVT/Mac, the current printer is set with the Chooser DA, and `xvt_print_is_valid` checks a `PRINT_RCD` against the current printer.

On XVT/Win32 the current printer is stored as part of the `PRINT_RCD`, and `xvt_print_is_valid` checks if the printer exists on the system, and if its settings are valid.

See Also

```
PRINT_RCD
xvt_dm_post_page_setup
xvt_print_create
xvt_print_destroy
```

The "Printing" chapter in the *XVT Portability Toolkit Guide*

Example

```
if (!xvt_print_is_valid(precp)){
    int size;
    xvt_print_destroy(precp);
    precp = xvt_print_create(&size);
}
```

xvt_print_open

Initialize Printing Manager

Summary

```
BOOLEAN xvt_print_open(void)
```

Description

This function opens the printing manager and returns `TRUE` if it is successfully opened. You call this function to determine if the print manager is accessible, and if there is an installed printer.

If successful, you should then close the print manager with a call to `xvt_print_close`. Do not keep the print manager open indefinitely, as doing so might prevent other applications from printing.

All of the XVT printing functions call `xvt_print_open` and `xvt_print_close` automatically, so normally, you don't have to call them yourself.

Return Value

`TRUE` if successful; `FALSE` on error (access denied).

Implementation Note

`xvt_print_open` is only implemented on certain platforms. On other platforms, it always returns `TRUE`.

On XVT/Mac, `xvt_print_open` prevents any other application from accessing the printer until `xvt_print_close` is called. This is necessary because of the way that the Mac Print Manager is structured.

See Also

`xvt_dm_post_error`
`xvt_print_close`

The "Printing" chapter in the *XVT Portability Toolkit Guide*

Example

```
BOOLEAN
is_printer_installed()
{
    if (!xvt_print_open()) {
        xvt_dm_post_error("No printer is installed.");
        return FALSE; }
    xvt_print_close();
    return TRUE;
}
```

xvt_print_open_page

Start New Page

Summary

```
BOOLEAN xvt_print_open_page(PRINT_RCD *precp)
```

`PRINT_RCD *precp`

Pointer to print record.

Description

Your application can call `xvt_print_open_page` only within a print loop to begin a new page.

Note: You should call `xvt_print_open_page` only within the print function whose address was passed to `xvt_print_start_thread`.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful (on error).

See Also

```
PRINT_RCD
xvt_print_start_thread
xvt_vobj_destroy
```

The "Printing" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_print_start_thread`.

xvt_print_set_page_orient

Set the Orientation of the Printed Page

Summary

```
BOOLEAN xvt_print_set_page_orient(PRINT_RCD* precp,
    XVT_PG_ORIENT pgorient)
```

`PRINT_RCD* precp`

Valid print record created with `xvt_print_create`.

`XVT_PG_ORIENT pgorient`

An enum describing the page orientation.

Description

This function is used to programmaticly set the page orientation of the printed page.

Return Value

`TRUE` if the function is succesful; `FALSE` otherwise.

Parameter Validity and Error Conditions

XVT issues an error if `precp` is `NULL`.

See Also

```
XVT_PG_ORIENT
xvt_print_create
xvt_print_set_page_size
```

xvt_print_set_page_size

Change the Printer Paper Size

Summary

```
BOOLEAN xvt_print_set_page_size(PRINT_RCD* precp,
     XVT_PG_SIZE pgsize)
```

`PRINT_RCD* precp`

Valid print record.

`XVT_PG_SIZE pgsize`

The XVT_PG_SIZE structure contains information about the size of the page, including the height, width, and units.

Description

This function allows you to programmatically change the printer paper size. The paper size is set using the XVT_PG_SIZE structure, which specifies the height, width, and units of the page. The function uses an algorithm that will select the closest supported printer paper size that matches the size sent in the pgsize argument. If no supported paper size is found, the smallest supported paper size that will contain the size passed. The width is the first criteria to determine the smallest supported size. If the size passed is larger than any supported size, then the largest supported size will be used.

Return Value

TRUE is returned if this function is succesful in setting a paper size; otherwise FALSE.

Parameter Validity and Error Conditions

XVT issues an error if any of the following are true:

- precp is NULL
- pgsize is NULL or negetive

See Also

XVT_PG_SIZE
xvt_print_set_page_orient

xvt_print_start_thread

Start Printing

Summary

```
BOOLEAN xvt_print_start_thread
    (BOOLEAN (* XVT_CALLCONV1 print_fcn)(void),
    long data)
```

`BOOLEAN (* XVT_CALLCONV1 print_fcn)(void)`

Print function.

`long data`

Data for print_fcn.

Description

Threads are not available on all platforms. XVT instead gives you the function `xvt_print_start_thread`, which creates a separate thread for printing if it is supported, but not on other platforms.

In XVT all printing is bounded by calls to `xvt_print_open` and `xvt_print_close`. In between these calls, you create a print window with `xvt_print_create_win` and draw into it with drawing functions. These drawing functions should be packaged into a single application function. Your printing function is given to XVT via the `print_fcn` argument to `xvt_print_start_thread`. XVT then calls your function to print. Do not call your print function directly; it should be called only from `xvt_print_start_thread`.

The `print_fcn` should use a normal sequence of printing commands to print the document.The `data` parameter can be used to pass information to `print_fcn`. Your printing function should return `TRUE` if successful and `FALSE` on error. This return value from the `print_fcn` is passed on to `xvt_print_start_thread` and returned so that your application can process errors.

Note: These the things that you can and cannot do within your print function:

- Print function cannot make any calls that would result in events being dispatched to your application (e.g., `xvt_*_create_*`, `xvt_dwin_update`, `xvt_vobj_destroy`, `xvt_vobj_move`, `xvt_vobj_set_enabled`, and `xvt_vobj_set_visible`).
- The print thread is restricted to operating only on the print window created from `xvt_print_create_win`.
- You cannot call any dialog including XVT standard dialogs (e.g., `xvt_dm_post_note` or `xvt_dm_post_error`) from within your print function; instead, your print function must return `FALSE` indicating an error.
- Because your application's event handlers can still receive `E_UPDATE` events, drawing functions must be re-entrant. They must be capable of printing your document and drawing it into a window simultaneously. This means that critical data necessary for drawing should be persistent.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful (on error).

Implementation Note

To insure portability across all platforms, you should include the macro `XVT_CALLCONV1` in the prototypes and headers of all callback functions used in XVT applications, including those of `print_fcns`. This macro defines the linkage conventions used in building XVT libraries.

See Also

`XVT_CALLCONV*`
`xvt_print_close`
`xvt_print_create`
`xvt_print_create_win`
`xvt_print_destroy`
`xvt_print_open`

The "Printing" chapter in the *XVT Portability Toolkit Guide*

Example

```
/* return TRUE for success */
BOOLEAN XVT_CALLCONV1 doc_print_fcn(long data)
{
    PRINT_RCD *print_rcd;
    WINDOW print_win;
    BOOLEAN print_ok;
    print_ok = FALSE;
    print_rcd = (PRINT_RCD *) data;
    if (print_rcd && xvt_print_is_valid(print_rcd))
    {
        print_win = xvt_print_create_win(print_rcd,
            "Print Job");
        if (print_win != NULL_WIN)
        {
            if (xvt_print_open_page(print_rcd))
            {
                RCT *rect;
                rect = xvt_print_get_next_band();
                while (rect != (RCT *) NULL)
                {
                    draw_page(win, rect);
                    rect = xvt_print_get_next_band();
                }
                if (xvt_print_close_page(print_rcd))
                    print_ok = TRUE;
            }
            xvt_vobj_destroy(print_win);
        }
    }
    return print_ok;
}
     ...
        /* processing routine for menus */
    case M_FILE_PRINT:
        if (xvt_print_start_thread(doc_print_fcn,
            (long) print_rcd) == FALSE)
        xvt_dm_post_error("Printing Failed");
    ...
```

xvt_rect_*

```
xvt_rect_get_height
xvt_rect_get_pos
xvt_rect_get_width
xvt_rect_has_point
xvt_rect_intersect
xvt_rect_is_empty
xvt_rect_offset
xvt_rect_set
xvt_rect_set_empty
xvt_rect_set_height
xvt_rect_set_pos
xvt_rect_set_width
```

xvt_rect_get_height

Get the Height of a Rectangle

Summary

```
short xvt_rect_get_height(RCT *rctp)

RCT *rctp
```

Pointer to the rectangle whose height is being inquired.

Description

This function returns the height (in pixels) of a rectangle, calculated as `rctp->bottom - rctp->top`.

Return Value

A `short` that is the height of the rectangle.

Parameter Validity and Error Conditions

XVT issues an error if the pointer to the rectangle is `NULL`.

See Also

```
RCT
xvt_rect_set_height
```

The "Points and Rectangles" section of the "Coordinate Systems" chapters in the *XVT Portability Toolkit Guide*

xvt_rect_get_pos

Get the Position of a Rectangle

Summary

```
PNT* xvt_rect_get_pos(RCT *rctp, PNT *pos)
```

`RCT *rctp`

Pointer to the rectangle whose position is being inquired.

`PNT *pos`

Pointer to the upper-left corner of rectangle.

Description

This function returns the upper-left corner of a rectangle, which is `rctp->top - rctp->left`, returned as a `PNT*`.

Return Value

A `PNT*` that is the position of the rectangle.

Parameter Validity and Error Conditions

XVT issues an error if either the pointer `rctp` or the pointer to `pos` is `NULL`.

See Also

`PNT`
`RCT`
`xvt_rect_set_pos`

The "Points and Rectangles" section of the "Coordinate Systems" chapter in the *XVT Portability Toolkit Guide*

xvt_rect_get_width

Get the Width of a Rectangle

Summary

```
short xvt_rect_get_width(RCT *rctp)
```

`RCT *rctp`

Pointer to the rectangle whose width is being inquired.

Description

This function returns the width (in pixels) of a rectangle, calculated as `rctp->right - rctp->left`.

Return Value

A `short` that is the width of the rectangle.

Parameter Validity and Error Conditions

XVT issues an error if the pointer to the rectangle is `NULL`.

See Also

`RCT`
`xvt_rect_set_width`

The "Points and Rectangles" section of the "Coordinate Systems" chapter in the *XVT Portability Toolkit Guide*

---

xvt_rect_has_point

Test Whether a Point is Inside a Rectangle

Summary

```
BOOLEAN xvt_rect_has_point(RCT *rctp, PNT pnt)
```

`RCT *rctp`

Pointer to the rectangle.

`PNT pnt`

Point whose position is being determined.

Description

This function determines whether the point `pnt` is inside the rectangle pointed to by `rctp`. "Inside" is defined as follows:

```
(pnt.h < rctp->left) &&
(pnt.h < rctp->right) &&
(pnt.v >= rctp->top) &&
(pnt.v < rctp->bottom)
```

This function works with any coordinate system, provided the point and the rectangle are in the same system.

Return Value

`TRUE` if the point specified by `pnt` is either on the border or inside the rectangle; `FALSE` otherwise.

Parameter Validity and Error Conditions

XVT issues an error if the pointer to the rectangle is `NULL`.

Implementation Note

XVT assumes that mathematical coordinates lie *between* the pixels, not *on* them.

See Also

`PNT`
`RCT`
`xvt_rect_intersect`

The "Points and Rectangles" section of the "Coordinate Systems" chapter in the *XVT Portability Toolkit Guide*

xvt_rect_intersect

Check if Rectangles Intersect

Summary

```
BOOLEAN xvt_rect_intersect(RCT *drctp, RCT *rctp1,
     RCT *rctp2)
```

`RCT *drctp`

Pointer to the destination rectangle, or `NULL`.

`RCT *rctp1`

First source rectangle.

`RCT *rctp2`

Second source rectangle.

Description

This function calculates the intersection of `rctp1` and `rctp2`. If the destination rectangle pointed to by `drctp` is not `NULL`, the intersection rectangle is stored there. The `rctp1` and `rctp2` arguments must be pointers to `RCT` structures. The `drcpt` argument can either be `NULL` or a pointer to an `RCT` structure.

Return Value

`TRUE` if the intersection of the two source rectangles is non-empty; `FALSE` otherwise.

Parameter Validity and Error Conditions

XVT issues an error if either the pointer to `rctp1` or the pointer to `rctp2` is `NULL`.

See Also

`xvt_rect_has_point`
`xvt_rect_is_empty`

The "Points and Rectangles" section of the "Coordinate Systems" chapter in the *XVT Portability Toolkit Guide*

---

xvt_rect_is_empty

Check for Empty Rectangle

Summary

```
BOOLEAN xvt_rect_is_empty(RCT *rctp)

RCT *rctp
```

Pointer to the rectangle whose boundaries are being checked.

Description

This function checks the boundaries of the given rectangle. An empty rectangle does not necessarily mean that all four coordinates are zero. An empty rectangle is one with a width and height of zero.

Return Value

`TRUE` if the area of the rectangle is empty; `FALSE` otherwise.

Parameter Validity and Error Conditions

XVT issues an error if the pointer to the rectangle is `NULL`.

See Also

`RCT`
`xvt_rect_intersect`
`xvt_rect_set`
`xvt_rect_set_empty`

The "Points and Rectangles" section of the "Coordinate Systems" chapter in the *XVT Portability Toolkit Guide*

xvt_rect_offset

Offset Rectangle's Coordinates

Summary

```
void xvt_rect_offset(RCT *rctp, short dh, short dv)
```

`RCT *rctp`

Pointer to the rectangle.

`short dh`

Horizontal value to be added to both the `left` and `right` fields.

`short dv`

Vertical value to be added to both the `top` and `bottom` fields.

Description

This function offsets the `RCT` structure pointed to by `rctp` by adding the value of the `dh` argument to both the `left` and `right` fields, and by adding the value of the `dv` argument to both the `top` and `bottom` fields. Thus, `dh` is the horizontal offset, and `dv` is the vertical offset. The size of the rectangle doesn't change.

The `dh` and `dv` arguments can be positive, negative, or zero.

Parameter Validity and Error Conditions

XVT issues an error if the pointer to the rectangle is `NULL`.

See Also

`RCT`
`xvt_rect_set`

The "Points and Rectangles" section of the "Coordinate Systems" chapter in the *XVT Portability Toolkit Guide*

xvt_rect_set

Set a Rectangle's Coordinates

Summary

```
void xvt_rect_set(RCT *rctp, short left, short top,
     short right, short bottom)
```

`RCT *rctp`

Pointer to the rectangle whose coordinates are being set.

`short left`

Left coordinate.

`short top`

Top coordinate.

`short right`

Right coordinate.

`short bottom`

Bottom coordinate.

Description

This function sets the members of the `RCT` pointed to by `rctp` to the values of the corresponding arguments. Any coordinate system can be used.

Parameter Validity and Error Conditions

XVT issues an error if the pointer to the rectangle is `NULL`.

See Also

```
RCT
xvt_rect_is_empty
xvt_rect_set_empty
```

Example

```
RCT r;xvt_rect_set(&r, 100, 75, 300, 200);
xvt_vobj_move(win, &r);
```

xvt_rect_set_empty

Set Rectangle to Empty

Summary

```
void xvt_rect_set_empty(RCT *rctp)
```

`RCT *rctp`

Pointer to the rectangle which is being "emptied."

Description

This function sets the rectangle's height and width to zero, but it does not change the rectangle's position (i.e., its left and top coordinates remain unchanged).

Parameter Validity and Error Conditions

XVT issues an error if the pointer to the rectangle is `NULL`.

See Also

`RCT`
`xvt_rect_is_empty`

The "Points and Rectangles" section of the "Coordinate Systems" chapter in the *XVT Portability Toolkit Guide*

xvt_rect_set_height

Set the Height of a Rectangle

Summary

```
BOOLEAN xvt_rect_set_height(RCT *rctp, short height)
```

`RCT *rctp`

Pointer to the rectangle whose height is being set.

`short height`

New height.

Description

This function sets the height (in pixels) of a rectangle by adjusting the bottom coordinate.

Return Value

`TRUE` if succcessful; `FALSE` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if the pointer to the rectangle is `NULL`.

See Also

`RCT`
`xvt_rect_get_height`
`xvt_rect_set_width`
`xvt_vobj_move`

The "Points and Rectangles" section of the "Coordinate Systems" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_vobj_move`.

---

xvt_rect_set_pos

Set the Position of a Rectangle

Summary

```
BOOLEAN xvt_rect_set_pos(RCT *rctp, PNT pos)
```

`RCT *rctp`

Pointer to the rectangle whose position is being set.

`PNT pos`

New upper-left position.

Description

This function sets the position of a rectangle by adjusting all four coordinates. The width and height of the rectangle are unchanged.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if the pointer to the rectangle is `NULL`.

See Also

`PNT`
`RCT`
`xvt_vobj_move`

The "Points and Rectangles" section of the "Coordinate Systems" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_vobj_move`.

---

xvt_rect_set_width

Set the Width of a Rectangle

Summary

```
BOOLEAN xvt_rect_set_width(RCT *rctp, short width)
```

`RCT *rctp`

Pointer to the rectangle whose width is being set.

`short width`

New width.

Description

This function sets the width (in pixels) of a rectangle by adjusting the right coordinate.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if the pointer to the rectangle is `NULL`.

See Also

`RCT`
`xvt_rect_get_width`
`xvt_rect_set_height`

The "Points and Rectangles" section of the "Coordinate Systems" chapter in the *XVT Portability Toolkit Guide*

xvt_res_*

Resource Management Functions

```
xvt_res_add_res
xvt_res_free_menu_tree
xvt_res_free_win_def
xvt_res_get_dlg_data
xvt_res_get_dlg_def
xvt_res_get_font
xvt_res_get_image
xvt_res_get_image_data
xvt_res_get_menu
xvt_res_get_menu_data
xvt_res_get_str
xvt_res_get_str_list
xvt_res_get_win_data
xvt_res_get_win_def
xvt_res_remove_res
xvt_res_use_res
```

xvt_res_add_res

Add a resource file to the application

Summary

```
XVT_RES xvt_res_add_res(char *res_name)

char *res_name
```

Resource's filename.

Description

This function opens the file specified by `res_name` and checks that it is a valid native resource file. If valid, it opens the specified resource file, makes it the current application resource file, and returns an `XVT_RES` object.

The number of resource files that can be open simultaneously is limited by native constraints. To use an already opened resource, call the function `xvt_res_use_res`.

Return Value

A valid `XVT_RES`; `NULL` if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- The specified resource file cannot be opened
- `res_name` is `NULL`
- Platform specific resource problems occur

Implementation Note

On XVT/Win32, a resource is a valid DLL with bound resources.

On XVT/XM a resource is a valid **UID** file.

On XVT/Mac a resource is any file with a valid resource fork.

See Also

```
XVT_RES
xvt_res_use_res
xvt_res_remove_res
```

Example

```
XVT_RES newResource = (XVT_RES)NULL;

if (NULL == (newResource =
    xvt_res_add_res("newres.dll")))
{
    xvt_dm_post_note("Can't load newres.dll");
}
```

xvt_res_free_menu_tree

Free MENU_ITEM Tree

Summary

```
void xvt_res_free_menu_tree(MENU_ITEM *mip)

MENU_ITEM *mip
```

`MENU_ITEM` tree to be freed.

Description

This function frees `MENU_ITEM` trees. When a pointer returned by `xvt_menu_get_tree` or `xvt_res_get_menu` is no longer needed, it

must be freed with a call to xvt_res_free_menu_tree. In addition, you can use this function to free MENU_ITEM trees that you have allocated yourself, provided that you have followed the conventions of xvt_menu_get_tree.

This function assumes that all memory (for MENU_ITEM arrays and the text members) was allocated with xvt_mem_*alloc. You can change the tree returned by xvt_menu_get_tree or xvt_res_get_menu yourself and still legally call xvt_res_free_menu_tree, provided that you allocate all new memory with xvt_mem_*alloc.

Parameter Validity and Error Conditions

If mip is NULL, the severity error "Argument NULL" is issued.

See Also

MENU_ITEM
xvt_menu_get_tree
xvt_res_get_menu
menu and menubar URL Statement

The "Menus" and the "Resources and URL" chapters in the *XVT Portability Toolkit Guide*

Example

See the example for xvt_menu_set_tree.

xvt_res_free_win_def

Free WIN_DEF Array

Summary

```
void xvt_res_free_win_def(WIN_DEF *win_def_p)
```

WIN_DEF *win_def_p

Pointer to WIN_DEF structures to be freed.

Description

This function frees an array of WIN_DEF structures pointed to by the win_def_p pointer. The array of WIN_DEF structures must adhere to the format returned by xvt_res_get_win_def or xvt_res_get_dlg_def. Typically, you call this function when you are done using a WIN_DEF array that was allocated with either

xvt_res_get_win_def or xvt_res_get_dlg_def. In particular, this function performs the following actions:

- It determines the number of elements in win_def_p, by scanning for the first element whose wtype field is set to W_NONE. This element is assumed to be the last element in the array.
- It frees the text for each win_def_p[i].text field that is not NULL.
- If win_def_p[0] describes a window, xvt_res_free_menu_tree will be called with win_def_p[0].v.win.menup if it is not NULL.
- It destroys the logical fonts embedded in the WIN_DEF array.
- It frees arrays of XVT_COLOR_COMPONENT.
- The entire array is freed; this function assumes that all memory has been allocated with xvt_mem_*alloc functions

Parameter Validity and Error Conditions

No error checking is performed. However, the program might crash if the pointers are invalid.

Caution: If this function encounters the same logical font ID, text string pointer, or array of XVT_COLOR_COMPONENT more than one, it will attempt to destroy (font IDs) or free (text and color components) for each instance. This could result in an error. The application should be careful to not call this function with a WIN_DEF array that is in this state.

See Also

```
xvt_font_destroy
xvt_mem_alloc
xvt_res_get_dlg_def
xvt_res_get_win_def
```

xvt_res_get_*

Get Resource File Functions

```
xvt_res_get_dlg_data
xvt_res_get_dlg_def
xvt_res_get_font
xvt_res_get_image
xvt_res_get_image_data
xvt_res_get_menu
xvt_res_get_menu_data
xvt_res_get_str
xvt_res_get_str_list
xvt_res_get_win_data
xvt_res_get_win_def
```

xvt_res_get_dlg_data

Get User Data String for Dialog Control

Summary

```
char *xvt_res_get_dlg_data(int rid, int cid,
     int data_tag)
```

`int rid`

Specified dialog.

`int cid`

Control whose user data string is being retrieved.

`int data_tag`

User data strings to be retrieved.

Description

This function retrieves user data for the control whose control ID is `cid` in the dialog specified by `rid`. User data is static text information that is specified in an URL resource script so that your application can retrieve the information later.

If `cid` is zero, the user data for the dialog itself is returned instead of the user data for a control. The user data must have been previously specified in an URL script for the dialog.

`data_tag` identifies which of the user data strings associated with the specified item is to be retrieved. The URL syntax allows the specification of an arbitrary number of user data strings, and **curl** associates a sequentially numbered `data_tag` with each user data string for a particular item. An application should use a `data_tag` of zero to retrieve the first user data string for an item, one for the second, etc.

URL allows different types of objects to share `rid` values within the same URL file. Thus, it is legal to have a menu, dialog, and window all possessing the `rid` 1000. In that case, `xvt_res_get_dlg_data(1000, 1, 0)` returns a different user data string than `xvt_res_get_win_data(1000, 1, 0)`.

Note: In addition to user data, XVT gives you another means of associating data with a dialog. Application data is information attached dynamically via `xvt_vobj_set_data` to an object that is instantiated at runtime. Application data generally holds state information that is kept at runtime, while user data generally holds additional static attributes for the dialog.

Return Value

A pointer to the user data string for the specified object if successful; `NULL` if unsuccessful.

Since the user data string is allocated via `xvt_mem_alloc`, you must free it using `xvt_mem_free`.

See Also

```
xvt_mem_alloc
xvt_mem_free
xvt_res_get_menu_data
xvt_res_get_win_data
xvt_vobj_get_data
xvt_vobj_set_data
dialog URL Statement
```

The "Dialogs" and the "Resources and URL" chapters in the *XVT Portability Toolkit Guide*

xvt_res_get_dlg_def

Load Dialog Definition from a Resource File

Summary

```
WIN_DEF *xvt_res_get_dlg_def(int rid)

int rid
```

Resource ID of the dialog definition.

Description

This function reads a dialog definition from a resource file, and loads it into an array of `WIN_DEF` structures, which it allocates from the heap. The returned array of `WIN_DEF` structures are appropriate as an argument to `xvt_dlg_create_def`.

If all you want to do is load a dialog resource and instantiate it in one step, then you should call `xvt_dlg_create_res`. Use `xvt_res_get_dlg_def` when you need to modify the dialog specification before instantiating it.

Return Value

A pointer to an array of `WIN_DEF` objects if successful; `NULL` if unsuccessful (on error).

You must free the array of `WIN_DEF` structures, when it is no longer needed, using `xvt_res_free_win_def`.

See Also

```
xvt_dlg_create_def
xvt_dlg_create_res
xvt_res_free_win_def
xvt_res_get_win_def
dialog URL Statement
```

xvt_res_get_font

Get an XVT_FNTID from a Resource File

Summary

```
XVT_FNTID xvt_res_get_font(int rid)
```

`int rid`

Resource ID of the `font` resource in the URL file.

Description

This function retrieves the logical font whose resource ID is `rid`. The returned `XVT_FNTID` is initialized to the values given in the corresponding URL `font` statement. You must call `xvt_font_destroy` to remove this font from the system.

Return Value

`XVT_FNTID` if successful; `NULL_FNTID` if unsuccessful.

Parameter Validity and Error Conditions

If `rid` is not the resource ID of a valid logical font, XVT issues an error.

See Also

`xvt_dwin_set_font_*`
`xvt_font_create`
`xvt_font_destroy`
`font` URL Statement
`font_map` URL statement

xvt_res_get_image

Get an Image from a Resource File

Summary

```
XVT_IMAGE xvt_res_get_image(int rid)
```

`int rid`

Resource ID of the image.

Description

This function retrieves an image whose resource ID is `rid`, and then returns it to the caller as an `XVT_IMAGE` object.

To destroy the object when you no longer need it, call `xvt_image_destroy`.

Return Value

A valid `XVT_IMAGE` object if successful; `NULL_IMAGE` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `rid` does not refer to an image resource in the resource file
- Sufficient memory cannot be allocated for the image object
- The resource image refers to another file, and that file does not exist, could not be read, or is not a valid image file

See Also

```
XVT_IMAGE
xvt_dwin_draw_image
xvt_image_destroy
image URL Statement
```

The "Portable Images" and the "Resources and URL" chapters in the *XVT Portability Toolkit Guide*

xvt_res_get_image_data

Get User Data String for an Image

Summary

```
char *xvt_res_get_image_data(int rid, int data_tag)
```

`int rid`

Resource ID of the image.

`int data_tag`

Identifies which of the user data strings associated with the specified item are to be retrieved.

Description

This function retrieves user data for an image. User data is static information that is stored in a resource file so that your application can retrieve the information later. The user data for the resource must have been previously specified in an URL script for the image.

The URL syntax allows the specification of an arbitrary number of user data strings, and **curl** associates a sequentially numbered `data_tag` with each user data string for a particular item. An application should use a `data_tag` of zero to retrieve the first user data string for an item, one for the second, etc.

Return Value

A pointer to the user data string for the specified object if successful;`NULL` if unsuccessful. Since the user data string is allocated with `xvt_mem_alloc`, you must free it using `xvt_mem_free`.

See Also

```
xvt_mem_alloc
xvt_mem_free
xvt_res_get_dlg_data
xvt_res_get_win_data
image URL Statement
```

The "Portable Images" and the "Resources and URL" chapters in the *XVT Portability Toolkit Guide*

xvt_res_get_menu

Load Menu Definition from a Resource File

Summary

```
MENU_ITEM *xvt_res_get_menu(int rid)

int rid
```

Resource ID of the menu.

Description

This function reads the menu description specified by `rid`, which must be a menubar resource, from an application's resources and loads it into a hierarchy of `MENU_ITEM` arrays. A `MENU_ITEM` array is allocated (with `xvt_mem_alloc`) for the menubar and each pull-

down menu and each submenu. The child pointers of these arrays are linked to reflect the menu hierarchy. Each `MENU_ITEM` array has an extra item at the end with the tag field set to zero.

You can modify this data structure if you want to change the menus, but you must follow the rules documented in `MENU_ITEM` and in "Menus" chapter in the *XVT Portability Toolkit Guide*. To have the menu hierarchy attached to a window and displayed as that window's menu, you can call `xvt_menu_set_tree`.

Return Value

A pointer to a `MENU_ITEM` hierarchy if successful; `NULL` if the `rid` cannot be found or storage could not be allocated for any of the `MENU_ITEM` arrays (in this case, any menus that were allocated have been freed already). If successful, you must free the menu hierarchy, when it is no longer needed, using `xvt_res_free_menu_tree`.

See Also

```
MENU_ITEM
xvt_menu_get_tree
xvt_menu_set_tree
xvt_res_free_menu_tree
menu and menubar URL Statement
```

The "Menus" and the "Resources and URL" chapters in the *XVT Portability Toolkit Guide*

Example

```
MENU_ITEM *menup;
menup = xvt_res_get_menu(MY_MENU_RID);
if(menup != NULL)
    xvt_menu_set_tree(win, menup);
```

xvt_res_get_menu_data

Get User Data String for Menu

Summary

```
char *xvt_res_get_menu_data(int rid,
     MENU_TAG menu_tag, int data_tag)

int rid
```

Resource ID of the menu.

```
MENU_TAG menu_tag
```

Identifies which of the menus or menu items associated with `rid` are to be retrieved.

`int data_tag`

Identifies which of the user data strings associated with the `menu_tag` item are to be retrieved.

Description

This function retrieves user data for the menu specified by `menu_tag`, or a menu item, located in the `rid` menu resource.

The `data_tag` parameter identifies which of the user data strings associated with the `menu_tag` item is to be retrieved. The URL syntax allows the specification of an arbitrary number of user data strings, and **curl** will associate a sequentially-numbered `data_tag` with each user data string for a particular item. An application should use a `data_tag` of zero to retrieve the first user data string for an item, one for the second, etc.

Note: URL allows different types of objects to share resource ID values within the same URL file. Thus, it is legal to have a menu, dialog, and window all possessing the resource ID 1000. In that case, `xvt_res_get_menu_data(1000, 1, 0)` returns a different user data string than `xvt_res_get_dlg_data(1000, 1, 0)`.

Return Value

A pointer to the user data string for the specified object if successful; `NULL` if unsuccessful. The user data string is allocated via `xvt_mem_alloc`, so you must free it using `xvt_mem_free`.

See Also

`xvt_mem_free`
`xvt_res_get_dlg_data`
`xvt_res_get_win_data`
`menu and menubar URL Statement`

The "Resources and URL" chapter in the *XVT Portability Toolkit Guide*

xvt_res_get_str

Get String Resource

Summary

```
char *xvt_res_get_str(int rid, char *s, int sz_s)
```

`int rid`

Resource ID of the string.

`char *s`

Buffer into which the string is to be stored.

`int sz_s`

Maximum capacity of the buffer.

Description

This function gets the string resource specified by resource ID `rid` and stores it into the string specified by `s`. The maximum capacity of `s` including the terminating `NULL` is `sz_s`. The loaded string is truncated and `NULL`-terminated as necessary to fit into `s`.

If you have a group of string resources that are stored together in your resource file, you might want to use `xvt_res_get_str_list` instead of `xvt_res_get_str`.

Return Value

Value of `s` argument if successful; `NULL` if unsuccessful (on error).

Implementation Note

On XVT/Mac, sequentially numbered string resources are not accessible by `xvt_res_get_str`. Instead, use `xvt_res_get_str_list`.

See Also

`xvt_res_get_str_list`
`string URL Statement`

The "Multibyte Character Sets and Localization" and the "Resources and URL" chapters in the *XVT Portability Toolkit Guide*

Example

```
char buf[100];if (xvt_res_get_str(STR_TEST, buf,
    sizeof(buf)) != NULL)
    xvt_dm_post_note("%s", buf);
else
    xvt_dm_post_error("Can't read string resource %d.",
        STR_TEST);
```

xvt_res_get_str_list

Get List String Resources

Summary

```
SLIST xvt_res_get_str_list(int rid_first, int rid_last)
```

`int rid_first`

Resource ID of the first string.

`int rid_last`

Resource ID of the last string.

Description

This function gets resource strings with resource IDs from `rid_first` through `rid_last` and returns them in the form of an `SLIST`. The data word associated with each string is zero.

There must be exactly `rid_last - rid_first + 1` string resources, and their resource IDs must be consecutive.

Note: If you need only a single string, or if you need several strings with non-consecutive IDs, you can call `xvt_res_get_str` instead.

Return Value

`SLIST` containing strings if successful; `NULL` if unsuccessful (on error). When the returned `SLIST` is no longer needed, use `xvt_slist_destroy` to free it.

Implementation Note

On XVT/Mac, the `rid_first` argument must be the first resource ID of a given `STR#` resource group. The value of `rid_last` is ignored; the resource must be of type `STR#`, which indicates a sequence of string resources associated with a single resource ID that is specified by the `rid_first` argument.

Also on XVT/Mac, sequential string resources specified in URL are only accessible through `xvt_res_get_str_list`, and not through `xvt_res_get_str`.

See Also

`xvt_res_get_str`
`xvt_slist_*`

The "Multibyte Character Sets and Localization" and the "Resources and URL" chapters in the *XVT Portability Toolkit Guide*

Example

This code adds the 50 state names (located in URL resources) to a list box:

```
#define STR_STATES1 260
#define STR_STATES2 309
...
SLIST slist;
WINDOW lbox;
...
slist = xvt_res_get_str_list(STR_STATES1,
    STR_STATES2);
xvt_list_clear(lbox);
if (!slist || !xvt_list_add(lbox, -1, slist))
    xvt_dm_post_error("Can not list states.");
if (slist)
    xvt_slist_destroy(slist);
```

xvt_res_get_win_data

Get User Data String for Window Control

Summary

```
char *xvt_res_get_win_data(int rid, int cid,
    int data_tag)
```

`int rid`

Resource ID of the window.

`int cid`

Control ID whose user data is being retrieved.

`int data_tag`

Identifies which of the user data strings associated with the specified item is to be retrieved.

Description

This function retrieves user data for the control whose control ID is `cid` in the window specified by `rid`. User data is static information that is stored in a resource file so that your application can retrieve the information later.

If `cid` is zero, then the user data for the window itself is returned instead of the user data for the control. The user data for the control must have been previously specified in an URL script for the window.

`data_tag` identifies which of the user data strings associated with the specified item is to be retrieved. The URL syntax allows the specification of an arbitrary number of user data strings, and **curl** will associate a sequentially-numbered `data_tag` with each user data string for a particular item. An application should use a `data_tag` of zero to retrieve the first user data string for an item, one for the second, etc.

URL allows different types of objects to share `rid` values within the same URL file. Thus, it is legal to have a menu, dialog, and window all possessing the `rid` 1000. In that case, `xvt_res_get_dlg_data(1000, 1, 0)` returns a different user data string than `xvt_res_get_win_data(1000, 1, 0)`.

In addition to user data, XVT gives you another means of associating data with a window. Application data is information that is attached dynamically via `xvt_vobj_set_data` to an object that is instantiated at runtime. Application data generally holds state information that is kept at runtime, while user data generally holds additional static attributes for the window.

Return Value

A pointer to the user data string for the specified object if successful; `NULL` if unsuccessful. The user data string is allocated via `xvt_mem_alloc`, so you must free it using `xvt_mem_free`.

See Also

```
xvt_mem_alloc
xvt_mem_free
xvt_res_get_dlg_def
xvt_res_get_menu_data
window URL Statement
```

Th "Windows" and the "Resources and URL" chapters in the *XVT Portability Toolkit Guide*

xvt_res_get_win_def

Load Window Definition from a Resource File

Summary

```
WIN_DEF *xvt_res_get_win_def(int rid)

int rid
```

Resource ID of the window definition.

Description

This function reads a window description from the resource file and loads it into an array of `WIN_DEF` structures, which it allocates from the heap. The returned array of `WIN_DEF` structures is appropriate as an argument to `xvt_win_create_def`.

If all you want to do is load a window resource and instantiate it in one step, then you should call `xvt_win_create_res`. Use `xvt_res_get_win_def` when you need to modify the window specification before instantiating it.

Return Value

A pointer to an array of `WIN_DEF` objects if successful; `NULL` if unsuccessful. When it is no longer needed, the array of `WIN_DEF` structures must be freed using `xvt_res_free_win_def`.

See Also

```
xvt_res_free_win_def
xvt_res_get_dlg_def
xvt_win_create_def
xvt_win_create_res
window URL Statement
```

The "Windows" and the "Resources and URL" chapters in the *XVT Portability Toolkit Guide*

xvt_res_remove_res

Remove Resource from Use

Summary

```
BOOLEAN xvt_res_remove_res(XVT_RES res)
```

`XVT_RES res`

A valid `XVT_RES`. (Resource must have been previously added with `xvt_res_add_res`.)

Description

This function removes a resource from the application. The resource is closed, and all related memory is freed.

Return Value

`TRUE` is returned on success; `FALSE` on failure.

Parameter Validity and Error Conditions

A possible cause of a failure is that you are trying to delete the only resource. There must always be at least one resource available to the application.

See Also

```
XVT_RES
xvt_res_add_res
xvt_res_use_res
```

Example

```
/* Make sure resource is valid */
if (newResource) {
    if (!xvt_res_remove_res(newResource))
        xvt_dm_post_note("Failed to remove resource!");
}
```

xvt_res_use_res

Set Current Resource

Summary

```
XVT_RES xvt_res_use_res(XVT_RES res)
```

`XVT_RES res`

A valid `XVT_Res`.

Description

This function sets the current resource to be used. All resources will be retrieved from this resource. The return value is the resource that was current before the call to this function. This is convenient in the sense that the first call to this function will return the startup resource, which is usually the resource bound to the executable.

Return Value

The `XVT_RES` for the resource that was current before the call; `NULL` if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `res` is `NULL`
- Platform specific resource problems occur

See Also

`XVT_RES`
`xvt_res_add_res`
`xvt_res_remove_res`

Example

```
extern XVT_RES appResource;
XVT_RES prevResource = (XVT_RES)NULL;

/* Make sure resource is valid */
if (newResource) {
    if ((prevResource = xvt_res_use_res(newResource)) ==
            NULL) {
        xvt_dm_post_note("Error setting new resource!");
    }
    /* If first time resource is changed from default */
    elseif (appResource == NULL) {
        /* Save default (startup) resources */
        appResource = prevResource;
    }
}
```

xvt_sbar_*

Scrollbar Functions

```
xvt_sbar_get_pos
xvt_sbar_get_proportion
xvt_sbar_get_range
xvt_sbar_set_pos
xvt_sbar_set_proportion
xvt_sbar_set_range
```

xvt_sbar_get_pos

Get Scrollbar's Thumb Position

Summary

```
int xvt_sbar_get_pos(WINDOW win, SCROLL_TYPE t)
```

`WINDOW win`

Window whose scrollbar's thumb position is to be retreived.

`SCROLL_TYPE t`

Scrollbar type, horizontal or vertical.

Description

This function returns the current position of the scrollbar's thumb. The value returned will fall within the range set by `xvt_sbar_set_range` minus the amount set by `xvt_sbar_set_proportion`.

If `t` is `HSCROLL` or `VSCROLL`, this function gets the thumb position of the horizontal or vertical scrollbar on the frame of win.

If `t` is `HVSCROLL`, then `win` must be the `WINDOW` of a scrollbar control in the client area of a window or dialog.

Return Value

The current position of a scrollbar's thumb.

Parameter Validity and Error Conditions

XVT issues an error if:

- `win` is not a valid `WINDOW`
- `win` is a control of type `WC_HSCROLL` or `WC_VSCROLL`; *and* `t` is not equal to `HVSCROLL`
- `win` is a `WINDOW` of type `W_DOC` or `W_PLAIN` (child window), *and* has been created with the appropriate scrollbar control as set by `WSF_HSCROLL` or `WSF_VSCROLL`; *and* `t` is not equal to `HSCROLL` or `VSCROLL`

See Also

```
SCROLL_TYPE
xvt_sbar_get_proportion
xvt_sbar_set_pos
xvt_sbar_set_range
```

xvt_sbar_get_proportion

Get Scrollbar's Thumb Proportion

Summary

```
int xvt_sbar_get_proportion(WINDOW win, SCROLL_TYPE t)
```

`WINDOW win`

Window whose scrollbar's thumb proportion to be retreived.

`SCROLL_TYPE t`

Scrollbar type, horizontal or vertical.

Description

This function returns the value set by `xvt_sbar_set_proportion`. If no call to `xvt_sbar_set_proportion` has yet been made for that scrollbar, this function returns zero.

If `t` is `HSCROLL` or `VSCROLL`, this function gets the proportion of the horizontal or vertical scrollbar on the frame of `win`.

If `t` is `HVSCROLL`, then `win` is the `WINDOW` of a scrollbar control in the client area of a window or dialog.

Return Value

The current scrollbar's thumb proportion if successful.

Parameter Validity and Error Conditions

XVT issues an error if:

- `win` is not a valid `WINDOW`
- `win` is a control of type `WC_HSCROLL` or `WC_VSCROLL`; *and* `t` is not equal to `HVSCROLL`
- `win` is a `WINDOW` of type `W_DOC` or `W_PLAIN` (child window), *and* has been created with the appropriate scrollbar control as set by `WSF_HSCROLL` or `WSF_VSCROLL`; *and* `t` is not equal to `HSCROLL` or `VSCROLL`

See Also

```
SCROLL_TYPE
xvt_sbar_set_pos
xvt_sbar_set_proportion
```

xvt_sbar_get_range

Get Scrollbar's Range Values

Summary

```
void xvt_sbar_get_range(WINDOW win, SCROLL_TYPE t,
     int *minp, int *maxp)
```

```
WINDOW win
```

Window whose scrollbar's range values are being retrieved.

```
SCROLL_TYPE t
```

Scrollbar type, horizontal or vertical.

`int *minp`

Minimum range value.

`int *maxp`

Maximum range value.

Description

This function gets the current minimum and maximum range values for a scrollbar, as set by a preceding call to `xvt_sbar_set_range`, and stores them into the integers pointed to by `minp` and `maxp`, respectively.

If `t` is `HSCROLL` or `VSCROLL`, this function gets the minimum and maximum of the range of the horizontal or vertical scrollbar on the frame of `win`.

If `t` is `HVSCROLL`, then `win` is the `WINDOW` of a scrollbar control in the client area of a window or dialog.

If no previous call to `xvt_sbar_set_range` has been made for this scrollbar, then the minimum and maximum values are undefined.

Parameter Validity and Error Conditions

XVT issues an error if:

- `win` is not a valid `WINDOW`
- `win` is a control of type `WC_HSCROLL` or `WC_VSCROLL`; *and* `t` is not equal to `HVSCROLL`
- `win` is a `WINDOW` of type `W_DOC` or `W_PLAIN` (child window), *and* has been created with the appropriate scrollbar control as set by `WSF_HSCROLL` or `WSF_VSCROLL`; *and* `t` is not equal to `HSCROLL` or `VSCROLL`
- `minp` or `maxp` are `NULL`.

See Also

`SCROLL_TYPE`
`xvt_sbar_set_range`

xvt_sbar_set_pos

Set Position of a Scrollbar's Thumb

Summary

```
void xvt_sbar_set_pos(WINDOW win, SCROLL_TYPE t,
     int pos)
```

`WINDOW win`

Window whose scrollbar's thumb position is being set.

`SCROLL_TYPE t`

Scrollbar type, horizontal or vertical.

`int pos`

Position being set for the scrollbar's thumb. The position has to fall within the previously set range.

Description

This function sets the position of the scrollbar's thumb. The value of `pos` must fall within the range set by a previous call to `xvt_sbar_set_range` minus the amount set by `xvt_sbar_set_proportion`.

If `t` is `HSCROLL` or `VSCROLL`, this function sets the thumb position of the horizontal or vertical scrollbar on a regular or child window.

If t is `HVSCROLL`, then `win` is the `WINDOW` of a scrollbar control in the client area of a window or dialog.

Parameter Validity and Error Conditions

XVT issues an error if:

- `win` is not a valid `WINDOW`
- `win` is a control of type `WC_HSCROLL` or `WC_VSCROLL`; *and* `t` is not equal to `HVSCROLL`
- `win` is a `WINDOW` of type `W_DOC` or `W_PLAIN` (child window), *and* has been created with the appropriate scrollbar control as set by `WSF_HSCROLL` or `WSF_VSCROLL`; *and* `t` is not equal to `HSCROLL` or `VSCROLL`.
- `pos` is outside the previously set range and proportion

See Also

```
*SCROLL Values for SCROLL_TYPE
xvt_sbar_get_pos
xvt_sbar_set_proportion
xvt_sbar_set_range
```

xvt_sbar_set_proportion

Set Scrollbar's Thumb Proportion

Summary

```
void xvt_sbar_set_proportion(WINDOW win,
     SCROLL_TYPE t, int proportion)
```

`WINDOW win`

Window whose scrollbar's thumb proportion is being set.

`SCROLL_TYPE t`

Scrollbar type, horizontal or vertical.

`int proportion`

Thumb proportion; it must be >= 0.

Description

This function sets the scrollbar's thumb proportion. Conceptually, the thumb proportion is the part of a document or drawing that is visible in the viewable area, compared to the total size of the document. The word "proportion" might be misleading--it should be thought of as a sub-range.

If `t` is `HSCROLL` or `VSCROLL`, this function sets the thumb proportion of the horizontal or vertical scrollbar on a regular or child window.

If `t` is `HVSCROLL`, then `win` is the `WINDOW` of a scrollbar control in the client area of a window or dialog.

`proportion` is a value between zero and the scroll range of the scrollbar as set by `xvt_sbar_set_range`. For example, if the scroll range were set to (-100, 100), then legal values of `proportion` would be between zero and 200.

The usable range of the scrollbar decreases by the size of the scroll proportion. In general, if the range is set to (`range_min`, `range_max`)

and the proportion is set to `proportion`, then the range of possible scrollbar positions is from `range_min` to (`range_max` proportion).

For example, if the range were set to (-100, 100) and the proportion were set to 50, then the range of possible scrollbar positions would be (-100, 50).

Parameter Validity and Error Conditions

XVT issues an error if:

- `win` is not a valid `WINDOW`
- `win` is a control of type `WC_HSCROLL` or `WC_VSCROLL`; *and* `t` is not equal to `HVSCROLL`
- `win` is a `WINDOW` of type `W_DOC` or `W_PLAIN` (child window), *and* has been created with the appropriate scrollbar control as set by `WSF_HSCROLL` or `WSF_VSCROLL`; *and* `t` is not equal to `HSCROLL` or `VSCROLL`
- The thumb proportion exceeds the difference of the maximum and minimum range values of the scrollbar as set by `xvt_sbar_set_range`.

See Also

```
SCROLL_TYPE
xvt_sbar_get_proportion
xvt_sbar_set_pos
xvt_sbar_set_range
```

xvt_sbar_set_range

Set a Scrollbar's Range

Summary

```
void xvt_sbar_set_range(WINDOW win, SCROLL_TYPE t,
     int min, int max)
```

`WINDOW win`

Window whose scrollbar's range is being set.

`SCROLL_TYPE t`

Scrollbar type, horizontal or vertical.

`int min`

Minimum range value.

`int max`

Maximum range value.

Description

This function sets a scrollbar's minimum and maximum range values to `min` and `max`, respectively. On some platforms, this call leaves the thumb position undefined, so you must follow it with a call to `xvt_sbar_set_pos`.

If `t` is `HSCROLL` or `VSCROLL`, this function sets the range of the horizontal or vertical scrollbar on a regular or child window.

If `t` is `HVSCROLL`, then `win` is the `WINDOW` of a scrollbar control in the client area of a window or dialog.

The range of allowable scrollbar thumb positions is the range set by this function, reduced by the proportion set by `xvt_sbar_set_proportion`. In general, if the range is set to (`range_min`, `range_max`) and the proportion is set to `proportion`, then the range of possible scrollbar positions will be from `range_min` to (`range_max - proportion`).

Parameter Validity and Error Conditions

XVT issues an error if:

- `win` is not a valid `WINDOW`
- `win` is a control of type `WC_HSCROLL` or `WC_VSCROLL`; *and* `t` is not equal to `HVSCROLL`
- `win` is a `WINDOW` of type `W_DOC` or `W_PLAIN` (child window), *and* has been created with the appropriate scrollbar control as set by `WSF_HSCROLL` or `WSF_VSCROLL`; *and* `t` is not equal to `HSCROLL` or `VSCROLL`

Implementation Note

Ranges and positions of scrollbars are limited to -32,768 to + 32,767.

See Also

`*SCROLL Values for SCROLL_TYPE`
`xvt_sbar_get_range`
`xvt_sbar_set_pos`
`xvt_sbar_set_proportion`

xvt_scr_*

Screen Objects

```
xvt_scr_beep
xvt_scr_get_focus_topwin
xvt_scr_get_focus_vobj
xvt_scr_hide_cursor
xvt_scr_launch_browser
xvt_scr_list_wins
xvt_scr_set_busy_cursor
xvt_scr_set_focus_vobj
```

xvt_scr_beep

Beep at User

Summary

```
void xvt_scr_beep(void)
```

Description

This function makes a sound by ringing a bell or playing a tone through a speaker.

xvt_scr_get_focus_topwin

Get Top-Level Window with the Focus

Summary

```
WINDOW xvt_scr_get_focus_topwin(void)
```

Description

This function returns the XVT top-level window (not a dialog or that has the keyboard focus, or contains a control or child window that has the keyboard focus (i.e., the "active" top-level window). This function is used when an application needs to know which top-level window is currently active. For example, if your application keeps a menu containing a list of all of the top-level windows, it might want to check the currently active one on that menu.

A top-level window can become active by several means, including the user clicking on a titlebar or the client area of the window. Depending on the platform, the user can activate a window by choosing the window from a "task list" maintained by the operating system. In addition, your application can call `xvt_scr_set_focus_vobj` on the top-level window or some control or child window contained in it.

Note: If you want to find out which window has the focus (including child windows, controls, and dialogs), you should call `xvt_scr_get_focus_vobj` instead of calling `xvt_scr_get_focus_topwin`.

Return Value

The top-level window that has the focus if successful; `NULL_WIN` if no XVT top-level window has the focus.

See Also

`WINDOW`
`xvt_scr_get_focus_vobj`
`xvt_scr_set_focus_vobj`
`xvt_vobj_is_focusable`

---

xvt_scr_get_focus_vobj

Get Window with Focus

Summary

```
WINDOW xvt_scr_get_focus_vobj(void)
```

Description

This function reports which XVT window, if any (including dialogs, child windows, and controls), has the focus. Print windows and `XVT_PIXMAP`s can never have focus.

A window can become active by several means, including the user clicking on the titlebar, on the control, or on the client area of the window. The user can change the focus for a control by using the platform traversal key. On some platforms, the user can activate a window by choosing the window from a "task list" maintained by the window system. In addition, your application can call `xvt_scr_set_focus_vobj`.

There are many situations in which you will want to know which window has the focus. Keeping track of the focus is especially important when performing navigation among windows using the keyboard.

Return Value

`WINDOW` of the XVT window, dialog, or control that has the focus if successful; `NULL_WIN` if no XVT window has the focus.

Implementation Note

This function will never return the `SCREEN_WIN`, but on XVT/Win32, it can return the `TASK_WIN`.

See Also

```
SCREEN_WIN
TASK_WIN
WINDOW
xvt_scr_set_focus_vobj
xvt_vobj_is_focusable
```

xvt_scr_hide_cursor

Temporarily Hide Cursor

Summary

```
void xvt_scr_hide_cursor(void)
```

Description

This function makes the mouse cursor disappear. As soon as the user moves the mouse, the cursor reappears automatically.

It is good to call this function when the user starts typing text, so that the mouse cursor doesn't interfere with the blinking caret or with what the user is typing.

Implementation Note

This function has no effect on XVT/XM.

See Also

```
xvt_scr_set_busy_cursor
xvt_win_get_cursor
xvt_win_set_cursor
```

xvt_scr_launch_browser

Launch the OS Default Web Browser

Summary

```
BOOLEAN xvt_scr_launch_browser(const char *url)
```

Description

This function launches the default web browser as specified by the platform's Internet configuration preferences. The url is a NULL-terminated string containing the URL to be displayed once the browser has been launched. The URL must be fully qualified and adhere to composition standards, i.e. 'file:///C:/mydoc.htm' or 'http://www.xvt.com'.

To send a new URL once the browser has been launched, calling xvt_scr_launch_browser with a new URL will update the browser with the new URL.

Return Value

`TRUE` if successful, otherwise `FALSE`.

Parameter Validity and Error Conditions

XVT issues an error if:

- `url is NULL`

See Also

```
WC_HTML for WIN_TYPE
xvt_html_get_url
xvt_html_set_url
```

xvt_scr_list_wins

List Window Titles

Summary

```
SLIST xvt_scr_list_wins(void)
```

Description

This function gets an SLIST containing the titles of top-level windows and dialogs.

The data word in the SLIST associated with each window is its WINDOW object. This may be used as an argument to xvt_vobj_get_type to determine whether the window is an ordinary XVT window, a modal dialog, or a modeless dialog.

Return Value

SLIST containing titles if successful; NULL on error (out of memory). Data associated with each item is the WINDOW. You must free the SLIST created, when no longer needed, using xvt_slist_destroy.

Implementation Note

If the application is running on XVT/Win32 and the attribute ATTR_WIN_PM_DRAWABLE_TWIN has been set, the task window is included in the list.

Modal dialogs may have titles as defined in resource scripts. On some platforms, the titles of modal dialogs might not be visible on the screen.

See Also

SLIST
WINDOW
xvt_slist_destroy
xvt_vobj_get_type
xvt_win_enum_wins
xvt_win_list_wins

Example

This code adds the titles of all windows to a list box:

```
SLIST slist;
WINDOW lbox;
...
slist = xvt_scr_list_wins();
xvt_list_clear(lbox);
if (slist)
{
    int i, count;
    count = xvt_slist_count(slist);
    for (i = 0; i < count; i++)
    {
        char *title;
        WINDOW window;
        title = xvt_slist_get_elt(slist, i, &window);
        switch(xvt_vobj_get_type(window))
        {
        case W_DOC:
        case W_PLAIN:
        case W_DBL:
        case W_NO_BORDER:
            xvt_list_add(lbox, -1, title);
            break;
        default:
            break;
        }
    }
    xvt_slist_destroy(slist);
}
```

xvt_scr_set_busy_cursor

Change Cursor to Waiting Shape

Summary

```
void xvt_scr_set_busy_cursor(void)
```

Description

This function changes the CURSOR to a shape that indicates waiting (e.g., wristwatch, hourglass, etc.). The cursor remains that way until your application allows XVT to process the next event--either by returning from the current event handler or by making a call to XVT that causes events to be processed (such as creating a dialog or window). Your application should call `xvt_scr_set_busy_cursor` just before starting a long unbroken operation to let the user know that the computer hasn't crashed. When your long operation is done, you return from your event handler, and the cursor is restored.

Examples of long operations are reformatting a document, reading a document from a file, saving a document to a file, or recalculating a spreadsheet.

Do not use `xvt_win_set_cursor` as a substitute for `xvt_scr_set_busy_cursor`, for the following reasons:

- `xvt_win_set_cursor` operates on a particular window and the cursor does not change until the cursor enters that window
- `xvt_win_set_cursor` might not take effect immediately if the mouse cursor is not actually over the window

However, you should use `xvt_win_set_cursor` if you are writing an application where one window is performing a long operation while other windows remained active. In this case, you would use `xvt_win_set_cursor` to set the mouse pointer to `CURSOR_WAIT` for a particular window. This technique is somewhat advanced, and requires the use of `xvt_app_process_pending_events` or `xvt_timer_create` to create a multithreaded application.

Implementation Note

On XVT/XM, this function dispatches all update events to assure that the display is complete before the long application is started. During the `xvt_scr_set_busy_cursor` display, input into all application windows and dialogs is disabled.

See Also

```
CURSOR_* Options
xvt_app_process_pending_events
xvt_scr_hide_cursor
xvt_timer_create
xvt_win_get_cursor
xvt_win_set_cursor
```

xvt_scr_set_focus_vobj

Set Window Focus

Summary

```
void xvt_scr_set_focus_vobj(WINDOW win)

WINDOW win
```

Window, dialog, or control whose focus is being set.

Description

This function sets the keyboard input focus to a window, dialog, or control.

A typical use of this function is to implement navigation between controls in a window. To do that, you would respond to a navigation keystroke (tab and/or arrow keys on an `E_CHAR` event) by calling `xvt_scr_set_focus_vobj` on the next control in the navigation sequence.

Parameter Validity and Error Conditions

If the `WINDOW` or any of its ancestors are not visible or enabled, XVT issues a warning.

Implementation Note

This function can be used to change the focus between controls in either a window or dialog. However, setting the focus to controls of type `WC_TEXT`, `WC_ICON`, and `WC_GROUPBOX` is not permitted, and would not have meaning anyway. Calling this function during an `E_FOCUS` event results in undefined behavior on some platforms.

Normally, `TASK_WIN` is not a valid window. However, on XVT/Win32, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before calling `xvt_app_create`. In that case, `TASK_WIN` is a valid window for this function.

See Also

```
E_CHAR
E_FOCUS
TASK_WIN
WINDOW
xvt_app_create
xvt_scr_get_focus_vobj
xvt_vobj_is_focusable
xvt_vobj_raise
```

xvt_slist_*

List of Tagged Strings

```
xvt_slist_add_at_elt
xvt_slist_add_at_pos
xvt_slist_add_sorted
xvt_slist_count
xvt_slist_create
xvt_slist_debug
xvt_slist_destroy
xvt_slist_get
xvt_slist_get_data
xvt_slist_get_elt
xvt_slist_get_first
xvt_slist_get_next
xvt_slist_is_valid
xvt_slist_rem
```

xvt_slist_add_at_elt

Add String or `SLIST` to `SLIST`

Summary

```
BOOLEAN xvt_slist_add_at_elt(SLIST x, SLIST_ELT e,
     char *sx, long data)
```

`SLIST x`

`SLIST` on which to operate.

`SLIST_ELT e`

Element on which to operate.

`char *sx`

String or `SLIST` to add.

`long data`

Data to associate with the new item.

Description

This function adds a new element to the `SLIST x`. The `NULL`-terminated string `sx` gives the string part of the element, and `data` gives its data. This function positions the new element after the `e`

element, or at the end of the list if `e` is `NULL`. Memory is automatically allocated for the new element, and the contents of the string `sx` is copied into it.

The `sx` argument can also point to an existing `SLIST`, which is spliced into the list. In this case, `data` is ignored, XVT steals the contents of the `SLIST`, and `sx` is left pointing to an empty list. To release the empty list, call `xvt_slist_destroy` on the `sx` pointer after the call to `xvt_slist_add_at_elt`.

The `SLIST x` must already exist. You can create a fresh one using `xvt_slist_create`.

If `sx` is a string, its first character must *not* be equal to 255. (This is how `SLIST`s are distinguished from strings.)

Note: This function does not allow you to add elements to the beginning of an `SLIST`. However, you can use `xvt_slist_add_at_pos` to add a new element to an `SLIST` at a given position.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful (out of memory).

See Also

`SLIST`
`xvt_slist_add_at_pos`
`xvt_slist_create`
`xvt_slist_destroy`

The "Utilities" chapter in the *XVT Portability Toolkit Guide*

Example

```
SLIST x;if ((x = xvt_slist_create()) == NULL)
     .../* error handling not shown */
 if (!xvt_slist_add_at_elt(x, NULL, "element one", 1001))
     .../* error handling not shown */
 if (!xvt_slist_add_at_elt(x, NULL, "element two", 2001))
     .../* error handling not shown */
```

xvt_slist_add_at_pos

Add to an SLIST at a Given Position

Summary

```
BOOLEAN xvt_slist_add_at_pos(SLIST x, int index,
     char *sx, long data)
```

`SLIST x`

SLIST on which to operate.

`int index`

Element position index.

`char *sx`

String or SLIST to add.

`long data`

Data to associate with new item.

Description

This function adds a new element to the SLIST x. The NULL-terminated string sx gives the string part of the element, and data gives its data. The function adds its new element (sx) before the element indexed by index in the SLIST x. Memory is automatically allocated for the new element, and the contents of the string sx are copied into it.

The sx argument can also point to an existing SLIST, which is spliced into the list. In this case, data is ignored, XVT steals the contents of the SLIST, and sx is left pointing to an empty list. To release the empty list, call xvt_slist_destroy on the sx pointer after the call to xvt_slist_add_at_pos.

The SLIST x must already exist. You can create a fresh one with xvt_slist_create.

If sx is a string, its first character *must not* be equal to 255. (This is how SLISTs are distinguished from strings.)

Note: This function does allow you to add elements to the beginning of an SLIST by specifying index = 0. If index is greater than the index of the last element in x, the string, or the SLIST, sx is added to the end of SLIST x.

Return Value

`TRUE` on success; `FALSE` if unsuccessful (out of memory).

See Also

```
SLIST
xvt_slist_add_at_elt
xvt_slist_create
xvt_slist_destroy
```

The "Utilities" chapter in the *XVT Portability Toolkit Guide*

Example

```
SLIST x;if ((x = xvt_slist_create()) == NULL)
     .../* error handling not shown */
 if (!xvt_slist_add_at_elt(x, NULL, "element two", 2001))
     .../* error handling not shown */
 if (!xvt_slist_add_at_pos(x, 0, "element one", 1001))
     .../* error handling not shown */
```

xvt_slist_add_sorted

Add String to Sorted `SLIST`

Summary

```
BOOLEAN xvt_slist_add_sorted(SLIST x, char *s,
     long data, BOOLEAN unique, BOOLEAN case_sensitive)
```

`SLIST x`

`SLIST` on which to operate.

`char *s`

String to be added.

`long data`

Data to associate with the new item.

`BOOLEAN unique`

If `TRUE`, and if an element with a matching string is already present, the new element is not added.

`BOOLEAN case_sensitive`

Determines whether the position comparison is case-sensitive.

Description

This function adds a new element to the `SLIST x`. The string part of the element is given by the `NULL`-terminated string `s`, and its data is given by `data`. The new element is positioned in lexically increasing order with respect to the elements already in the `SLIST`. Memory is automatically allocated for the new element and the contents of the string `s` are copied into it.

The comparison to determine the position for the new element is case-sensitive only if `case_sensitive` is `TRUE`. If `unique` is `TRUE` the new element is not added only if an element with a matching string is already present (the case sensitivity of this match is also determined by `case_sensitive`). The function set by the attribute `ATTR_COLLATE_HOOK` affects the sorting of items added to the `SLIST`.

The `SLIST x` must already exist. You can create a fresh one using `xvt_slist_create`.

Return Value

`TRUE` on success; `FALSE` on error (out of memory).

See Also

```
ATTR_COLLATE_HOOK
SLIST
xvt_slist_create
```

The "Utilities" chapter in the *XVT Portability Toolkit Guide*

Example

```
SLIST x;
char *s;
long n;if ((x = xvt_slist_create()) == NULL)
     .../* error handling not shown */
while ((s = get_data(&n)) != NULL)
    if (!xvt_slist_add_sorted(x, s, n, FALSE, TRUE))
         .../* error handling not shown */
```

xvt_slist_count

Count Elements in `SLIST`

Summary

```
int xvt_slist_count(SLIST x)

SLIST x
```

`SLIST` on which to operate.

Description

This function counts the number of elements in the `SLIST x`.

Return Value

Number of elements (0 if `x` is `NULL` or empty).

See Also

`SLIST`

The "Utilities in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_scr_list_wins`.

---

xvt_slist_create

Create New `SLIST`

Summary

```
SLIST xvt_slist_create(void)
```

Description

This function allocates a new `SLIST` with no elements.

Some XVT functions (such as `xvt_fsys_list_files`) also allocate `SLIST`s by calling `xvt_slist_create` themselves.

Return Value

`SLIST` if successful; `NULL` on error (out of memory). You must free the created `SLIST` with `xvt_slist_destroy`.

See Also

`SLIST`
`xvt_slist_destroy`

The "Utilities" chapter in the *XVT Portability Toolkit Guide*

Example

```
SLIST x;if ((x = xvt_slist_create()) == NULL)
     .../* error handling not shown */
xvt_slist_destroy (x);
```

xvt_slist_debug

Append Dump of SLIST to File

Summary

```
void xvt_slist_debug(SLIST x)
```

SLIST x

SLIST on which to operate.

Description

This function dumps a representation of SLIST x to the "debug" file (using function xvt_debug_printf). Each string and its associated data is shown. If the SLIST x parameter is NULL, this is noted in the debug file.

See Also

ATTR_DEBUG_FILENAME
SLIST
xvt_debug_printf

The "Utilities" chapter in the *XVT Portability Toolkit Guide*

Example

```
SLIST x;if ((x = xvt_scr_list_wins()) == NULL)
     .../* error handling not shown */
xvt_slist_debug(x);
xvt_slist_destroy(x);
```

xvt_slist_destroy

Free SLIST Storage

Summary

```
void xvt_slist_destroy(SLIST x)
```

SLIST x

SLIST on which to operate. If x is NULL, this function simply returns.

Description

This function frees the memory occupied by `SLIST x`, which then is no longer a valid `SLIST`. Nothing is done with the data associated with each element. If the data values are in fact pointers that have to be freed, you must do this prior to calling `xvt_slist_destroy`.

See Also

`SLIST`
`xvt_slist_create`

The "Utilities" chapter in the *XVT Portability Toolkit Guide*

Example

The following code shows a loop that frees the memory pointed to by each element of an `SLIST` (presumably it was allocated when the element was originally added). Then the `SLIST` itself is freed. The variable `x` is set to `NULL` to make sure its value (which is no longer valid) is never used by accident.

```
SLIST x;
SLIST_ELT e;
char *p;

...          /* code to add elements to SLIST not shown */
for (e = xvt_slist_get_first(x); e != NULL; e =
        xvt_slist_get_next(x, e)) {
    if (xvt_slist_get(x, e, (long *)&p) == NULL
        ...  /* error handling not shown */
    }
    xvt_mem_free(p);
}
xvt_slist_destroy(x);
x = NULL;
```

Note: For other examples that use xvt_slist_destroy, see `xvt_scr_list_wins`, `xvt_res_get_str_list`, `xvt_list_get_sel`, and `xvt_list_get_all`.

xvt_slist_get

Get String and Data from `SLIST` Element

Summary

```
char *xvt_slist_get(SLIST x, SLIST_ELT e, long *datap)

SLIST x
```

`SLIST` on which to operate.

`SLIST_ELT e`

Element on which to operate.

`long *datap`

Address to hold data associated with the element. If it is non-`NULL`, the data associated with the element is placed into `datap`. If it is `NULL`, no `data` is placed into `datap`.

Description

This function retrieves the string and data (returned through `datap`) associated with element `e` in `SLIST x`. It is usually used in a `for` loop that calls `xvt_slist_get_first` and `xvt_slist_get_next`. To change the data associated with an element, use `xvt_slist_get_data`.

Return Value

Pointer to string if successful; `NULL` if `e` is `NULL`. Note that the pointer refers to storage in the `SLIST`. Do not attempt to free this pointer yourself. If the `e` element parameter is `NULL`, `NULL` is returned as the function value.

See Also

`SLIST`
`SLIST_ELT`
`xvt_slist_get_data`
`xvt_slist_get_first`
`xvt_slist_get_next`

The "Utilities" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_slist_get_first`.

xvt_slist_get_data

Get Data Associated with `SLIST` Element

Summary

```
long *xvt_slist_get_data(SLIST_ELT e)

SLIST_ELT e
```

Data to be retrieved.

Description

This function retrieves the address of the data associated with the `SLIST` element `e`.

Return Value

Address of associated `SLIST` data.

See Also

`SLIST_ELT`
`xvt_slist_get`

The "Utilities" chapter in the *XVT Portability Toolkit Guide*

Example

This code illustrates how you can change the data element by putting this function on the left side of an assignment operation:

```
*xvt_slist_get_data(elt) = d;
```

xvt_slist_get_elt

Get String and Data from `SLIST` Element

Summary

```
char *xvt_slist_get_elt(SLIST x, int index,
     long *datap)
```

`SLIST x`

`SLIST` on which to operate.

`int index`

Index position of element.

`long *datap`

Address to hold the associated data. If it is non-`NULL`, the data associated with the element is placed into `datap`. If it is `NULL`, no data is placed into `datap`.

Description

This function retrieves the string and data (returned through `datap`) associated with element numbered `index` (origin 0) in the `SLIST x`.

The requested element isn't located particularly quickly, as there is no index table of element pointers. The function simply traverses the list element-by-element until it finds the one numbered `index`.

Return Value

Pointer to string if successful; `NULL` if element not found. Note that the pointer refers to storage within the `SLIST`. Do not attempt to free this pointer yourself.

See Also

`SLIST`
`xvt_slist_get`
`xvt_slist_get_data`
`xvt_scr_list_wins`

The "Utilities" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_scr_list_wins`.

xvt_slist_get_first

Get First Element in `SLIST`

Summary

```
SLIST_ELT xvt_slist_get_first(SLIST x)
```

`SLIST x`

`SLIST` on which to operate.

Description

This function retrieves the first element of `SLIST x` and returns a reference to it of type `SLIST_ELT`. The string and data components of the `SLIST_ELT` can then be obtained by calling `xvt_slist_get`.

This function is almost always executed in the first expression of a loop that cycles through all elements. The "incrementation" expression in the loop is normally a call to `xvt_slist_get_next`.

Return Value

First element if successful; `NULL` if `x` is `NULL` or empty.

See Also

`SLIST`
`SLIST_ELT`
`xvt_slist_get`
`xvt_slist_get_next`

The "Utilities" chapter in the *XVT Portability Toolkit Guide*

Example

```
SLIST x;
SLIST_ELT e;
struct ... {
...
} *state;
char *name;
long ltype;
struct state st;..../* code to initialize state not shown
*/if ((x = xvt_fsys_list_files(state->type, state->pat,
        state->dirs)) == NULL)
    xvt_dm_post_error("Can't list files.");
for (e = xvt_slist_get_first(x); e != NULL;
        e = xvt_slist_get_next(x, e)) {
    if (stat(name = xvt_slist_get(x, e, &ltype), &st)
        == -1) {
        xvt_dm_post_error("Can't get stat info.");
        break;
    }
```

xvt_slist_get_next

Get Next Element of `SLIST`

Summary

`SLIST_ELT xvt_slist_get_next(SLIST x, SLIST_ELT e)`

`SLIST x`

`SLIST` on which to operate.

`SLIST_ELT e`

Element prior to the element on which to operate.

Description

This function retrieves the element in `x` following element `e`. It is usually used in the "incrementation" expression of a `for` loop that calls `xvt_slist_get_first` in its first expression. The string and

data components of the retrieved element can be obtained by calling `xvt_slist_get`.

Return Value

The next element if successful; `NULL` if no elements remain, or if `e` is `NULL`. If the element parameter is `NULL`, `NULL` is returned as the function value.

See Also

`SLIST`
`SLIST_ELT`
`xvt_slist_get`
`xvt_slist_get_first`

The "Utilities" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_slist_get_first`.

---

xvt_slist_is_valid

Test if `SLIST` Reference is Valid

Summary

```
BOOLEAN xvt_slist_is_valid(SLIST x)
```

`SLIST x`

`SLIST` object or pointer to be tested.

Description

Given an `SLIST` object or pointer `x`, this function tests whether it is an `SLIST`. Generally, this test is reliable only if `x` is a genuine `SLIST` or a string.

Return Value

`TRUE` if `x` is a real `SLIST`; `FALSE` otherwise.

See Also

`SLIST`

The "Utilities" chapter in the *XVT Portability Toolkit Guide*

xvt_slist_rem

Remove Element of SLIST

Summary

```
BOOLEAN xvt_slist_rem(SLIST x, SLIST_ELT e)
```

`SLIST x`

`SLIST` on which to operate.

`SLIST_ELT e`

Element on which to operate.

Description

This function removes element `e` from the `SLIST x` and frees the memory it occupies. If its data word is a pointer to memory that also has to be freed, your application must take care of freeing that memory.

Return Value

`TRUE` if successful; `FALSE` if `e` is not in the `SLIST`.

See Also

`SLIST`
`SLIST_ELT`

The "Utilities" chapter in the *XVT Portability Toolkit Guide*

Example

```
/* remove first element from an SLIST */
SLIST x;
SLIST_ELT e;.../* code to build up SLIST not shown */
if ((e = xvt_slist_get_first(x)) == NULL)
     .../* error handling not shown */
 if (!xvt_slist_rem(x, e))
     .../* error handling not shown */
```

xvt_str_*

String Operations

xvt_str_collate
xvt_str_collate_ignoring_case
xvt_str_compare
xvt_str_compare_ignoring_case
xvt_str_compare_n_char
xvt_str_concat
xvt_str_concat_n_char
xvt_str_convert_mb_to_wc
xvt_str_convert_mbs_to_wcs
xvt_str_convert_to_lower
xvt_str_convert_to_upper
xvt_str_convert_wc_to_mb
xvt_str_convert_wchar_to_lower
xvt_str_convert_wchar_to_upper
xvt_str_convert_wcs_to_mbs
xvt_str_copy
xvt_str_copy_n_char
xvt_str_copy_n_size
xvt_str_create_codeset_map
xvt_str_destroy_codeset_map
xvt_str_duplicate
xvt_str_find_char_set
xvt_str_find_eol
xvt_str_find_first_char
xvt_str_find_last_char
xvt_str_find_not_char_set
xvt_str_find_substring
xvt_str_find_token
xvt_str_get_byte_count
xvt_str_get_char_count
xvt_str_get_char_size
xvt_str_get_n_char_count
xvt_str_get_n_char_size
xvt_str_get_next_char
xvt_str_get_prev_char
xvt_str_is_alnum
xvt_str_is_alpha
xvt_str_is_digit
xvt_str_is_equal

```
xvt_str_is_invariant
xvt_str_is_lower
xvt_str_is_space
xvt_str_is_upper
xvt_str_is_xdigit
xvt_str_match
xvt_str_parse_double
xvt_str_parse_long
xvt_str_parse_ulong
xvt_str_sprintf and xvt_str_vsprinf
xvt_str_translate_codeset
```

xvt_str_collate

Compare Multibyte Strings

Summary

```
int xvt_str_collate(const char *mbs1, const char *mbs2)
```

`const char *mbs1`

First string to compare.

`const char *mbs2`

Second string to compare.

Description

This function compares the multibyte string `mbs1` with the multibyte string `mbs2`. It compares the strings interpreting both strings as appropriate to the current locale. If the attribute `ATTR_COLLATE_HOOK` is set, then the application collate function is called, otherwise an XVT internal collate function is called. This function is the multibyte replacement for the ANSI function `strcoll`.

Return Value

Positive value if `mbs1` is greater than `mbs2`; negative value if `mbs1` is less than `mbs2`; zero if they are equal in value and length.

See Also

```
ATTR_COLLATE_HOOK
xvt_str_collate_ignoring_case
xvt_str_compare
```

Example

```
if (xvt_str_collate(list[i], list[i+1]) > 0) {
/* swap strings */
    char* temp;temp = list[i];
    list[i] = list[i+1];
    list[i+1] = temp;
}
```

xvt_str_collate_ignoring_case

Compare Multibyte Strings Ignoring Case of Character Set

Summary

```
int xvt_str_collate_ignoring_case(const char *mbs1,
    const char *mbs2)
```

`const char *mbs1`

First string to compare.

`const char *mbs2`

Second string to compare.

Description

This function compares the multibyte string `mbs1` with the multibyte string `mbs2` ignoring the case for those character sets that do distinguish between upper and lowercase. It compares the strings interpreting both strings as appropriate to the current locale. If the attribute `ATTR_COLLATE_HOOK` is set, then the application collate function is called, otherwise an XVT internal collate function is called.

Return Value

Positive value if `mbs1` is greater than `mbs2`; negative value if `mbs1` is less than `mbs2`; zero if they are equal in value and length.

See Also

`ATTR_COLLATE_HOOK`
`xvt_str_collate`
`xvt_str_compare_ignoring_case`

xvt_str_compare

Compare Numeric Value of Multibyte String Characters

Summary

```
int xvt_str_compare(const char *mbs1, const char *mbs2)
```

`const char *mbs1`

First string to compare.

`const char *mbs2`

Second string to compare.

Description

This function compares the multibyte string `mbs1` with the multibyte string `mbs2`. It compares the numeric values of the characters in the strings. It is the multibyte replacement for the ANSI function `strcmp`.

Return Value

Positive value if `mbs1` is greater than `mbs2`; negative value if `mbs1` is less than `mbs2`; zero if they are equal in value and length.

See Also

`xvt_str_collate`
`xvt_str_compare_ignoring_case`
`xvt_str_match`

xvt_str_compare_ignoring_case

Compare Numeric Value of Multibyte String Characters Ignoring Case of Character Set

Summary

```
int xvt_str_compare_ignoring_case(const char *mbs1,
     const char *mbs2)
```

`const char *mbs1`

First string to compare.

`const char *mbs2`

Second string to compare.

Description

This function compares the multibyte string `mbs1` with the multibyte string `mbs2`. It works identically to the standard C function `strcmp`, except for ignoring the case. This function differs from `xvt_str_match` in that it allows the comparison of strings containing "`*`" and "`?`" characters.

Return Value

Positive value if `mbs1` is greater than `mbs2`; negative value if `mbs1` is less than `mbs2`; zero if they are equal in value and length.

See Also

`xvt_str_collate_ignoring_case`
`xvt_str_compare`
`xvt_str_match`

---

xvt_str_compare_n_char

Compare n Characters of Multibyte String

Summary

```
int xvt_str_compare_n_char(const char *mbs1,
     const char *mbs2, const size_t n)
```

`const char *mbs1`

First string to compare.

`const char *mbs1`

Second string to compare.

`const size_t n`

Number of strings to compare.

Description

This function compares `n` characters of the multibyte string `mbs1` with the multibyte string `mbs2`. It compares the numeric values of the characters in the strings until `n` characters have been compared or the end of one of the strings is reached. It is the multibyte replacement for the ANSI function `strncmp`.

Return Value

Positive value if `mbs1` is greater than `mbs2`; negative value if `mbs1` is less than `mbs2`; zero if they are equal in value and length.

See Also

`xvt_str_compare`

xvt_str_concat

Append Multibyte Strings

Summary

```
char *xvt_str_concat(char *mbs1, const char *mbs2)
```

`char *mbs1`

First string.

`const char *mbs2`

Second string.

Description

This function appends a copy of the multibyte string `mbs2` to the end of the multibyte string `mbs1`, overwriting the `NULL` character at the end of `mbs1`. It is the multibyte replacement for the ANSI function `strcat`.

Return Value

Modified `mbs1` (sufficient memory in `mbs1` is required).

See Also

`xvt_str_concat_n_char`

xvt_str_concat_n_char

Append n Characters of Multibyte Strings

Summary

```
char *xvt_str_concat_n_char(char *mbs1,
     const char *mbs2, const size_t n)
```

`char *mbs1`

First multibyte string.

`const char *mbs2`

Second multibyte string.

`const size_t n`

Number of characters to append from the second string.

Description

This function appends a copy of `n` multibyte characters from the string `mbs2` to the end of the string `mbs1`, overwriting the `NULL` character at the end of `mbs1`. It copies until `n` multibyte characters or the end of string `mbs2` is reached. This function is the multibyte replacement for the ANSI function `strncat`.

Return Value

Modified `mbs1` (sufficient memory in `mbs1` is required).

See Also

`xvt_str_concat`

xvt_str_convert_mb_to_wc

Convert First Character of Multibyte String to Wide Character

Summary

```
int xvt_str_convert_mb_to_wc(XVT_WCHAR * wc,
     const char * mbs)
```

`XVT_WCHAR * wc`

Pointer to wide character.

`const char * mbs`

Multibyte string.

Description

This function converts the first character of the multibyte string `mbs` to the wide character `wc`. It replaces the ANSI function `mbtowc`.

Return Value

Zero if `mbs` is `NULL`, or if `mbs` is the `NULL` character '<fc0', or if `wc` is `NULL`; -1 if `mbs` does not point to a valid multibyte character; size in bytes of the multibyte character otherwise.

See Also

`xvt_str_convert_mbs_to_wcs`
`xvt_str_convert_wc_to_mb`

Example

```
int i, len;
XVT_WCHAR WC;for (i = 0; str[i]; i += len) {
    len = xvt_str_convert_mb_to_wc(&wc, &str[i]);
    if (len <= 0) break;
    if (wc == wc_text) {
        ...
    }
}
```

xvt_str_convert_mbs_to_wcs

Convert Multibyte Character String to Wide Character String

Summary

```
int xvt_str_convert_mbs_to_wcs(XVT_WCHAR *wcs,
    const char *mbs, const size_t n)
```

`XVT_WCHAR *wcs`

Pointer to an array of wide characters.

`const char *mbs`

Multibyte string to convert.

`const size_t n`

Number of characters to convert.

Description

This function converts the characters in the multibyte string `mbs` to the wide character string `wcs`, stopping after it copies `n` characters or the end of `mbs` is reached. If a `NULL` character is encountered before `n` characters are converted, the `NULL` is converted and processing terminates. This function replaces the ANSI function `mbstowcs`.

Return Value

The number of characters converted, not including the `NULL` character if successful. -1 if an invalid multibyte character is encountered. Sufficient memory in `wcs` is required to hold the wide character string.

See Also

`xvt_str_convert_mb_to_wc`
`xvt_str_convert_wcs_to_mbs`

---

xvt_str_convert_to_lower

Convert First n Bytes in Multibyte String to Lowercase Characters

Summary

```
size_t xvt_str_convert_to_lower(char *mbs1,
     const char *mbs2, const size_t n)
```

`char *mbs1`

Target string.

`const char *mbs2`

String to convert.

`const size_t n`

Number of bytes to convert.

Description

This function converts the first `n` bytes in the multibyte string `mbs2` to lowercase characters and copies them into the multibyte string `mbs1`. If a `NULL` character is encountered before `n` bytes are processed, the `NULL` is copied to the end of `mbs1` and processing terminates.

Return Value

The number of bytes processed. The values of `mbs1` and `mbs2` can refer to the same string.

See Also

`xvt_str_convert_to_upper`
`xvt_str_convert_wchar_to_lower`

---

xvt_str_convert_to_upper

Convert First n Bytes in Multibyte String to Uppercase Characters

Summary

```
size_t xvt_str_convert_to_upper(char *mbs1,
     const char *mbs2, size_t n)
```

`char *mbs1`

Target string.

`const char *mbs2`

String to convert.

`size_t n`

Number of bytes to convert.

Description

This function converts the first `n` bytes in the multibyte string `mbs2` to uppercase characters and copies them into the multibyte string `mbs1`. If a `NULL` character is encountered before `n` bytes are processed the `NULL` is copied to the end of `mbs1` and processing terminates.

Return Value

The number of bytes processed. The values of `mbs1` and `mbs2` can refer to the same string.

See Also

`xvt_str_convert_to_lower`
`xvt_str_convert_wchar_to_lower`

xvt_str_convert_wc_to_mb

Convert Wide Character to Multibyte Character

Summary

```
int xvt_str_convert_wc_to_mb(char * mbs,
     const XVT_WCHAR wc)
```

`char * mbs`

Target string.

`const XVT_WCHAR wc`

Wide character to convert.

Description

This function converts the wide character `wc` to a multibyte character in the multibyte array pointed to by `mbs`. It also returns the size in bytes of the multibyte character. Sufficient memory in `mbs` is required to hold a multibyte character. The string `mbs` is not `NULL`-terminated unless the `wc` character is the `NULL` character. This function replaces the ANSI function `wctomb`.

Return Value

Zero if `mbs` is `NULL` or `wc` is the `NULL` character; -1 if `wc` is not a valid wide character; size in bytes otherwise.

See Also

`xvt_str_convert_mb_to_wc`
`xvt_str_convert_wcs_to_mbs`

Example

```
case E_CHAR:
    if (!xvt_event_is_virtual_key(ep)) {
        int len;
        char* mbc[XVT_MAX_MB_SIZE];len =
            xvt_str_convert_wc_to_mb(
            mbc, ep->v.chr.ch);
        if (len > 0) {
            int i;
            for (i = 0; i < len; i++)
                line[offset++] = mbc[i];
        }
    }
```

xvt_str_convert_wchar_to_lower

Convert Wide Character to Lowercase Wide Character

Summary

```
XVT_WCHAR xvt_str_convert_wchar_to_lower(XVT_WCHAR wc)
```

`XVT_WCHAR wc`

Wide character.

Description

This function converts the wide character `wc` to a lowercase wide character. It is the wide character replacement for the ANSI function `tolower`.

Return Value

Lowercase wide character.

See Also

`xvt_str_convert_to_lower`
`xvt_str_convert_wchar_to_upper`

xvt_str_convert_wchar_to_upper

Converts Wide Character to Uppercase Wide Character

Summary

```
XVT_WCHAR xvt_str_convert_wchar_to_upper(XVT_WCHAR wc)
```

`XVT_WCHAR wc`

Wide character.

Description

This function converts the wide character `wc` to an uppercase wide character. It is the wide character replacement for the ANSI function `toupper`.

Return Value

Uppercase wide character.

See Also

```
xvt_str_convert_to_upper
xvt_str_convert_wchar_to_lower
```

xvt_str_convert_wcs_to_mbs

Converts Wide Character String to Multibyte String

Summary

```
int xvt_str_convert_wcs_to_mbs(char *mbs,
     const XVT_WCHAR *wcs, const size_t n)
```

`char *mbs`

Target string.

`const XVT_WCHAR *wcs`

Pointer to an array of wide characters.

`const size_t n`

Size of `mbs` in bytes.

Description

This function converts characters in the wide character string `wcs` to the multibyte string `mbs`, stopping when the next stored multibyte character exceeds the limit of `n` total bytes in `mbs` or if a `NULL` character is stored. It replaces the ANSI function `wcstombs`.

Return Value

Number of bytes written in `mbs`, not including the `NULL` character (sufficient memory in `mbs` is required to hold the multibyte string).

See Also

```
xvt_str_convert_mbs_to_wcs
xvt_str_convert_wc_to_mb
```

xvt_str_copy

Copies One Multibyte String into Another

Summary

```
char *xvt_str_copy(char *mbs1, const char *mbs2)
```

`char *mbs1`

Target string.

`const char *mbs2`

Source string.

Description

This function copies the multibyte string `mbs2` into the multibyte string `mbs1` including the terminating `NULL`. It is the multibyte replacement for the ANSI function `strcpy`.

Return Value

Modified `mbs1` (sufficient memory in `mbs1` is required to hold the multibyte string).

See Also

`xvt_str_copy_n_char`
`xvt_str_copy_n_size`

xvt_str_copy_n_char

Copies n Characters from One Multibyte String into Another

Summary

```
size_t xvt_str_copy_n_char(char *mbs1,
     const char *mbs2, size_t n)
```

`char *mbs1`

Target string.

`const char *mbs2`

Source string.

`size_t n`

Number of characters to copy.

Description

This function copies `n` characters from the multibyte string `mbs2` into the multibyte string `mbs1`. It copies until `n` multibyte characters or the end of string `mbs2` is reached. If the number of characters in `mbs2` is less than `n`, a `NULL` character is appended to the end of `mbs1`.

This function is the multibyte replacement for the ANSI function `strncpy`. Note some differences, however. The function takes a character count, not a byte count, and does not pad the unused area with `NULL` characters. It also returns a `size_t` for the number of bytes (not characters) actually copied, including a `NULL` character if one was copied.

Return Value

Number of bytes actually copied.

See Also

`xvt_str_copy`
`xvt_str_copy_n_size`

---

xvt_str_copy_n_size

Copy n Bytes from one Multibyte String to Another

Summary

```
size_t xvt_str_copy_n_size (char * mbs1,
     const char * mbs2, size_t n)
```

`char * mbs1`

Target string.

`const char * mbs2`

Source string.

`size_t n`

Number of bytes to copy.

Description

This function copies `n` bytes from the multibyte string `mbs2` into the multibyte string `mbs1`. If the number of bytes in `mbs2` is less then `n`, a `NULL` character is appended at the end of `mbs1`. If the number of

bytes does not fall on a character boundary, then the string is copied to the last character boundary before `n` bytes.

This function is the multibyte replacement for the ANSI function `strncpy`.Note some differences, however. The function does not pad the unused area with `NULL` characters. It also returns a `size_t` for the number of bytes (not characters) actually copied, including a `NULL` character if one was copied.

Return Value

Number of bytes copied (sufficient memory in `mbs1` is required).

See Also

`xvt_str_copy`
`xvt_str_copy_n_char`

---

xvt_str_create_codeset_map

Creates an `xvt_codeset_map` From Two Codeset Map Files

Summary

```
XVT_CODESET_MAP xvt_str_create_codeset_map(
    XVT_IOSTREAM fromcodeset, XVT_IOSTREAM tocodeset)
```

`XVT_IOSTREAM fromcodeset`

IO stream to read the compiled codeset file to map from.

`XVT_IOSTREAM tocodeset`

IO stream to read the compiled codeset file to map to.

Description

This function loads the binary codeset table (**.bct**) map file from which to map text into Unicode and the binary codeset table map file from which to map text into from Unicode. The IO streams must be open for reading appropriately formatted data. The `XVT_CODESET_MAP` can be set up to map to or from Unicode by passing `NULL` for the appropriate `XVT_IOSTREAM`.

The **.bct** map files are generated by the compiler utility or **maptabc.app**.

See the Reference section "Tools" for more information on **maptabc**.

Return Value

An XVT_CODESET_MAP object is returned if successful; NULL if an error occurred.

Parameter and Validity Conditions

XVT returns NULL if any of the following conditions occur:

- If both the XVT_IOSTREAM parameters are NULL.
- If either of the XVT_IOSTREAM parameters do not refer to data in the **.bct** file format or the format of the data selected is an older format and needs to be recompiled.
- There is insufficient memory to create the XVT_CODESET_MAP object.

See Also

XVT_CODESET_MAP
xvt_str_destroy_codeset_map
xvt_str_translate_codeset

xvt_str_destroy_codeset_map

Destroys an XVT_CODESET_MAP and Frees Associated Memory

Summary

```
void xvt_str_destroy_codeset_map(
    XVT_CODESET_MAP codeset_map)
```

XVT_CODESET_MAP codeset_map

Codeset map to destroy.

Description

This function destroys an XVT_CODESET_MAP object and frees all associated memory.

Parameter and Validity Conditions

XVT issues an error if codeset_map is NULL or is not a valid XVT_CODESET_MAP.

See Also

XVT_CODESET_MAP
xvt_str_create_codeset_map
xvt_str_translate_codeset

xvt_str_duplicate

Duplicate Multibyte String and Allocate New Memory

Summary

```
char *xvt_str_duplicate(const char * mbs)
```

`const char * mbs`

String to duplicate.

Description

This function allocates new memory with `xvt_mem_alloc` and copies the multibyte string `mbs` including the terminating `NULL`. The string must be freed by the caller with `xvt_mem_free`.

Return Value

The newly allocated memory; `NULL` if insufficient memory is available or if `mbs` is `NULL`.

See Also

`xvt_mem_alloc`
`xvt_mem_free`
`xvt_str_copy`

xvt_str_find_char_set

Search Multibyte String for Character

Summary

```
char *xvt_str_find_char_set(const char *mbs,
     const char *mbset)
```

`const char *mbs`

String to search.

`const char *mbset`

Set of characters for which to search.

Description

This function searches the multibyte string `mbs` for the first occurrence of a multibyte character that is also in the multibyte string `mbset`. It is the multibyte replacement for the ANSI function `strcspn`.

Return Value

Pointer to the first occurrence of `mbc` in `mbs`; `NULL` if `mbc` is not found in `mbs`.

See Also

`xvt_str_find_first_char`
`xvt_str_find_not_char_set`

---

xvt_str_find_eol

Find End-of-Line Character in Multibyte String

Summary

```
char *xvt_str_find_eol(const char *mbs, const long n,
     long *lenp, EOL_FORMAT *fp)
```

`const char *mbs`

Pointer to the lines of the text.

`const long n`

Number of bytes in the specified text.

`long *lenp`

Length of the line.

`EOL_FORMAT *fp`

Type of end-of-line sequence.

Description

This function breaks a collection of text lines pointed to by `mbs` into individual lines.

`n` is the total number of bytes pointed to by `mbs`, including the end-of-line sequences separating the lines and ending the last line. The collection of text lines need not be `NULL`-terminated; if it is, the `NULL` byte should not be included in the `n` count.

The first time your application calls `xvt_str_find_eol` with the address of the lines of text, it scans the text for an end-of-line sequence (not necessarily the one used by the local operating system). The second and subsequent calls must be made with the mbs argument set to `NULL`, which tells `xvt_str_find_eol` to continue scanning from where it left off. The `n` argument is unused when `mbs` is `NULL`.

Each time `xvt_str_find_eol` is called, it returns a pointer to the next line in the buffer `mbs` and stores its length, excluding the end-of-line sequence, in the `lenp` argument. It also determines the type of line-ending sequence found, and stores the result in `fp`. These are the possible values returned in `fp`:

`EOL_NORMAL`

For the first call, this symbol means that the line was terminated with a normal end-of-line sequence for some XVT environment, but not necessarily the local one. For subsequent calls, this means that the end-of-line sequence was the same as that ending the first line (i.e., the lines are terminated consistently).

`EOL_DIFF`

The line terminated with an end-of-line sequence different from the one terminating the first line.

`EOL_NONE`

The line did not terminate with an end-of-line sequence. This can be true only for the last line.

The data pointed to by `mbs` is not tampered with in any way. Lines pointed to by returned values are not `NULL`-terminated--you must use the stored length to find their ends. `NULL` is returned when no lines remain.

If you don't want the length of a returned line, you can use a lenp argument of `NULL`. Similarly, a `NULL fp` argument suppresses a returned `EOL_FORMAT`.

Note: It is not necessary to use `xvt_str_find_eol` to break text read from a file into separate lines. The standard C runtime library always converts native end-of-line sequences into "<fcn" on reading.

Return Value

Pointer to line (not `NULL`-terminated, with end-of line-sequence intact) if successful; `NULL` if no lines remain.

See Also

```
EOL_* Values for EOL_FORMAT
EOL_SEQ
```

Example

This code adds text lines from a buffer to a list box:

```
    WINDOW lbox;
    char *buf;
    char *line;
    long len;
    ...
    xvt_list_clear(lbox);
    line = xvt_str_find_eol(buf, strlen(buf), &len,
NULL);
    while (line != NULL) {
         l[len] =’0’;
        xvt_list_add(lbox, -1, line;
        line = xvt_str_find_eol(NULL, 0, &len, NULL);
     }
```

xvt_str_find_first_char

Find First Character in Multibyte String

Summary

```
char *xvt_str_find_first_char(const char *mbs,
     const char *mbc)
```

`const char *mbs`

String to search.

`const char *mbc`

Multibyte character for which to search.

Description

This function searches the multibyte string `mbs` for the first occurrence of the multibyte character in `mbc`. It is the multibyte replacement for the ANSI function `strchr`.

Return Value

A pointer to the first occurrence of `mbc` in `mbs`; a pointer to the terminating `NULL` character in `mbs` if `mbc` is the `NULL` character; `NULL` if `mbc` is not found in `mbs`.

See Also

```
xvt_str_find_char_set
xvt_str_find_last_char
```

xvt_str_find_last_char

Find Last Character in Multibyte String

Summary

```
char *xvt_str_find_last_char(const char *mbs,
     const char *mbc)
```

`const char *mbs`

String to search.

`const char *mbc`

Multibyte character for which to search.

Description

This function searches the multibyte string `mbs` for the last occurrence of the multibyte character in `mbc`. It is the multibyte replacement for the ANSI function `strrchr`.

Return Value

A pointer to the last occurrence of `mbc` in `mbs`; a pointer to the terminating `NULL` character in `mbs` if `mbc` is the `NULL` character; `NULL` if `mbc` is not found in `mbs`.

See Also

```
xvt_str_find_char_set
xvt_str_find_first_char
```

xvt_str_find_not_char_set

Search Multibyte String for Character not in Set

Summary

```
char *xvt_str_find_not_char_set(const char *mbs,
     const char *mbset)
```

`const char *mbs`

String to search.

`const char *mbset`

Set of characters.

Description

This function searches the multibyte string `mbs` for the first occurrence of a multibyte character that is not also in the multibyte string `mbset`. It is the multibyte replacement for the ANSI function `strspn`.

Return Value

A pointer to the first occurrence of a character not in `mbset` that is in `mbs`; `NULL` if all characters in `mbs` are in `mbset`.

See Also

`xvt_str_find_char_set`
`xvt_str_find_last_char`

Example

```
char *eon;
/* Find the end of the number */
eon = xvt_str_find_not_char_set(input_line,
"0123456789");
```

xvt_str_find_substring

Find Substring

Summary

```
char *xvt_str_find_substring(const char *mbs1,
     const char *mbs2)
```

`const char *mbs1`

String to search.

`const char *mbs2`

Substring for which to search.

Description

This function searches the multibyte string `mbs1` for the first occurrence of the multibyte string `mbs2`. It is the multibyte replacement for the ANSI function `strstr`.

Return Value

A pointer to the first occurrence of `mbs2` in `mbs1`; `NULL` if `mbs2` is not found in `mbs1`.

See Also

`xvt_str_compare_n_char`
`xvt_str_find_first_char`

xvt_str_find_token

Separate Multibyte String into Tokens

Summary

```
char *xvt_str_find_token(const char *mbs,
     const char *delimiter_set, size_t *n)
```

`const char *mbs`

String of tokens.

`const char *delimiter_set`

Set of delimiter characters.

`size_t *n`

Number of bytes in token.

Description

This function breaks the multibyte string `mbs` into tokens that are separated by one or more characters from the string of delimiters `delimiter_set`. It returns a pointer to the next token following a delimiter and returns the number of bytes in the token before the next delimiter or end of string in the parameter `n`.

To find all the tokens in a string, you must call `xvt_str_find_token` in a loop, once for each token. On each loop, pass in the value returned, which is the value of the current token, incremented by the byte count of the token. You can use `xvt_str_copy_n_size` to save the token. If no token is found, `NULL` is returned and `n` is set to zero.

This function is the multibyte replacement for the ANSI function `strtok`. Note some differences, however. The function does not cache the string on the first call and does not insert `NULL` characters into the string. It leaves the value of `mbs` unchanged. Therefore, when using `xvt_str_copy_n_size` to copy the token, make sure to properly `NULL`-terminate the copied string afterward.

Return Value

Pointer into `mbs` of the next token; `NULL` if no token is found. Sets `n` to the number of bytes in the token.

See Also

`xvt_str_copy_n_size`
`xvt_str_get_n_char_count`

Example

```
#define MAX_TOKEN_LEN 6  main()
{
     char *delims = " ,.";
    char *string = "Gee, what a great API."
    char *token = string;
    size_t nbytes; char tokenbuf[MAX_TOKEN_LEN];while
((token = xvt_str_find_token(token, delims,
         &nbytes)) != NULL)
     {
         /* Copy token but don't exceed buffer size */
xvt_str_copy_n_size(tokenbuf, token,
mi(nbytes,MAX_TOKEN_LEN-1)); /* Null terminate token for
as much as we could copy */ tokenbuf[min(nbytes,
MAX_TOKEN_LEN-1)] = '0'; printf ("%*s
", token); token += nbytes;
     }
 }
```

The program prints out the following tokens:

```
Gee
what
a
great
API
```

xvt_str_get_byte_count

Count Bytes in Multibyte String

Summary

```
size_t xvt_str_get_byte_count(const char *mbs)
```

`const char *mbs`

Multibyte string.

Description

This function counts the number of bytes in the multibyte string `mbs` up to the terminating `NULL`. This function is the multibyte equivalent of `strlen`.

See Also

`xvt_str_get_char_count`
`xvt_str_get_n_char_size`

xvt_str_get_char_count

Count Characters in Multibyte String

Summary

```
size_t xvt_str_get_char_count(const char *mbs

)const char *mbs
```

Multibyte string.

Description

This function counts the number of characters in the multibyte string `mbs`. This function differs from the `strlen` function, which returns the number of bytes in a `NULL`-terminated multibyte string.

Return Value

The number of characters in the multibyte string `mbs`.

See Also

```
xvt_str_get_byte_count
xvt_str_get_n_char_count
```

xvt_str_get_char_size

Count Number of Bytes in Multibyte Character

Summary

```
int xvt_str_get_char_size(const char * mbs)

const char * mbs
```

Multibyte character.

Description

This function counts the number of bytes in the multibyte character pointed to by `mbs`. It replaces the ANSI function `mblen`.

Return Value

The number of bytes in the multibyte character pointed to by `mbs`. If `*mbs` is the `NULL` character '`<fc0`', 0 is returned. If an error occurs, -1 is returned.

See Also

```
xvt_str_convert_mb_to_wc
xvt_str_get_byte_count
```

Example

```
char* s;/* skip first character */
s = str + xvt_str_get_char_size(str);
```

xvt_str_get_n_char_count

Count Characters in n Bytes of Multibyte String

Summary

```
size_t xvt_str_get_n_char_count(const char *mbs,
     const size_t n, size_t *used)
```

`const char *mbs`

Multibyte string.

`const size_t n`

Number of bytes to process.

`size_t *used`

Number of bytes processed.

Description

This function returns the number of complete characters in the first `n` bytes of the multibyte string `mbs`. If `mbs` is `NULL`, this function returns zero, otherwise the number of characters is returned. The number of bytes processed is returned in `used`. If the `n`th byte is not the end of a complete character, then the character defined by that byte is not included in the count. If a `NULL` character is encountered, the number of characters up to the `NULL` is returned but the `NULL` character is not included in the count. `NULL` can be passed as the argument used if this information is not desired.

Return Value

The number of complete multibyte characters.

See Also

```
xvt_str_get_char_count
xvt_str_get_n_char_size
```

Example

```
int nbytes, nchars;
char* token;if ((token = xvt_str_find_token(str, delims,
&nbytes))
     ! = NULL) {
     nchars = xvt_str_get_n_char_count(token,
         nbytes, NULL);
     if (0 == xvt_str_compare_n_char(token,
         last_keyword, nchars)) {
        ...
     }
 }
```

xvt_str_get_n_char_size

Counts Bytes in n Characters of Multibyte String

Summary

```
size_t xvt_str_get_n_char_size(const char *mbs,
     const size_t n)
```

`const char *mbs`

Multibyte string.

`const size_t n`

Number of characters.

Description

This function counts the number of bytes in the first `n` characters of the multibyte string `mbs`.

Return Value

Number of bytes; zero if `s` is `NULL`.

See Also

`xvt_str_get_char_count`
`xvt_str_get_n_char_count`

xvt_str_get_next_char

Get Next Character in a Multibyte String

Summary

```
char * xvt_str_get_next_char(const char * mbs)
```

`const char * mbs`

Pointer to multibyte string.

Description

This function returns a pointer to the next character in the multibyte string `mbs` following the character currently pointed to by `mbs`. The function assumes `mbs` points to the beginning of a multibyte character.

Return Value

A pointer to the next character in the multibyte string `mbs` following the character currently pointed to by `mbs`; a pointer to the `NULL` character if the current character or next character is a `NULL` character; `NULL` if the next character is not a valid multibyte character.

See Also

`xvt_str_get_char_size`
`xvt_str_get_prev_char`

xvt_str_get_prev_char

Get Preceding Character in a Multibyte String

Summary

```
char * xvt_str_get_prev_char (const char *start,
     const char *mbs)
```

`const char *start`

Pointer to the start of multibyte string.

`const char *mbs`

Pointer into multibyte string.

Description

This function returns a pointer to the character preceding the character pointed to by `mbs` in the multibyte string `start`. This function assumes `start` points to the beginning of a multibyte string and `mbs` points to the beginning of a multibyte character in the string `start`. The efficiency of this function may be limited to codeset design and available functionality on some platforms. Use it sparingly.

Return Value

A pointer to the character preceding the character pointed to by `mbs` in the multibyte string `start`; `start` if `mbs` equals `start` or if the character pointer for the preceding character is less than `start`.

See Also

`xvt_str_get_next_char`

xvt_str_is_*

xvt_str_is_* Functions

```
xvt_str_is_alnum
xvt_str_is_alpha
xvt_str_is_digit
xvt_str_is_equal
xvt_str_is_invariant
xvt_str_is_lower
xvt_str_is_space
xvt_str_is_upper
xvt_str_is_xdigit
```

xvt_str_is_alnum

Check if Multibyte Character is Alphanumeric

Summary

```
BOOLEAN xvt_str_is_alnum(const char *mbs)
```

`const char *mbs`

Pointer to multibyte character.

Description

This function checks if the first character of the multibyte string `mbs` is an alphanumeric character (a-z, A-Z, 0-9). This check also includes alphabetical characters with diacritical marks (depending on the current locale). It is the multibyte replacement for the ANSI function `isalnum`.

Return Value

`TRUE` if the first character is alphanumeric; `FALSE` otherwise.

See Also

```
xvt_str_is_alpha
xvt_str_is_digit
xvt_str_is_equal
xvt_str_is_invariant
xvt_str_is_lower
xvt_str_is_space
xvt_str_is_upper
xvt_str_is_xdigit
```

xvt_str_is_alpha

Check if Multibyte Character is Alphabetic

Summary

```
BOOLEAN xvt_str_is_alpha(const char *mbs)

const char *mbs
```

Pointer to multibyte character.

Description

This function checks if the first character of the multibyte string `mbs` is an alphabetic character (a-z, A-Z). This check also includes alphabetical characters with diacritical marks (depending on the current locale). It is the multibyte replacement for the ANSI function `isalpha`.

Return Value

`TRUE` if the first character is alphabetic; `FALSE` otherwise.

See Also

```
xvt_str_is_alnum
xvt_str_is_digit
xvt_str_is_equal
xvt_str_is_invariant
xvt_str_is_lower
xvt_str_is_space
xvt_str_is_upper
xvt_str_is_xdigit
```

xvt_str_is_digit

Check if Multibyte Character is a Decimal

Summary

```
BOOLEAN xvt_str_is_digit(const char *mbs)
```

`const char *mbs`

Pointer to multibyte character.

Description

This function checks if the first character of the multibyte string `mbs` is a decimal character (0-9). It is the multibyte replacement for the ANSI function `isdigit`.

Return Value

`TRUE` if the first character is a decimal; `FALSE` otherwise.

See Also

```
xvt_str_is_alnum
xvt_str_is_alpha
xvt_str_is_equal
xvt_str_is_invariant
xvt_str_is_lower
xvt_str_is_space
xvt_str_is_upper
xvt_str_is_xdigit
```

xvt_str_is_equal

Check if Strings are Equal

Summary

```
BOOLEAN xvt_str_is_equal(const char *mbs1,
     const char *mbs2)
```

`const char *mbs1`

First string.

`const char *mbs2`

Second string.

Description

This function determines if multibyte strings `mbs1` and `mbs2` are equal (character for character) and of the same length.

Return Value

`TRUE` if the strings are equal and of the same length; `FALSE` if the strings are not equal.

See Also

`xvt_str_compare`

xvt_str_is_invariant

Check if Multibyte Character is Invariant

Summary

```
BOOLEAN xvt_str_is_invariant(const char *mbs)
```

`const char *mbs`

Pointer to a multibyte character.

Description

This function detects if the first character of the multibyte string `mbs` is in the ISO 646 invariant code set (space ! " % & ’ ( ) * + , - . / 0-9 : ; < = > ? _ A-Z a-z).

Return Value

`TRUE` if the character is in the ISO 646 invariant code set; `FALSE` otherwise.

See Also

```
xvt_str_is_alnum
xvt_str_is_alpha
xvt_str_is_digit
xvt_str_is_equal
xvt_str_is_lower
xvt_str_is_space
xvt_str_is_upper
xvt_str_is_xdigit
```

xvt_str_is_lower

Check if First Multibyte Character is Lowercase

Summary

```
BOOLEAN xvt_str_is_lower(const char *mbs)
```

`const char *mbs`

Pointer to multibyte character.

Description

This function determines if the first character of the multibyte string `mbs` is a lowercase character (a-z). This check also includes lowercase alphabetical characters with diacritical marks (depending on the current locale). It is the multibyte replacement for the ANSI function `islower`.

Return Value

`TRUE` if the first character is lowercase; `FALSE` otherwise.

See Also

```
xvt_str_is_alnum
xvt_str_is_alpha
xvt_str_is_digit
xvt_str_is_equal
xvt_str_is_invariant
xvt_str_is_space
xvt_str_is_upper
xvt_str_is_xdigit
```

xvt_str_is_space

Check if First Multibyte Character is a Space

Summary

```
BOOLEAN xvt_str_is_space(const char *mbs)

const char *mbs
```

Pointer to multibyte character.

Description

This function determines if the first character of the multibyte string `mbs` is a standard white-space character. White-space characters include: space (' '), form feed ('<fcf'), new-line ('<fcn'), carriage return ('<fcr'), horizontal tab ('<fct'), vertical tab ('v'). This function is the multibyte replacement for the ANSI function `isspace`.

Return Value

`TRUE` if the first character is a standard white-space character; `FALSE` otherwise.

See Also

```
xvt_str_is_alnum
xvt_str_is_alpha
xvt_str_is_digit
xvt_str_is_equal
xvt_str_is_invariant
xvt_str_is_lower
xvt_str_is_upper
xvt_str_is_xdigit
```

xvt_str_is_upper

Check if First Multibyte Character is Uppercase Alphabetic

Summary

```
BOOLEAN xvt_str_is_upper(const char *mbs)

const char *mbs
```

Pointer to multibyte character.

Description

This function determines if the first character of the multibyte string `mbs` is uppercase alphabetic (A-Z). This check also includes uppercase alphabetical characters with diacritical marks (depending on the current locale). It is the multibyte replacement for the ANSI function `isupper`.

Return Value

`TRUE` if the first character is uppercase alphabetic; `FALSE` otherwise.

See Also

```
xvt_str_is_alnum
xvt_str_is_alpha
xvt_str_is_digit
xvt_str_is_equal
xvt_str_is_invariant
xvt_str_is_lower
xvt_str_is_space
xvt_str_is_xdigit
```

xvt_str_is_xdigit

Check if First String Character is a Hexadecimal Digit

Summary

```
BOOLEAN xvt_str_is_xdigit(const char *mbs)

const char *mbs
```

Pointer to multibyte character.

Description

This function determines if the first character of the multibyte string `mbs` is a hexadecimal digit (0-9, A-F, a-f). It is the multibyte replacement for the ANSI function `isxdigit`.

Return Value

`TRUE` if the first character is a hexadecimal digit; `FALSE` otherwise.

See Also

xvt_str_is_alnum
xvt_str_is_alpha
xvt_str_is_digit
xvt_str_is_equal
xvt_str_is_invariant
xvt_str_is_lower
xvt_str_is_space
xvt_str_is_upper

xvt_str_match

Match Multibyte Pattern Against String

Summary

```
BOOLEAN xvt_str_match(const char *mbs,
     const char *pat, BOOLEAN case_sensitive)
```

`const char *mbs`

Multibyte string to be scanned.

`const char *pat`

String with pattern.

`BOOLEAN case_sensitive`

Determines whether matching is case-sensitive.

Description

This function is used to compare a string to a pattern. It returns `TRUE` if the string `mbs` matches the pattern `pat` (possibly containing the wildcard characters "`*`" and "`?`"), and `FALSE` otherwise. Either `mbs` and `pat` (or both) can be `NULL`. If they are not `NULL`, they must be `NULL`-terminated strings.

Within the pattern `pat`, the wildcard character "`*`" matches any sequence of zero or more characters. The wildcard character "`?`" matches any single character. Other characters in the pattern match only themselves, unless `case_sensitive` is `FALSE`, which causes letters to match either upper- or lowercase versions of themselves in `mbs`.

The entire pattern must match the entire string `mbs` for the match to succeed. Use the wildcard "`*`" at the beginning or end of the pattern if the match can occur anywhere in the string.

The matching algorithm used by `xvt_str_match` is identical to that used by `xvt_fsys_list_files`.

Return Value

`TRUE` if the string matches the pattern, or if both `mbs` and `pat` are `NULL`; `FALSE` if there is no match, or if either `mbs` or `pat` (but not both) are `NULL`.

Implementation Note

The matching algorithm is not the same as that used by NTFS to match filenames; `xvt_str_match` doesn't treat the "extension" in a special way, and it allows multiple "`*`" wildcards in the pattern. The matching algorithm is the same as that typically used by UNIX shells.

See Also

`xvt_fsys_list_files`

Example

Given this initialization:

```
char *s = "Virtual";
```

The following table shows the results from several calls to `xvt_str_match`:

Expression	Result
`xvt_str_match(s, "Virtual", FALSE)`	TRUE
`xvt_str_match(s, "virtual", FALSE)`	TRUE
`xvt_str_match(s, "Virtual", TRUE)`	TRUE
`xvt_str_match(s, "virtual", TRUE)`	FALSE
`xvt_str_match(s, "V*rtu*", FALSE)`	TRUE
`xvt_str_match(s, "?ir?ual", FALSE)`	TRUE
`xvt_str_match(s, "V*V", FALSE)`	FALSE
`xvt_str_match(s, "*", FALSE)`	TRUE
`xvt_str_match(s, "?", FALSE)`	FALSE
`xvt_str_match(NULL, NULL, TRUE)`	TRUE
`xvt_str_match(s, NULL, TRUE)`	FALSE
`xvt_str_match(NULL, "VIRTUAL", TRUE)`	FALSE

xvt_str_parse_double

Convert Multibyte String to Double-Precision Floating Point Value

Summary

```
double xvt_str_parse_double(const char *mbs,
     char **mbs_end)
```

`const char *mbs`

String to convert.

`char **mbs_end`

Pointer to a character pointer.

Description

This function converts the multibyte string `mbs` to a double-precision floating point value. It skips over any white-space characters at the beginning of `mbs`. It stops converting when it reaches a character that can't be part of a number (including multibyte characters). It looks for a number formatted like `[+|-][digits][.digits][d|D|e|E[+|-]digits]`. This function is the multibyte replacement for the ANSI function `strtod`.

Return Value

Double-precision floating point value; sets `*mbs_end` to the first non-number character if `mbs_end` is not `NULL`; zero if no number is found (`mbs_end` is set to `mbs`). If the number exceeds the double range, a warning is signaled and `HUGE_VAL` with the same sign as the number is returned.

See Also

`xvt_str_parse_long`
`xvt_str_parse_ulong`

xvt_str_parse_long

Convert a Multibyte String to a Long Integer Value

Summary

```
long xvt_str_parse_long(const char *mbs,
     char **mbs_end, short base)
```

`const char *mbs`

Strings to convert.

`char **mbs_end`

Pointer to a character pointer.

`short base`

Base to use for the conversion.

Description

This function converts the multibyte string `mbs` to a long integer value in the indicated numeric base. It skips over any white-space characters at the beginning of `mbs`. It stops converting when it reaches a character that can't be part of a number (this includes multibyte characters). It looks for a number formatted like `[+|-][0x|0X|0][digits]`.

The value for base must be zero or between 2 and 16. If base is between 2 and 16, the number is converted to a number in the specified base. If base is zero, then the number is converted using a numeric base that is determined by the prefix (see below).

Prefix	Base Is
0x or 0X	16
0	8
none	10

The numbers must contain numerals and letters that are valid for the base. For example, if base is 8, the number can contain only the numerals 0-7. If base is 16, the number can contain only the numerals 0-9 and the letters A-F and a-f.

This function is the multibyte replacement for the ANSI function `strtol`.

Return Value

`long` integer value; sets `*mbs_end` to the first non-number character if `mbs_end` is not `NULL`; zero if no number is found (and `mbs_end` is set to `mbs`). If the number exceeds the long range, a warning is signaled and `LONG_MAX` or `LONG_MIN` is returned.

See Also

`xvt_str_parse_double`
`xvt_str_parse_ulong`

xvt_str_parse_ulong

Convert Multibyte String to an Unsigned Long Integer Value

Summary

```
unsigned long xvt_str_parse_ulong(const char *mbs,
     char **mbs_end, short base)
```

`const char *mbs`

String to convert.

`char **mbs_end`

Pointer to a character pointer.

`short base`

Base to use for the conversion.

Description

This function converts the multibyte string `mbs` to an `unsigned long` integer value in the indicated numeric base and returns it. It skips over any white-space characters at the beginning of `mbs`. It stops converting when it reaches a character that can't be part of a number (this includes multibyte characters). It looks for a number formatted like `[+][0x|0X|0][digits]`.

The value for base must be zero or between 2 and 16. If base is between 2 and 16, the number is converted to a number in the specified base. If base is zero, then the number is converted using a numeric base that is determined by the prefix (see below).

Prefix	Base Is
0x or 0X	16

0	8
none	10

The numbers must contain numerals and letters that are valid for the base. For example, if base is 8, the number can contain only the numerals 0-7. If base is 16, the number can contain only the numerals 0-9 and the letters A-F and a-f.

This function is the multibyte replacement for the ANSI function `strtoul`.

Return Value

`unsigned` long integer value; sets `*mbs_end` to first non-number character if `mbs_end` is not `NULL`; zero if no number is found (`mbs_end` is set to `mbs`). If the number exceeds the `unsigned long` range, a warning is signaled and `ULONG_MAX` is returned.

See Also

`xvt_str_parse_double`
`xvt_str_parse_long`

xvt_str_sprintf and xvt_str_vsprinf

Process Formats

Summary

```
size_t xvt_str_sprintf(char *mbs, const char *format,
     ...)
 size_t xvt_str_vsprintf(char *mbs, const char *format,
     va_list varg)
```

`char *mbs`

Target string.

`const char *format`

Format string.

`va_list varg`

Argument list.

Description

These functions process formats according to the ANSI C specification for `sprintf` and `vsprintf`. They also process formats

according to the compiler ANSI C Library (as available) and specifying the argument order.

The format specifiers are introduced by the `%` character or by the sequence `%digit$` followed by the normal specifiers. The conversion can be applied to the nth argument in the argument list, rather than the next unused argument by using the `%digit$` sequence. `digit` is a decimal integer in the range 1-9 (inclusive) and gives the position of the argument in the argument list. All format specifiers for a given format string must follow this convention or none of them can. The application cannot mix conventions. Also the caller cannot reference an argument more than once nor can it skip arguments.

This feature provides for the definition of format strings that select arguments in an order appropriate to the specific locale and language. This syntax generally follows that described in the Sun Release 4.1 C Library Functions description of `printf`.

Return Value

The number of bytes processed.

See Also

`printf`

Example

```
char format_us[ ] = "%d/%od/%d";
char format_eu[ ] = "%2$d/%1$d/%3$d";
char* format;
if (locale == US_LOCALE)
     format = format_us;
 else
     format = format_eu;
xvt_str_sprintf(outstr, format, month, day, year);
xvt_dwin_draw_text(x, y, outstr, -1);
```

xvt_str_translate_codeset

Translates a Text String According to the XVT_CODESET_MAP

Summary

```
long xvt_str_translate_codeset(XVT_CODESET_MAP
codeset_map, char *string,
char *strbuf, size_t bufsize)
```

`XVT_CODESET_MAP codeset_map`

Codeset map used in codeset translation.

`char *string`

Text string to translate.

`char *strbuf`

Output buffer for translated string.

`size_t bufsize`

Size in bytes of output buffer.

Description

This function traverses the input string and converts the characters from one character codeset to another according to the `XVT_CODESET_MAP`. The input string is assumed to be encoded in the "from codeset" of the `XVT_CODESET_MAP`. The "to codeset" should support the same languages and contain the same characters as the input character codeset. Any character found in the input string for which there is no mapping value in the "to codeset" is discarded. The resulting string is returned in `strbuf` up to a length of `bufsize` bytes.

Return Value

The number of characters processed is returned if successful; -1 if an error occurred.

Parameter and Validity Conditions

XVT issues an error if any parameter is `NULL` or `codeset_map` is not a valid `XVT_CODESET_MAP`.

See Also

`XVT_CODESET_MAP`
`xvt_str_create_codeset_map`
`xvt_str_destroy_codeset_map`

xvt_timer_*

Timer Objects

```
xvt_timer_create
xvt_timer_destroy
```

xvt_timer_create

Start Generation of Timer Events

Summary

```
long xvt_timer_create(WINDOW win, long interval)
```

`WINDOW win`

Window whose timer events are being started. `win` must be a window of type `W_DOC`, `W_PLAIN`, `W_DBL`, `W_TASK`, `W_NO_BORDER`, `W_MODAL`, `WD_MODAL`, or `WD_MODELESS`.

`long interval`

Interval of `E_TIMER` events in milliseconds.

Description

This function starts a timer that sends an `E_TIMER` event to the event handler for `win`. The event is sent at an interval of milliseconds as indicated by `interval`. An ID uniquely identifying the timer is returned.

Timer events are useful for setting connection or login timeouts, as well as for crude animation or "slide-show" displays. In addition, timer events can be used to implement a crude sort of multi-threading, where a window in which a continuous operation is to be performed is set up to receive continuous timer events, and performs a bit of the operation with each timer tick. For multi-threading ideas, see `xvt_app_process_pending_events`.

Return Value

The ID of the timer assigned to the window. If the return value is `XVT_TIMER_ERROR`, then no timer was available to be started.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are not met:

- `win` must be a valid XVT `WINDOW` and must not be a control or print window.
- `interval` must be positive.

Implementation Note

The number of timers that can be created for a window or dialog and the total number of timers that can be created are platform-specific.

See Also

`E_TIMER`
`xvt_app_process_pending_events`
`xvt_timer_destroy`

The "Events" chapter in the *XVT Portability Toolkit Guide*

xvt_timer_destroy

Turn Off Timer

Summary

```
void xvt_timer_destroy(long id)
```

`long id`

Timer ID returned by `xvt_timer_create`.

Description

This function turns off the timer identified by `id`, and makes it available for reuse. Turning off the timer causes XVT to stop sending `E_TIMER` events to an event handler as a result of that particular timer. Since timers tie up resources, using the `xvt_timer_create`/`xvt_timer_destroy` method of eliminating timer events is preferred to the technique of masking the events to prevent the event handler from receiving the events.

On some platforms, `E_TIMER` events can remain in the queue even after `xvt_timer_destroy` has been called. Therefore, to be completely safe, your application should call `xvt_win_set_event_mask` before calling `xvt_timer_destroy` to mask `E_TIMER` events. This guarantees that any remaining `E_TIMER` events in the event queue for a particular window are ignored.

Parameter Validity and Error Conditions

If `id` is not the ID of a valid timer returned from `xvt_timer_create`, XVT issues an error.

See Also

```
E_TIMER
xvt_timer_create
```

Example

```
WINDOW window;
long timer_id;
...
/* turn off timer */
xvt_timer_destroy(timer_id);
/* mask any pending E_TIMER events */
xvt_win_set_event_mask(window, ~EM_TIMER &
xvt_win_get_event_mask(window));
```

xvt_treeview_add_child_node

Add a child treeview node to existing treeview node

Summary

```
BOOLEAN xvt_treeview_add_child_node(
        XVT_TREEVIEW_NODE parent_node,
        XVT_TREEVIEW_NODE child_node);
```

`XVT_TREEVIEW_NODE parent_node`

Defines the parent node. Must be of type `XVT_TREEVIEW_NODE_NONTERMINAL`.

`XVT_TREEVIEW_NODE child_node`

Defines the child node. Can be of type `XVT_TREEVIEW_NODE_TERMINAL` or `XVT_TREEVIEW_NODE_NONTERMINAL`.

Description

This function adds the treeview child_node to the existing treeview parent_node. The view of treeview control is not updated by this call.

Return Value

TRUE is successful; FALSE otherwise.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·parent_node and child_node must not be null

·parent_node must of type XVT_TREEVIEW_NODE_NONTERMINAL

See Also

XVT_TREEVIEW_NODE

XVT_TREEVIEW_NODE_* Values for XVT_TREEVIEW_NODE_TYPE

xvt_treeview_get_child_node

xvt_treeview_create_node

xvt_treeview_get_parent_node

xvt_treeview_get_root_node

xvt_treeview_remove_child_node

xvt_treeview_update

xvt_treeview_collapse_node

Collapses node

Summary

```
void xvt_treeview_collapse_node(XVT_TREEVIEW_NODE node,
        BOOLEAN  recurse);
```

XVT_TREEVIEW_NODE node

The node to collapse.

BOOLEAN recurse

Flag stating whether to recursively collapse all child nodes.

Description

This function will collapse the node if expanded. The expansion state of the node's children are not changed unless recurse is TRUE, then all child nodes are also collapsed. The view of treeview control is not updated by this call.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

See Also

`XVT_TREEVIEW_NODE`

`xvt_treeview_expand_node`

`xvt_treeview_update`

xvt_treeview_create

Creates treeview control

Summary

```
WINDOW xvt_treeview_create(WINDOW parent_win,
    RCT *  rct_p, char * title, long ctl_flags,
    long data, int ctl_id, XVT_IMAGE item_image,
    XVT_IMAGE collapsed_image, XVT_IMAGE expanded_image,
    long attrs, int line_height);
```

`WINDOW parent_win`

Window in which the control should be placed. It must be a valid window of type W_*.

`RCT *  rct_p`

Defines the bounding rectangle for the control in terms of the parent window's client area. This parameter must not be NULL and must point to a valid rectangle.

`char * title`

Used to set the text of the control. It has no effect for this control. If it is NULL, the control will have no title.

`long ctl_flags`

Controls the attributes and the initial state of a control. The applicable control flags vary among controls, and a complete table listing the valid control flags for each control can be found in Window/Dialog/Control Creation Function Parameters.

`long data`

Contains any application data that you wish to attach to a control. Typically, this will be a pointer to some structure allocated from the heap, cast into a long such that later your application can retrieve the structure and look at it.

`int ctl_id`

An ID number for the control relative to its parent window. When XVT sends an E_CONTROL event to the event handler for the window containing the control, it sets the v.ctl.id field of the EVENT structure to the ID of the control that was activated. A control ID-parent window combination is a way of uniquely identifying a control independent of its window handle. Keep in mind that it is not necessary to use control IDs, but if you choose to use them, then all of the IDs for the controls in a window must be unique. You can also call the xvt_win_get_ctl function with an ID and parent window to retrieve the WINDOW for a control.

`XVT_IMAGE item_image`

The image for an item within the node. If the image is NULL_IMAGE then the item will have no image. This image will be used for all items within the treeview unless it is overridden with a item image at the node level.

`XVT_IMAGE collapsed_image`

The image for a node in the collapsed state. If the image is NULL_IMAGE then the node will have no image. This image will be used for all collapsed nodes within the treeview unless it is overridden with a collapsed node image at the node level.

`XVT_IMAGE expanded_image`

The image for a node in the expanded state. If the image is NULL_IMAGE then the node will have no image. This image will be used for all expanded nodes within the treeview unless it is overridden with an expanded node image at the node level.

`long attrs`

Controls the additional attributes, those outside of ctl_flags, for the control. These attributes can be ORed together. See "Treeview Attribute Constants" for complete list.

`int line_height`

The height of the line for a node. If set to 0, the control will calculate the line height based on the control font size.

Description

This function creates the treeview control and adds it to parent_win. This function does not add the control to dialogs.

Return Value

A WINDOW if successful; NULL_WIN if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·parent_win must be a valid XVT window of type W_*. You cannot create a control in a dialog, a print window, a task window, or a screen window. (The exception to this is if your application is running with XVT/Win32, and has set the nonportable attribute ATTR_WIN_PM_DRAWABLE_TWIN. In that case, it can create controls in the task window.)

·rct_p must point to a valid rectangle.

·ctl_flags must be appropriate for the control you want to create, as defined in Window/Dialog/Control Creation Function Parameters.

·attrs must be one or more of the defined treeview attribute constants

See Also

`CTL_FLAG_* Options`

E_CONTROL

EVENT

RCT

WINDOW

WIN_TYPE

xvt_treeview_get_attributes

xvt_treeview_get_line_height

xvt_treeview_get_root_node

xvt_treeview_resume

xvt_treeview_set_attributes

xvt_treeview_set_line_height

xvt_treeview_suspend

xvt_treeview_update

xvt_vobj_get_data

xvt_vobj_get_title

xvt_vobj_set_data

xvt_vobj_set_title

xvt_win_get_ctl

Window/Dialog/Control Creation Function Parameters

The "Controls" chapter in the “XVT Portability Toolkit Guide”

Treeview Attribute Constants

xvt_treeview_create_node

Creates a treeview node

Summary

```
XVT_TREEVIEW_NODE
xvt_treeview_create_node(XVT_TREEVIEW_NODE_TYPE type,
    XVT_IMAGE  item_image, XVT_IMAGE  collapsed_image,
    XVT_IMAGE  expanded_image, char *string,
    XVT_TREEVIEW_CALLBACK  callback, long data);
```

XVT_TREEVIEW_NODE_TYPE type

The treeview node can be of type XVT_TREEVIEW_NODE_TERMINAL,a leaf with no children, or XVT_TREEVIEW_NODE_NONTERMINAL, a branch which may have have children.

`XVT_IMAGE  item_image`

The image for an item within the node. If the image is NULL_IMAGE then the item will use the treeview control's item image.

`XVT_IMAGE  collapsed_image`

The image for a node in the collapsed state. If the image is NULL_IMAGE then the node will use the treeview control's collapsed image.

`XVT_IMAGE  expanded_image`

The image for a node in the expandeded state. If the image is NULL_IMAGE then the node will use the treeview control's expanded image.

`char *string`

Used to set the title of the node. If it is NULL, the node will have no title.

`XVT_TREEVIEW_CALLBACK  callback`

callback is a pointer to a function that will be executed in response to a double-click on any node or the "Enter" key on a selected node. The node will continue to perform the standard behavior if the callback function returns TRUE. Set callback to NULL for no call back.

`long data`

Contains any application data that you wish to attach to a control. Typically, this will be a pointer to some structure allocated from the heap, cast into a long such that later your application can retrieve the structure and look at it.

Description

This function creates a treeview node that can later be assigned to the treeview control's root node or to another node as a child node.

Return Value

A XVT_TREEVIEW_NODE if successful; NULL if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·type is not of XVT_TREEVIEW_NODE_TYPE

See Also

`XVT_TREEVIEW_NODE`

`XVT_TREEVIEW_NODE_* Values for XVT_TREEVIEW_NODE_TYPE`

`xvt_treeview_add_child_node`

`xvt_treeview_create`

`xvt_treeview_destroy_node`

`xvt_treeview_get_node_callback`

`xvt_treeview_get_node_data`

`xvt_treeview_get_node_image_collapsed`

`xvt_treeview_get_node_image_expanded`

`xvt_treeview_get_node_image_item`

`xvt_treeview_get_node_string`

`xvt_treeview_get_node_type`

`xvt_treeview_set_node_callback`

`xvt_treeview_set_node_data`

`xvt_treeview_set_node_image_collapsed`

`xvt_treeview_set_node_image_expanded`

`xvt_treeview_set_node_string`

`xvt_treeview_set_node_type`

xvt_treeview_destroy_node

Destroys treeview node

Summary

```
void xvt_treeview_destroy_node(XVT_TREEVIEW_NODE node);
```

`XVT_TREEVIEW_NODE node`

A valid treeview node.

Description

This function will destroy the treeview node. If the node has a parent, the node will be removed from the parent. If the node is of type XVT_TREEVIEW_NODE_NONTERMINAL then all child nodes will also be destroyed. The view of treeview control is not updated by this call.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

See Also

`XVT_TREEVIEW_NODE_* Values for XVT_TREEVIEW_NODE_TYPE`

`xvt_treeview_update`

`xvt_vobj_destroy`

xvt_treeview_expand_node

Expands node

Summary

```
void xvt_treeview_expand_node(XVT_TREEVIEW_NODE node,
    BOOLEAN  recurse);
```

`XVT_TREEVIEW_NODE node`

A valid treeview node.

`BOOLEAN  recurse`

Flag stating whether to recursively expand all child nodes.

Description

This function will collapse the node if expanded. The expansion state of the node's children are not changed unless recurse is TRUE, then all child nodes are also collapsed. The view of treeview control is not updated by this call.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

See Also

`XVT_TREEVIEW_NODE`

`xvt_treeview_collapse_node`

`xvt_treeview_update`

xvt_treeview_get_attributes

Get the attributes for the treeview control

Summary

```
long xvt_treeview_get_attributes(WINDOW ctl_win);
```

`WINDOW ctl_win`

Window of control of type WC_TREEVIEW.

Description

This function returns the current attributes for the treeview control.

Return Value

ORed list of treeview attributes.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·ctl_win must not be null

·ctl_win must of type WC_TREEVIEW

See Also

WIN_TYPE

Treeview Attribute Constants

xvt_treeview_create

xvt_treeview_set_attributes

xvt_treeview_get_child_node

Get a child node from a parent node

Summary

```
XVT_TREEVIEW_NODE xvt_treeview_get_child_node(
    XVT_TREEVIEW_NODE parent_node, int position);
```

`XVT_TREEVIEW_NODE parent_node`

Defines the parent node. Must be of type XVT_TREEVIEW_NODE_NONTERMINAL.

`int position`

The position of the child node, 0 to the number of children minus 1, within the parent to retrieve.

Description

This function retrieves the nth child node of the parent.

Return Value

XVT_TREEVIEW_NODE of the nth child node of the parent if valid.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·parent_node must not be null

·parent_node must of type XVT_TREEVIEW_NODE_NONTERMINAL

See Also

`XVT_TREEVIEW_NODE`

`XVT_TREEVIEW_NODE_* Values for XVT_TREEVIEW_NODE_TYPE`

`xvt_treeview_add_child_node`

`xvt_treeview_get_parent_node`

`xvt_treeview_get_root_node`

`xvt_treeview_remove_child_node`

xvt_treeview_get_line_height

Get line height of treeview node

Summary

```
long xvt_treeview_get_line_height(WINDOW ctl_win);
```

`WINDOW ctl_win`

Window of control of type WC_TREEVIEW.

Description

This function returns the line height of the treeview node. If treeview control was created with the line height set to 0 then this function returns the line as calculated from the node image heights.

Return Value

The line height of the treeview node.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·ctl_win must not be null

·ctl_win must of type WC_TREEVIEW

See Also

`WIN_TYPE`

`xvt_treeview_create`

`xvt_treeview_set_line_height`

xvt_treeview_get_node_callback

Get the node call back function

Summary

```
XVT_TREEVIEW_CALLBACK
xvt_treeview_get_node_callback( XVT_TREEVIEW_NODE node);
```

`XVT_TREEVIEW_NODE node`

The node.

Description

This function returns the call back function assigned to a node either at node creation or by using xvt_treeview_set_node_callback.

The callback is a pointer to a function that will be called in response to a double-click on any node or the “Enter” key on a selected node. The node will continue to perform the standard behavior if the callback function returns TRUE. Set callback to NULL for no call back.

Return Value

XVT_TREEVIEW_CALLBACK for treeview node if successful.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

See Also

`XVT_TREEVIEW_NODE`

`XVT_TREEVIEW_CALLBACK`

`xvt_treeview_create_node`

`xvt_treeview_set_node_callback`

xvt_treeview_get_node_data

Get the node data

Summary

```
long xvt_treeview_get_node_data(XVT_TREEVIEW_NODE node);
```

`XVT_TREEVIEW_NODE node`

The node.

Description

This function returns the data assigned to a node either at node creation or by using xvt_treeview_set_node_data.

Return Value

long value contain node data.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

See Also

`XVT_TREEVIEW_NODE`

`xvt_treeview_create_node`

`xvt_treeview_set_node_data`

xvt_treeview_get_node_image_collapsed

Get the collapsed image for a node

Summary

```
XVT_IMAGE xvt_treeview_get_node_image_collapsed(
        XVT_TREEVIEW_NODE node);
```

`XVT_TREEVIEW_NODE node`

The node.

Description

This function returns the collapsed image assigned to a node either at node creation or by using xvt_treeview_set_node_image_collapsed.

Return Value

XVT_IMAGE for the collapsed image of the node.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

See Also

`XVT_TREEVIEW_NODE`

`xvt_treeview_create_node`

xvt_treeview_set_node_image_collapsed

xvt_treeview_get_node_image_expanded

Get the expanded image for a node

Summary

```
XVT_IMAGE xvt_treeview_get_node_image_expanded(
        XVT_TREEVIEW_NODE node);
```

XVT_TREEVIEW_NODE node

The node.

Description

This function returns the expanded image assigned to a node either at node creation or by using xvt_treeview_set_node_image_expanded.

Return Value

XVT_IMAGE for the expanded image of the node.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

See Also

XVT_TREEVIEW_NODE

xvt_treeview_create_node

xvt_treeview_set_node_image_expanded

xvt_treeview_get_node_image_item

Get the item image for a node

Summary

```
XVT_IMAGE xvt_treeview_get_node_image_item(
        XVT_TREEVIEW_NODE node);
```

```
XVT_TREEVIEW_NODE node
```

The node.

Description

This function returns the item image assigned to a node either at node creation or by using xvt_treeview_set_node_image_item.

Return Value

XVT_IMAGE for the item image of the node.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

See Also

`XVT_TREEVIEW_NODE`

`xvt_treeview_create_node`

`xvt_treeview_set_node_image_item`

xvt_treeview_get_node_num_children

Get the number of child nodes for a node

Summary

```
int xvt_treeview_get_node_num_children(
        XVT_TREEVIEW_NODE node);
```

```
XVT_TREEVIEW_NODE node
```

The node.

Description

This function returns the number of child nodes in node. Nodes of type XVT_TREEVIEW_NODE_TERMINAL have 0 children.

Return Value

Number of child nodes.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

See Also

XVT_TREEVIEW_NODE

XVT_TREEVIEW_NODE_* Values for XVT_TREEVIEW_NODE_TYPE

xvt_treeview_add_child_node

xvt_treeview_get_node_num_vis_children

xvt_treeview_remove_child_node

xvt_treeview_get_node_num_vis_children

Get the number of visible child nodes for a node

Summary

```
int xvt_treeview_get_node_num_vis_children(
        XVT_TREEVIEW_NODE node);

XVT_TREEVIEW_NODE node
```

The node.

Description

This function returns the number of visible child nodes, through expansion, in node. Nodes of type XVT_TREEVIEW_NODE_TERMINAL have 0 children.

Return Value

Number of visible child nodes.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

See Also

XVT_TREEVIEW_NODE

XVT_TREEVIEW_NODE_* Values for XVT_TREEVIEW_NODE_TYPE

```
xvt_treeview_add_child_node
xvt_treeview_get_node_num_children
xvt_treeview_remove_child_node
```

xvt_treeview_get_node_string

Get the item text for a node

Summary

```
char * xvt_treeview_get_node_string(
       XVT_TREEVIEW_NODE node,
       char *string, int sz_string);
```

`XVT_TREEVIEW_NODE node`

The node.

`char *string`

Character buffer for item text.

`int sz_string`

Maximum character buffer capacity.

Description

This function copies the item text to the string buffer that was assigned to a node either at node creation or by using xvt_treeview_set_node_string. The maximum number of characters copied to the string buffer is determined by sz_string.

Return Value

Pointer to string buffer.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

·string must not be null

See Also

`XVT_TREEVIEW_NODE`

```
xvt_treeview_create_node
xvt_treeview_set_node_string
```

xvt_treeview_get_node_type

Get node type

Summary

```
XVT_TREEVIEW_NODE_TYPE
xvt_treeview_get_node_type(XVT_TREEVIEW_NODE node);

XVT_TREEVIEW_NODE node
```

The node.

Description

This function returns the node type assigned to a node either at node creation or by using xvt_treeview_set_node_type.

Return Value

XVT_TREEVIEW_NODE_TYPE of node.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

See Also

```
XVT_TREEVIEW_NODE
XVT_TREEVIEW_NODE_TYPE
xvt_treeview_create_node
xvt_treeview_set_node_type
```

xvt_treeview_get_parent_node

Get parent node

Summary

```
XVT_TREEVIEW_NODE xvt_treeview_get_parent_node(
```

```
        XVT_TREEVIEW_NODE child_node);
```

`XVT_TREEVIEW_NODE node`

The node.

Description

This function returns the parent node of a child node.

Return Value

XVT_TREEVIEW_NODE for the parent, if assigned. NULL if no parent.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

See Also

`XVT_TREEVIEW_NODE`

`xvt_treeview_add_child_node`

`xvt_treeview_destroy_node`

`xvt_treeview_remove_child_node`

---

xvt_treeview_get_root_node

Get root node form treeview control

Summary

```
XVT_TREEVIEW_NODE xvt_treeview_get_root_node(
        WINDOW ctl_win);
```

`WINDOW ctl_win`

Window of control of type WC_TREEVIEW.

Description

This functions returns the root node for the treeview control for which all other nodes of the treeview control are children. The root node may or may not be visible based on the TREEVIEW_SHOW_ROOT_NODE attribute.

Return Value

XVT_TREEVIEW_NODE containing the root node of the treeview control.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·ctl_win must not be null

·ctl_win must of type WC_TREEVIEW

See Also

WIN_TYPE

XVT_TREEVIEW_NODE

Treeview Attribute Constants

xvt_treeview_add_child_node

xvt_treeview_create

xvt_treeview_create_node

xvt_treeview_set_attributes

xvt_treeview_node_selected

Get node selection state

Summary

```
BOOLEAN xvt_treeview_node_selected(
        XVT_TREEVIEW_NODE node);

XVT_TREEVIEW_NODE node
```

The node.

Description

This function returns the selection state of a node. The selection state is dependant on the following attributes: TREEVIEW_SELECT_NONE, TREEVIEW_SELECT_ONE, or TREEVIEW_SELECT_MANY. The selection is set via user interaction.

Return Value

TRUE is node is selected, FALSE otherwise.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

See Also

`XVT_TREEVIEW_NODE`

`xvt_treeview_get_node_callback`

`xvt_treeview_set_node_callback`

xvt_treeview_remove_child_node

Remove child node from list

Summary

```
BOOLEAN xvt_treeview_remove_child_node(
        XVT_TREEVIEW_NODE child_node);

XVT_TREEVIEW_NODE node
```

The node.

Description

This function removes the child node from its parent. The node is note destroyed.

Return Value

TRUE is node is successful, FALSE otherwise.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·child node must not be null

See Also

`XVT_TREEVIEW_NODE`

`xvt_treeview_add_child_node`

`xvt_treeview_create_node`

```
xvt_treeview_destroy_node
xvt_treeview_get_parent_node
```

xvt_treeview_resume

Resume updating of treeview control

Summary

```
void xvt_treeview_resume(WINDOW ctl_win);

WINDOW ctl_win
```

Window of control of type WC_TREEVIEW.

Description

This function resumes the updating of the treeview control and will update the control's view if needed.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·ctl_win must not be null

·ctl_win must of type WC_TREEVIEW

See Also

```
WIN_TYPE
xvt_treeview_suspend
xvt_treeview_update
```

xvt_treeview_set_attributes

Set the attributes for treeview control

Summary

```
void xvt_treeview_set_attributes(WINDOW ctl_win,
        long attrs);

WINDOW ctl_win
```

Window of control of type WC_TREEVIEW.

`long attrs`

Treeview Attribute Constants

Description

This function sets the current attributes, those outside of ctl_flags, for treeview control. These attributes can be ORed together. See “Treeview Attribute Constants” for complete list.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·ctl_win must not be null

·ctl_win must of type WC_TREEVIEW

·attrs must be one or more of the defined treeview attribute constants

See Also

`WIN_TYPE`

`Treeview Attribute Constants`

`xvt_treeview_create`

`xvt_treeview_get_attributes`

`xvt_treeview_update`

xvt_treeview_set_line_height

Set line height for a node

Summary

```
void xvt_treeview_set_line_height(
        WINDOW ctl_win, long line_height);
```

`WINDOW ctl_win`

Window of control of type WC_TREEVIEW.

`long line_height`

New line height.

Description

This function sets the height of the line for a node. If set to 0, the control will calculate the line height based on the control font size.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·ctl_win must not be null

·ctl_win must of type WC_TREEVIEW

See Also

`WIN_TYPE`

`xvt_treeview_create`

`xvt_treeview_set_line_height`

---

xvt_treeview_set_node_callback

Set the node call back function

Summary

```
void xvt_treeview_set_node_callback(
        XVT_TREEVIEW_NODE  node,
        XVT_TREEVIEW_CALLBACK fcn);
```

`XVT_TREEVIEW_NODE  node`

The node.

`XVT_TREEVIEW_CALLBACK fcn`

The callback function expects to be passed the following parameters: (WINDOW ctl_win, XVT_TREEVIEW REEVIEW_NODE node) and returns a BOOLEAN. Set callback to NULL for no call back.

Description

This function sets the call back function for a node.

The callback is a pointer to a function that will be called in response to a double-click on any node or the “Enter” key on a selected node.

The node will continue to perform the standard behavior if the callback function returns TRUE. Set callback to NULL for no call back.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

See Also

`XVT_TREEVIEW_NODE`

`XVT_TREEVIEW_CALLBACK`

`xvt_treeview_create_node`

`xvt_treeview_get_node_callback`

xvt_treeview_set_node_data

Set the node data

Summary

```
void  xvt_treeview_set_node_data (
       XVT_TREEVIEW_NODE node, long  data);
```

`XVT_TREEVIEW_NODE  node`

The node.

`long  data`

Node data.

Description

This function sets the data assigned to a node.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

See Also

`XVT_TREEVIEW_NODE`

`xvt_treeview_create_node`

xvt_treeview_get_node_data

xvt_treeview_set_node_image_collapsed

Set the collapsed image for a node

Summary

```
void xvt_treeview_set_node_image_collapsed(
        XVT_TREEVIEW_NODE node, XVT_IMAGE image);
```

`XVT_TREEVIEW_NODE  node`

The node.

`XVT_IMAGE image`

The image for a node in the collapsed state. If the image is NULL_IMAGE then the node will use the treeview control's collapsed image.

Description

This function sets the collapsed image for a node.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

See Also

`XVT_TREEVIEW_NODE`

`xvt_treeview_create_node`

`xvt_treeview_get_node_image_collapsed`

xvt_treeview_set_node_image_expanded

Set the expanded image for a node

Summary

```
void xvt_treeview_set_node_image_expanded(
        XVT_TREEVIEW_NODE node, XVT_IMAGE image);
```

`XVT_TREEVIEW_NODE  node`

The node.

`XVT_IMAGE image`

The image for a node in the expanded state. If the image is NULL_IMAGE then the node will use the treeview control's expanded image.

Description

This function sets the expanded image for a node.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

See Also

`XVT_TREEVIEW_NODE`

`xvt_treeview_create_node`

`xvt_treeview_get_node_image_expanded`

xvt_treeview_set_node_image_item

Set the item image for a node

Summary

```
void xvt_treeview_set_node_image_item(
        XVT_TREEVIEW_NODE node, XVT_IMAGE image);
```

`XVT_TREEVIEW_NODE  node`

The node.

`XVT_IMAGE image`

The item image for a node. If the image is NULL_IMAGE then the node will use the treeview control's item image.

Description

This function sets the item image for a node.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

See Also

`XVT_TREEVIEW_NODE`

`xvt_treeview_create_node`

`xvt_treeview_get_node_image_item`

xvt_treeview_set_node_string

Set the item text for a node

Summary

```
void xvt_treeview_set_node_string(
        XVT_TREEVIEW_NODE node, char *string);
```

`XVT_TREEVIEW_NODE  node`

The node.

`char *string`

Null terminated character buffer for item text. Set sting to NULL for no item text.

Description

This function copies the item text to in string buffer and assigns it to the node.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

·string must not be null

See Also

`XVT_TREEVIEW_NODE`

`xvt_treeview_create_node`

`xvt_treeview_get_node_string`

xvt_treeview_set_node_type

Set node type

Summary

```
void xvt_treeview_set_node_type(
        XVT_TREEVIEW_NODE node,
        XVT_TREEVIEW_NODE_TYPE type);
```

`XVT_TREEVIEW_NODE  node`

The node.

`XVT_TREEVIEW_NODE_TYPE type`

The treeview node can be of type XVT_TREEVIEW_NODE_TERMINAL,a leaf with no children, or XVT_TREEVIEW_NODE_NONTERMINAL, a branch which may have have children.

Description

This function sets the node type assigned to a node. If the node type is changing from XVT_TREEVIEW_NODE_NONTERMINAL to XVT_TREEVIEW_NODE_TERMIAL all children of the node will be destroyed.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·node must not be null

·type is not of XVT_TREEVIEW_NODE_TYPE

See Also

`XVT_TREEVIEW_NODE`

`XVT_TREEVIEW_NODE_TYPE`

`xvt_treeview_create_node`

`xvt_treeview_destroy_node`

`xvt_treeview_get_node_type`

xvt_treeview_suspend

Suspend updating of treeview control

Summary

```
void xvt_treeview_suspend(WINDOW ctl_win);

WINDOW ctl_win
```

Window of control of type WC_TREEVIEW.

Description

This function suspends the updating of the treeview control until the updating is resumed with xvt_treeview_resume.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·ctl_win must not be null

·ctl_win must of type WC_TREEVIEW

See Also

`WIN_TYPE`

`xvt_treeview_resume`

`xvt_treeview_update`

xvt_treeview_update

Force update of treeview control

Summary

```
void xvt_treeview_suspend(WINDOW ctl_win);

WINDOW ctl_win
```

Window of control of type WC_TREEVIEW.

Description

This function forces the updating of the treeview control’s view.

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

·ctl_win must not be null

·ctl_win must of type WC_TREEVIEW

See Also

`WIN_TYPE`

`xvt_treeview_resume`

`xvt_treeview_suspend`

xvt_tx_*

Text Edit Functions

```
xvt_tx_add_par
xvt_tx_append
xvt_tx_clear
xvt_tx_create
xvt_tx_create_def
xvt_tx_destroy
xvt_tx_get_attr
xvt_tx_get_limit
xvt_tx_get_line
xvt_tx_get_margin
xvt_tx_get_next_tx
xvt_tx_get_num_chars
xvt_tx_get_num_lines
xvt_tx_get_num_par_lines
xvt_tx_get_num_pars
xvt_tx_get_origin
xvt_tx_get_sel
xvt_tx_get_tabstop
xvt_tx_get_view
xvt_tx_rem_par
xvt_tx_reset
xvt_tx_resume
xvt_tx_scroll_hor
xvt_tx_scroll_vert
xvt_tx_set_attr
xvt_tx_set_limit
xvt_tx_set_margin
xvt_tx_set_par
xvt_tx_set_scroll_callback
xvt_tx_set_sel
xvt_tx_set_tabstop
xvt_tx_suspend
```

xvt_tx_add_par

Add Paragraph to Text Edit Object

Summary

```
BOOLEAN xvt_tx_add_par(TXEDIT tx, T_PNUM pnum, char *s)

TXEDIT tx
```

Text edit object.

`T_PNUM pnum`

Number of the paragraph before which to add the string.

`char *s`

String to add.

Description

This function adds the `NULL`-terminated string `s` to the text edit object designated by `tx`. The string becomes a new paragraph. It is added before paragraph `pnum` (zero is the first). A `pnum` beyond the last paragraph (e.g., `USHRT_MAX`) causes the new paragraph to be added after the last paragraph.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful.

See Also

`T_PNUM`
`TXEDIT`
`xvt_tx_append`
`xvt_tx_rem_par`
`xvt_tx_set_par`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

```
FILE *in;
...
xvt_tx_suspend(txedit);
while (fgets(buf, sizeof(buf), in) != NULL) {
     len = strlen(buf);
    if (had_newline = (buf[len - 1] == '
'))
         buf[len - 1] = '0';
     if (new_par) {
         if (!xvt_tx_add_par(txedit, USHRT_MAX, buf)) {
             xvt_dm_post_error(
                "Error adding paragraph.");
             break;
         }
     }
    else if (!xvt_tx_append(txedit, USHRT_MAX, buf)) {
         xvt_dm_post_error(
             "Error appending to paragraph.");
         break;
     }
    new_par = had_newline;
 }
fclose(in);
xvt_dm_post_note("Read in %d paragraphs and %d lines.",
         xvt_tx_get_num_pars(txedit),
         xvt_tx_get_num_lines(txedit));
 xvt_tx_resume(txedit);
```

xvt_tx_append

Add to Text Edit Paragraph

Summary

```
BOOLEAN xvt_tx_append(TXEDIT tx, T_PNUM pnum, char *s)
```

`TXEDIT tx`

Text edit object.

`T_PNUM pnum`

Number of the paragraph on which to append the string.

`char *s`

String to append.

Description

This function appends the NULL-terminated string s to the end of paragraph pnum (zero is the first) in the text edit object designated by tx. If pnum is greater than the last paragraph (e.g., USHRT_MAX), the string is appended to the last paragraph.

Note: A newly-created text edit object has zero paragraphs. Therefore, you must first call xvt_tx_add_par before calling xvt_tx_append for a newly-created text edit object.

Return Value

TRUE if successful; FALSE if unsuccessful (on error).

See Also

T_PNUM
TXEDIT
xvt_tx_add_par
xvt_tx_rem_par
xvt_tx_set_par

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for xvt_tx_add_par.

xvt_tx_clear

Clear Text Edit Object

Summary

```
BOOLEAN xvt_tx_clear(TXEDIT tx)
```

TXEDIT tx

Text edit object.

Description

This function removes all text from the text edit object designated by tx without destroying tx. If you want to destroy a text edit object, call xvt_tx_destroy instead.

Note: A cleared text edit object has no paragraphs. Therefore, you must call xvt_tx_add_par before calling xvt_tx_append or xvt_tx_set_par.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful (on error).

See Also

```
TXEDIT
xvt_tx_add_par
xvt_tx_append
xvt_tx_destroy
xvt_tx_rem_par
xvt_tx_set_par
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

xvt_tx_create

Create Text Edit Object

Summary

```
TXEDIT xvt_tx_create(WINDOW win, RCT *rctp,
     unsigned attrib, XVT_FNTID font_id, int margin,
    int limit)
```

`WINDOW win`

Window in which to create the text edit object.

`RCT *rctp`

Border rectangle, in window-relative coordinates. If the `TX_BORDER` attribute isn't set, this is taken as the requested view rectangle. If `TX_BORDER` is set, the view rectangle is inset by 4 pixels. Also, the bottom coordinate of the view rectangle may be reduced so that an integral number of text lines will appear in the view.

`unsigned attrib`

Attribute flags, normally formed by ORing one or more of the symbols defined in `TX_* Attributes`.

`XVT_FNTID font_id`

Handle of the logical font. All text in a single text edit object must have the same logical font (same family, style, and point size). Your application can get a logical font by calling `xvt_font_create` or `xvt_res_get_font`, or by copying the

`v.font.font_id member` from an `E_FONT` event with `xvt_font_copy`.

`int margin`

Right margin, in pixels. This is meaningful only if the `TX_WRAP` attribute is set. A value of zero means wrap to the view width.

`int limit`

Character limit. The maximum number of characters that can be typed into a single paragraph. A value of zero implies no limit.

Description

This function creates a new text edit object in an existing window and returns an object of type `TXEDIT`, which must be used in subsequent calls to operate on the object. The application owns the `font_id`, and is responsible for allocating and freeing it. The text edit system creates its own copy of the logical font.

Note: You can also use `xvt_tx_create_def` to create text edit objects. This function has all of the capabilities of `xvt_tx_create`, plus it allows your application to assign application data to the text edit object and to give the text edit object an ID. The ID can be used later in a call to `xvt_win_get_tx` to retrieve a `TXEDIT` in a particular window. Also note that you can use `xvt_win_create_def` and `xvt_win_create_res` to create a window and multiple text edit objects with a single function call.

It is not possible to create a text edit object in a dialog.

Return Value

The `TXEDIT` descriptor if successful; `NULL_TXEDIT` if unsuccessful.

Parameter Validity and Error Conditions

XVT issues an error if any of the following are true:

- There is insufficient memory available.
- The function is called during an `E_UPDATE`
- The window is invalid
- The font is invalid
- `rctp` is `NULL`

See Also

E_FONT
RCT
TX_* Attributes
TXEDIT
WINDOW
XVT_FNTID
xvt_font_copy
xvt_font_create
xvt_res_get_font
xvt_tx_create_def
xvt_tx_destroy
xvt_tx_set_scroll_callback
xvt_tx_set_tabstop
xvt_win_create_def
xvt_win_create_res
xvt_win_get_tx

The "Controls" chapter in the *XVT Portability Toolkit Guide*

xvt_tx_create_def

Create a Text Edit Object from a Data Structure

Summary

```
TXEDIT xvt_tx_create_def(WIN_DEF *win_def_p, WINDOW
parent_win, long app_data)
```

`WIN_DEF *win_def_p`

Pointer to a `WIN_DEF` structure (see description below).

`WINDOW parent_win`

Parent window in which the text edit object will be created.

`long app_data`

Contains any application data you wish to attach to the text edit object (it usually is a pointer to a data structure cast into a `long`).

Description

This function adds an XVT text edit object to the `win` parent window. This function does *not* add text edit objects to dialogs.

`win_def_p` points to a `WIN_DEF` structure describing the text edit object to be created. When filling in the `WIN_DEF`, keep in mind the following:

- win_def_p->wtype is always set to WC_TEXTEDIT.
- win_def_p->rct defines the bounding rectangle for the text edit object in terms of the window's client area.
- win_def_p->v.tx.attrib specifies the attribute flag of the text edit object. You set the flag by ORing together one or more of the TX_* Attributes.
- win_def_p->text is used to set the text of the text edit object. It must either be NULL or point to a NULL-terminated string.
- win_def_p->units is one of the type U_PIXELS, U_CHARS, or U_SEMICHARS, and specifies the units used to measure the bounding rectangle defined by win_def_p->rct.
- win_def_p->v.tx.margin is used to set the width of the right margin in pixels. It is meaningful only if the TX_WRAP attribute is set. A value of zero means wrap to the view width.
- win_def_p->v.tx.tx_id is an ID number for the text edit object. This is a way of uniquely identifying a text edit object in a window. Keep in mind that it is not necessary to specify an ID, but if you choose to use it in your application, then you must set this field. All of the IDs for the text edit objects in a window must be unique.
- win_def_p->v.tx.limit defines the maximum number of characters allowed per parargraph. Setting win_def_p->v.tx.limit to zero implies no limit.
- win_def_p->v.tx.font_id is the logical font to be displayed in the text edit object. You can obtain a valid font structure by calling xvt_font_create.

Keep in mind that xvt_tx_create_def has all of the capabilities of xvt_tx_create, plus it allows the application to give the text edit object an ID. The ID can later be used in a call to xvt_win_get_tx to retrieve a TXEDIT in a particular window.

Return Value

A TXEDIT if successful; NULL_TXEDIT if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if any of the following are true:

- There is insufficient memory available.
- The function is called during an E_UPDATE event
- The window is invalid
- win_def_p->wtype is not WC_TEXTEDIT

- `win_def_p->units` is not `U_PIXELS`, `U_CHARS`, or `U_SEMICHARS`
- `win_def_p->v.tx.font_id` is not a valid logical font

See Also

```
TXEDIT
TX_* Attributes
U_* Values for UNIT_TYPE
U_* Values for UNIT_TYPE
WIN_DEF
WINDOW
xvt_tx_create
xvt_tx_destroy
xvt_tx_set_tabstop
xvt_vobj_get_data
xvt_vobj_set_data
xvt_win_create_def
xvt_win_create_res
xvt_win_get_tx
```

The "Controls" chapter in the *Guide*

xvt_tx_destroy

Destroy Text Edit Object

Summary

```
BOOLEAN xvt_tx_destroy(TXEDIT tx)

TXEDIT tx
```

Text edit object.

Description

This function deletes all text from the text edit object designated by `tx` and releases all memory associated with it. Finally, the value of `tx` will no longer designate a valid text edit object.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful (on error).

See Also

```
TXEDIT
xvt_tx_clear
xvt_tx_create
xvt_tx_create_def
xvt_tx_rem_par
```

The "Controls" chapter in the *Guide*

xvt_tx_get_attr

Get Text Edit Attributes

Summary

```
unsigned xvt_tx_get_attr(TXEDIT tx)
```

`TXEDIT tx`

Text edit object.

Description

This function retrieves the attributes of the text edit object designated by `tx`. The attributes can be set by `xvt_tx_create`, `xvt_tx_create_def`, or `xvt_tx_set_attr`. For a complete description of the attributes, see `xvt_tx_create`.

Return Value

The attributes OR'd together.

See Also

```
TXEDIT
xvt_tx_create
xvt_tx_create_def
xvt_tx_set_attr
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

xvt_tx_get_limit

Get Text Edit Character Limit

Summary

```
int xvt_tx_get_limit(TXEDIT tx)
```

`TXEDIT tx`

Text edit object.

Description

This function retrieves the character limit (the maximum number of characters per paragraph) of the text edit object designated by `tx`. The character limit is set by `xvt_tx_create`, `xvt_tx_create_def`, or `xvt_tx_set_limit`.

Return Value

The character limit.

See Also

`TXEDIT`
`xvt_tx_create`
`xvt_tx_create_def`
`xvt_tx_set_limit`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

xvt_tx_get_line

Get Line from Text Edit Object

Summary

```
char *xvt_tx_get_line(TXEDIT tx, T_PNUM pnum,
     ACCESS_CMD cmd, T_LNUM lnum, unsigned *lenp)
```

`TXEDIT tx`

Text edit object.

`T_PNUM pnum`

Number of the paragraph that contains the line.

`ACCESS_CMD cmd`

Command. For each line, you must call it three times: once with cmd set to A_LOCK, once with cmd set to A_GET, and once with cmd set to A_UNLOCK.

`T_LNUM lnum`

Line number in the paragraph.

`unsigned *lenp`

Returned length of the line in bytes.

Description

This function retrieves the text of line `lnum` in paragraph `pnum` in the text edit object designated by `tx`. For each line, you must call it three times: once with `cmd` set to `A_LOCK`, once to actually get the text with `cmd` set to `A_GET`, and once with `cmd` set to `A_UNLOCK`. For the `A_LOCK` and `A_UNLOCK` calls, the `lenp` argument can be `NULL`, and the return value should be ignored.

For the `A_GET` call, a pointer to the text line is returned. That pointer is valid until the `A_UNLOCK` call is made. The length of the line is returned through the `lenp` argument. This is essential, because the text itself is not `NULL`-terminated.

Note: You must *not* interleave `A_LOCK / A_GET / A_UNLOCK` sequences. The complete three-call sequence for a line must be completed before another line can be accessed.

Return Value

A pointer to the text if successful; `NULL` if `pnum` or `lnum` are out of range. No valid data is returned if `cmd` is `A_LOCK` or `A_UNLOCK`. Do not attempt to free or modify the line text.

See Also

```
ACCESS_CMD
T_LNUM
T_PNUM
TXEDIT
xvt_tx_get_num_chars
xvt_tx_get_num_lines
xvt_tx_get_num_pars
xvt_tx_get_num_par_lines
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

The following code illustrates how to get the text in the first visible line of a text edit:

```
    void
    text_get_string(TXEDIT tx, char *line, int line_len)
    {
         if (tx != NULL_TXEDIT)
        {
             char *lp;
            T_PNUM pnum;
            T_LNUM lnum;
            unsigned int len;
            xvt_tx_get_origin(tx, &pnum, &lnum, NULL,
                NULL);
             (void) xvt_tx_get_line(tx, pnum, A_LOCK,
            lnum, &len);
            lp = xvt_tx_get_line(tx, pnum, A_GET, lnum,
                &len);
             xvt_str_copy_n_size(line, lp, min(len,
line_len-1));
            line[len] = '0';
            (void) xvt_tx_get_line(tx, pnum, A_UNLOCK,
                lnum, &len);
         }
        else
             line[0] = '0';
     }
```

xvt_tx_get_margin

Get Text Edit Margin

Summary

```
int xvt_tx_get_margin(TXEDIT tx)
```

`TXEDIT tx`

Text edit object.

Description

This function retrieves the right margin of the text edit object designated by `tx`, as set by `xvt_tx_create`, `xvt_tx_create_def`, or `xvt_tx_set_margin`.

Note: This value is meaningful only if the `TX_WRAP` attribute is set.

Return Value

The margin in pixels. A value of zero means that word wrapping occurs at a value slightly less than the view width.

See Also

```
TXEDIT
xvt_tx_create
xvt_tx_create_def
xvt_tx_set_margin
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

xvt_tx_get_next_tx

Get Next Text Edit Object

Summary

```
TXEDIT xvt_tx_get_next_tx(TXEDIT tx, WINDOW win)
```

`TXEDIT tx`

Previous textedit object or `NULL_TXEDIT`.

`WINDOW win`

Parent window or `NULL_WIN`.

Description

This function returns the next text edit object from the internal list maintained by the text edit module. If `win` is a valid window, the next text edit object whose parent window is `win` is returned. If `win` is `NULL_WIN`, then the next text edit object in any window is returned. If the `tx` parameter is `NULL_TXEDIT`, the first text edit object that meets the criteria above is returned; otherwise, the search in the list starts from `tx` in the list.

Nothing should be assumed about the order of text edit objects in the internal list.

Return Value

The next valid text edit object after `tx` that meets the criteria set by the `win` parameter; `NULL_TXEDIT` if the end of the list is reached.

See Also

```
NULL_TXEDIT
TXEDIT
WINDOW
xvt_win_get_tx
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

The following example clears all of the text edit objects in the application:

```
    TXEDIT tx = NULL_TXEDIT;while (NULL_TXEDIT != (tx =
xvt_tx_get_next_tx(tx, NULL_WIN)))
        xvt_tx_clear(tx);
```

xvt_tx_get_num_chars

Get Number of Characters in Text Edit Line

Summary

```
T_CNUM xvt_tx_get_num_chars(TXEDIT tx, T_PNUM pnum,
     T_LNUM lnum)
```

`TXEDIT tx`

Text edit object.

`T_PNUM pnum`

Number of the paragraph containing the line.

`T_LNUM lnum`

Line number in the paragraph.

Description

This function retrieves the number of characters in line `lnum` of paragraph `pnum` of the text edit object designated by `tx`. Line numbers and paragraph numbers both start at zero. An `lnum` that's too large is taken to mean the last line of the paragraph.

Return Value

The number of characters if successful; zero if `pnum` is out of range.

See Also

```
T_CNUM
T_LNUM
T_PNUM
TXEDIT
xvt_tx_get_num_pars
xvt_tx_get_num_par_lines
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

xvt_tx_get_num_lines

Get Number of Lines in Text Edit Object

Summary

```
T_LNUM xvt_tx_get_num_lines(TXEDIT tx)

TXEDIT tx
```

Text edit object.

Description

This function retrieves the total number of lines in the text edit object designated by `tx`. This function is useful when computing the range of the vertical scrollbar of a window containing the text edit object.

If `TX_WRAP` is not set, then the number of lines is equal to the number of paragraphs. If `TX_WRAP` is set, then the number of lines is greater than or equal to the number of paragraphs, due to wrapping.

Return Value

The number of lines.

See Also

`T_LNUM`
`TXEDIT`
`xvt_tx_add_par`
`xvt_tx_get_num_pars`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_tx_add_par`.

xvt_tx_get_num_par_lines

Get Number of Lines in Text Edit Paragraph

Summary

```
T_LNUM xvt_tx_get_num_par_lines(TXEDIT tx, T_PNUM pnum)

TXEDIT tx
```

Text edit object.

`T_PNUM pnum`

Number of the paragraph from which to inquire line count.

Description

This function retrieves the number of lines in paragraph `pnum` (zero is the first) of the text edit object designated by `tx`. This function is useful for computing the number of lines to scroll via `xvt_tx_scroll_vert` to view a particular paragraph.

Note that if you add the return value from this function for each paragraph in a text edit object, this value will equal the return value of `xvt_tx_get_num_lines`.

Return Value

The number of lines if successful; zero if `pnum` is out of range.

See Also

`T_PNUM`
`TXEDIT`
`xvt_tx_get_num_lines`
`xvt_tx_get_num_pars`
`xvt_tx_scroll_vert`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

xvt_tx_get_num_pars

Get Number of Paragraphs in Text Edit Object

Summary

```
T_PNUM xvt_tx_get_num_pars(TXEDIT tx)

TXEDIT tx
```

Text edit object.

Description

This function retrieves the total number of paragraphs in the text edit object designated by `tx`. Note that if the `TX_WRAP` attribute isn't set, the number of lines is equal to the number of paragraphs.

Return Value

The number of paragraphs.

See Also

`T_PNUM`
`TXEDIT`
`TX_* Attributes`
`xvt_tx_add_par`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_tx_add_par`.

xvt_tx_get_origin

Get Top Paragraph and Line of View Rectangle

Summary

```
void xvt_tx_get_origin(TXEDIT tx, T_PNUM *pnump,
     T_LNUM *lnump, T_LNUM *org_linep,
     T_CPOS *org_offsetp)
```

`TXEDIT tx`

Text edit object.

`T_PNUM *pnump`

Paragraph number containing top line.

`T_LNUM *lnump`

Line number within the paragraph.

`T_LNUM *org_linep`

Line number from the start of the whole text edit object.

`T_CPOS *org_offsetp`

Horizontal origin of the view, in pixels.

Description

This function gets the paragraph and line shown at the top of the view rectangle for the text edit object designated by `tx`. The absolute paragraph number and line number within that paragraph are returned through the `pnump` and `lnump` arguments. In addition to

returning the top visible line in terms of the paragraph and a line within that paragraph, this function returns the line at the top of the view in terms of an absolute line count from the top of the text edit object. That value is returned through the `org_linep` argument.

This function returns the horizontal origin of the view in terms of the number of pixels that have been shifted to the left of the view, through the `org_offsetp` argument. Your application can set any of these pointers to `NULL` if it doesn't need that particular data item.

An application can create scrollbar controls around a text edit object, but the application must do the scrolling and scrollbar manipulation itself. This function is useful when a text edit object is created along with application-created scrollbar controls. You can use it to determine where to set the thumb position, or to figure out how much to scroll when the thumb of the scrollbar is operated.

Note: The function `xvt_tx_get_num_lines` is also useful when manipulating scrollbars, to determine the vertical scroll range.

See Also

`T_CPOS`
`T_LNUM`
`T_PNUM`
`TXEDIT`
`xvt_tx_get_line`
`xvt_tx_get_num_lines`
`xvt_tx_get_num_pars`
`xvt_tx_get_num_par_lines`
`xvt_tx_set_scroll_callback`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_tx_get_line`.

xvt_tx_get_sel

Get Text Edit Selection

Summary

```
void xvt_tx_get_sel(TXEDIT tx, T_PNUM *p1, T_LNUM *l1,
     T_CNUM *c1, T_PNUM *p2, T_LNUM *l2, T_CNUM *c2)

TXEDIT tx
```

Text edit object.

`T_PNUM *p1`

Starting paragraph number.

`T_LNUM *l1`

Starting line number.

`T_CNUM *c1`

Starting character number.

`T_PNUM *p2`

Ending paragraph number.

`T_LNUM *l2`

Ending line number.

`T_CNUM *c2`

Ending character number.

Description

This function gets the boundaries of the text selection for the text edit object designated by `tx`. The selection starts with paragraph `p1`, line `l1` within the paragraph, character `c1` within the line, and it ends just before `p2`, `l2`, `c2`. All numbering starts with zero.

If there is no current text selection, but there is a current insertion point, the return values will be such that (`p1 == p2 && l1 == l2 && c1 == c2`), where the values returned in `p1`, `l1`, and `c1` give the current insertion point. In this manner, both the text selection and the current insertion point are handled by the same functions.

See Also

`T_CNUM`
`T_CPOS`
`T_LNUM`
`T_PNUM`
`TXEDIT`
`xvt_tx_rem_par`
`xvt_tx_set_sel`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_tx_rem_par`.

xvt_tx_get_tabstop

Get Text Edit Tabstop

Summary

```
T_CNUM xvt_tx_get_tabstop(TXEDIT tx)
```

`TXEDIT tx`

Text edit object.

Description

This function retrieves the current tabstop value of the text edit object designated by `tx`. This value is the number of average width characters between tab stops.

Return Value

The current tabstop value.

See Also

`T_CNUM`
`TXEDIT`
`xvt_tx_set_tabstop`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

xvt_tx_get_view

Get Text Edit View Rectangle

Summary

```
RCT *xvt_tx_get_view(TXEDIT tx, RCT *rct
```

`)TXEDIT tx`

Text edit object.

`RCT *rct`

Address of a rectangle.

Description

This function retrieves the view rectangle of the text edit object designated by `tx`. For a discussion of the view rectangle, see

`xvt_tx_create`. The view rectangle is the same as or inside of the border rectangle, which can be retrieved separately with `xvt_vobj_get_outer_rect`.

You can't directly change the view rectangle; instead, change the border with `xvt_vobj_move`.

The rectangle coordinates are copied into the rct whose address is passed as a parameter, and this address is returned as the function value.

Return Value

The rectangle address passed as a parameter.

See Also

`RCT`
`TXEDIT`
`xvt_tx_create`
`xvt_tx_get_origin`
`xvt_vobj_get_outer_rect`
`xvt_vobj_move`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

xvt_tx_rem_par

Delete Text Edit Paragraph

Summary

```
BOOLEAN xvt_tx_rem_par(TXEDIT tx, T_PNUM pnum)
```

`TXEDIT tx`

Text edit object.

`T_PNUM pnum`

Number of the paragraph to be deleted.

Description

This function deletes the paragraph `pnum` (zero is the first) from the text edit object designated by `tx`.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful (on error).

See Also

```
T_PNUM
TXEDIT
xvt_tx_add_par
xvt_tx_append
xvt_tx_get_num_pars
xvt_tx_set_par
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

The following code deletes the selected first paragraph:

```
TXEDIT txedit;
T_PNUM p1, p2;
T_LNUM l1, l2;
T_CNUM c1, c2;
    ...
 xvt_tx_get_sel(txedit, &p1, &l1, &c1,
     &p2, &l2, &c2);
 if (!xvt_tx_rem_par(txedit, p1))
     xvt_dm_post_error("Error deleting paragraph.");
```

xvt_tx_reset

Reset Text Edit Object

Summary

```
void xvt_tx_reset(TXEDIT tx)

TXEDIT tx
```

Text edit object.

Description

This function resets the text edit object designated by `tx`. Any selected text is unselected, the caret is positioned before the first character, the text is scrolled as far up and to the left as possible, all paragraphs are rewrapped, and an update event is queued for the border rectangle.

Note: Changing any of the following three attributes via the function `xvt_tx_set_attr` also results in `xvt_tx_reset` being called: `TX_WRAP`, `TX_BORDER`, and `TX_ONEPAR`. In addition, calling `xvt_tx_set_margin` or `xvt_tx_set_limit` can also cause `xvt_tx_reset` to be called.

See Also

```
TX_* Attributes
TXEDIT
xvt_tx_clear
xvt_tx_set_attr
xvt_tx_set_limit
xvt_tx_set_margin
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

xvt_tx_resume

Resume Text Edit Screen Updating

Summary

```
void xvt_tx_resume(TXEDIT tx)
```

`TXEDIT tx`

Text edit object.

Description

This function causes screen updating to be resumed for the text edit object designated by `tx`. Call it to resume and repaint a text edit object that you suspended with `xvt_tx_suspend`.

See Also

```
TXEDIT
xvt_tx_add_parxvt_tx_suspend
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_tx_add_par`.

xvt_tx_scroll_hor

Scroll Text Edit Object Horizontally

Summary

```
void xvt_tx_scroll_hor(TXEDIT tx, int pixel_amt)
```

`TXEDIT tx`

Text edit object.

`int pixel_amt`

Amount in pixels to scroll horizontally.

Description

This function scrolls the text in the view rectangle of the text edit object designated by `tx`, by `pixel_amt` pixels in the horizontal direction. If `pixel_amt` is positive the text moves to the right; if negative, to the left.

You can use `xvt_tx_get_origin` to determine by how much the text edit is currently scrolled horizontally.

See Also

`TXEDIT`
`xvt_tx_scroll_vert`
`xvt_tx_get_origin`
`xvt_tx_set_scroll_callback`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

```
void do_hscroll(tx, what)
TXEDIT tx;
SCROLL_CONTROL what;
{
     switch (what) {
    case SC_LINE_UP:
         xvt_tx_scroll_hor(tx, 10);
        break;
     case SC_LINE_DOWN:
         xvt_tx_scroll_hor(tx, -10);
        break;
     case SC_PAGE_UP:
         xvt_tx_scroll_hor(tx, 100);
        break;
     case SC_PAGE_DOWN:
         xvt_tx_scroll_hor(tx, -100);
        break;
     case SC_THUMB:
         xvt_dm_post_note("not implemented yet");
        break;
     }
 }
```

xvt_tx_scroll_vert

Scroll Text Edit Object Vertically

Summary

```
void xvt_tx_scroll_vert(TXEDIT tx, int line_amt)
```

`TXEDIT tx`

Text edit object.

`int line_amt`

Lines to scroll vertically.

Description

This function scrolls vertically the text in the view rectangle of the text edit object designated by `tx` by `line_amt` lines. If `line_amt` is positive, the text moves downward; if negative, upward.

See Also

```
TXEDIT
xvt_tx_get_origin
xvt_tx_scroll_hor
xvt_tx_set_scroll_callback
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

```
void do_vscroll(tx, what)
TXEDIT tx;
SCROLL_CONTROL what;
{
     switch (what) {
    case SC_LINE_UP:
         xvt_tx_scroll_vert(tx,1);
        break;
     case SC_LINE_DOWN:
         xvt_tx_scroll_vert(tx, -1);
        break;
     case SC_PAGE_UP:
         xvt_tx_scroll_vert(tx, 10);
        break;
     case SC_PAGE_DOWN:
         xvt_tx_scroll_vert(tx, 10);
        break;
     case SC_THUMB:
         xvt_dm_post_note("not implemented yet");
        break;
     }
 }
```

xvt_tx_set*

xvt_tx_set_* Functions

```
xvt_tx_set_attr
xvt_tx_set_limit
xvt_tx_set_margin
xvt_tx_set_par
xvt_tx_set_scroll_callback
xvt_tx_set_sel
xvt_tx_set_tabstop
```

xvt_tx_set_attr

Change Text Edit Attributes

Summary

```
void xvt_tx_set_attr(TXEDIT tx, unsigned attrib)
```

`TXEDIT tx`

Text edit object.

`unsigned attrib`

New attributes.

Description

This function changes the attributes of the text edit object designated by `tx` to those specified by `attrib`. For a list of attributes, see `xvt_tx_create`. The text edit object is automatically reset if any of the following attributes are changed: `TX_WRAP`, `TX_BORDER`, or `TX_ONEPAR`.

To alter a single attribute bit, you should first get the current attributes via `xvt_tx_get_attr`, then set or reset the single attribute bit of interest, and finally set the new attributes via `xvt_tx_set_attr`. For example, to turn off the read only attribute of a text edit object, you should do the following:

```
    xvt_tx_set_attr(tx, xvt_tx_get_attr(tx) &
~TX_READONLY);
```

See Also

```
TX_* Attributes
TXEDIT
xvt_tx_create
xvt_tx_get_attr
xvt_tx_reset
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

xvt_tx_set_limit

Change Text Edit Character Limit

Summary

```
void xvt_tx_set_limit(TXEDIT tx, int limit)
```

`TXEDIT tx`

Text edit object.

`int limit`

New character limit value.

Description

This function changes the character limit of the text edit object designated by `tx` to the value `limit`. The character limit is the maximum number of characters allowed per paragraph. If `limit` is zero, there is no character limit.

The text edit object is not updated or reset. If the number of characters in any paragraph is above the new limit, the paragraph is left as is.

See Also

`TXEDIT`
`xvt_tx_create`
`xvt_tx_create_def`
`xvt_tx_get_limit`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

xvt_tx_set_margin

Change Text Edit Margin

Summary

```
void xvt_tx_set_margin(TXEDIT tx, int margin)
```

`TXEDIT tx`

Text edit object.

`int margin`

New right margin.

Description

This function changes the right margin of the text edit object designated by `tx` to that specified by `margin`. The text edit object is automatically reset (see `xvt_tx_reset`), so the text will be wrapped to the new margin. No wrapping or resetting occurs, of course, if the `TX_WRAP` attribute isn't set. A value of zero means that word wrapping occurs at a value slightly less than the view width.

See Also

```
TX_* Attributes
TXEDIT
xvt_tx_create
xvt_tx_create_def
xvt_tx_get_margin
xvt_tx_reset
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

---

xvt_tx_set_par

Change Text Edit Paragraph

Summary

```
BOOLEAN xvt_tx_set_par(TXEDIT tx, T_PNUM pnum, char *s)
```

`TXEDIT tx`

Text edit object.

`T_PNUM pnum`

Number of the paragraph whose text is to be replaced.

`char *s`

Replacement paragraph.

Description

This function completely replaces the text of paragraph `pnum` (zero is the first) in the text edit object designated by `tx` with the `NULL`-terminated string `s`.

Paragraph `pnum` must exist. Hence, this function can't be used to add text to a brand-new or cleared text edit object. Use `xvt_tx_add_par` instead.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful (on errror).

See Also

```
T_PNUM
TXEDIT
xvt_tx_add_par
xvt_tx_append
xvt_tx_rem_par
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

The following code changes the text of the selected first paragraph:

```
    TXEDIT txedit;
    T_PNUM p1, p2;
    T_LNUM l1, l2;
    T_CNUM c1, c2;xvt_tx_get_sel(txedit, &p1, &l1, &c1,
         &p2, &l2, &c2);
     if (!xvt_tx_set_par(txedit, p1,
             "This is the new paragraph text."))
         xvt_dm_post_error("Error changing paragraph.");
```

xvt_tx_set_scroll_callback

Set Text Edit Scroll Callback Function

Summary

```
void xvt_tx_set_scroll_callback(TXEDIT tx,
     SCROLL_CALLBACK fcn)
```

`TXEDIT tx`

Text edit object.

`SCROLL_CALLBACK fcn`

Scroll callback function.

Description

This function sets the scroll-activity callback function for the text edit object designated by `tx`. If you want to logically connect horizontal or vertical scrollbars to the horizontal or vertical scrolling behavior of the text edit object, call `xvt_tx_set_scroll_callback` after creating a text edit object.

Setting the scroll callback allows your application to be notified when the text edit object automatically scrolls itself due to the user dragging out a selection or using the arrow keys. When this happens, your callback is called, and you can adjust the thumb position of the scrollbars to reflect the new view.

The scroll callback function is also called if the number of lines in the text edit object changes. This allows you to adjust the vertical scrollbar range appropriately.

You should declare your callback function as follows:

```
void XVT_CALLCONV1 scroll_callback(TXEDIT tx,
     T_LNUM org_line, T_LNUM nlines, T_CPOS
    org_offset)
 {
     ...
 }
```

The text edit system calls your function whenever text in the view rectangle is scrolled or the number of lines changes. The `tx` argument designates the affected text edit object. The `org_line` argument is the number of the line at the top of the new view rectangle, numbered from the top of the text edit object's text, starting with zero (numbering spans paragraphs). The `nlines` argument is the total number of lines in the object (*not* in the view). The `org_offset` argument is the number of pixels to the left of the view.

The `org_line` and `org_offset` parameters are the same values returned by `xvt_tx_get_origin` in the `org_linep` and `org_offsetp` parameters. The parameter `nlines` is the same as the return value for `xvt_tx_get_num_lines`.

To prevent unnecessary scrollbar updating, `org_line`, `nlines`, and `org_offset` may be equal to `USHRT_MAX` if they have not changed since the previous callback.

Typically you'll do the following on each call:

- If `nlines` isn't equal to `USHRT_MAX`, set the range of the vertical scrollbar to start at zero and end at `nlines` with a call to `xvt_sbar_set_range`; if you are using the scrollbar proportion (described later), then set the upper bound of the scroll range to `nlines` plus the value used for the scrollbar proportion
- If `org_line` isn't `USHRT_MAX`, set the position of the vertical scrollbar to `org_line` with a call to `xvt_sbar_set_pos`

- If `org_offset` isn't `USHRT_MAX`, set the position of the horizontal scrollbar to `org_offset` with a call to `xvt_sbar_set_pos`

If you want to have the vertical scrollbar thumbsize reflect the viewable portion of the document with respect to the total document, then call `xvt_sbar_set_proportion` when the text edit object is created. Set the scroll proportion to the number of lines that are visible in the text edit object. This number can be computed by dividing the height of the text edit object by the height of the mapped logical font (which you can obtain with `xvt_font_get_metrics`).

The range of the horizontal scrollbar isn't meaningful to the text edit system; you need to set it, probably the first time your callback is invoked. You can set it to whatever you like (e.g., 0 to 2000).

See Also

```
SCROLL_CALLBACK
TXEDIT
USHRT_MAX
XVT_CALLCONV*
xvt_font_get_metrics
xvt_sbar_set_pos
xvt_sbar_set_proportion
xvt_sbar_set_range
xvt_tx_get_num_lines
xvt_tx_get_origin
xvt_tx_scroll_hor
xvt_tx_scroll_vert
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

```
...
     xvt_tx_set_scroll_callback(tx, scroll_callback);
 ...void XVT_CALLCONV1 scroll_callback(TXEDIT tx,
     T_LNUM org_line, T_LNUM nlines, T_CPOS org_offset)
 {
     static BOOLEAN first_horz_setting = TRUE;
    WINDOW win = xvt_vobj_get_parent(tx);if (nlines !=
USHRT_MAX)
         xvt_sbar_set_range(win, VSCROLL, 0, nlines)
     if (org_line != USHRT_MAX)
         xvt_sbar_set_pos(win, VSCROLL, org_line);
     if (org_offset != USHRT_MAX) {
         if (first_horz_setting) {
             first_horz_setting = FALSE;
            xvt_sbar_set_range(win, HSCROLL, 0, 2000);
         }
        xvt_sbar_set_pos(win, HSCROLL, org_offset);
     }
 }
```

xvt_tx_set_sel

Set Text Edit Selection

Summary

```
void xvt_tx_set_sel(TXEDIT tx, T_PNUM p1, T_LNUM l1,
     T_CNUM c1, T_PNUM p2, T_LNUM l2, T_CNUM c2)
```

`TXEDIT tx`

Text edit object.

`T_PNUM p1`

Starting paragraph number.

`T_LNUM l1`

Starting line number.

`T_CNUM c1`

Starting character number.

`T_PNUM p2`

Ending paragraph number.

`T_LNUM l2`

Ending line number.

`T_CNUM c2`

Ending character number.

Description

This function sets the text selection for the text edit object designated by `tx`. The selection starts with paragraph `p1`, line `l1` within the paragraph, character `c1` within the line, and it ends just before `p2`, `l2`, `c2`. The start can be after the end. All numbering starts with zero.

To set the current insertion point instead of the text selection, call `xvt_tx_set_sel` with parameters such that (p1 == p2 && l1 == l2 && c1 == c2), where the values in `p1`, `l1`, and `c1` specify the current insertion point. In this manner, both the text selection and current insertion point are handled by the same functions.

See Also

```
T_CNUM
T_CPOS
T_LNUM
T_PNUM
TXEDIT
xvt_tx_get_sel
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

xvt_tx_set_tabstop

Change Text Edit Tabstop

Summary

```
void xvt_tx_set_tabstop(TXEDIT tx, tabstop)
```

`TXEDIT tx`

Text edit object.

`T_CNUM tabstop`

New tabstop value.

Description

This function changes the tabstop value of the text edit designated by `tx` to that specified by `tabstop`. The text edit object is automatically reset (see `xvt_tx_reset`) so that all `tabstop`s will be redrawn according to the new `tabstop`. The `tabstop` value is the number of average-width characters between tab stops. The default value is eight.

See Also

```
T_CNUM
TXEDIT
xvt_tx_get_tabstop
xvt_tx_reset
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

xvt_tx_suspend

Suspend Text Edit Screen Updating

Summary

```
void xvt_tx_suspend(TXEDIT tx)
```

`TXEDIT tx`

Text edit object.

Description

This function causes screen updating to be suspended for the text edit object designated by `tx`. Call it when you are about to make a major change to the text, such as when loading text from a file with calls to `xvt_tx_add_par`.

To restore operations to normal and redraw the text edit object with its new contents, you must call `xvt_tx_resume`.

See Also

`TXEDIT`
`xvt_tx_add_par`
`xvt_tx_resume`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_tx_add_par`.

xvt_vobj_*

Visible Object Functions

```
xvt_vobj_destroy
xvt_vobj_get_attr
xvt_vobj_get_client_rect
xvt_vobj_get_data
xvt_vobj_get_flags
xvt_vobj_get_formatter
xvt_vobj_get_outer_rect
xvt_vobj_get_palet
xvt_vobj_get_parent
xvt_vobj_get_title
xvt_vobj_get_type
xvt_vobj_is_focusable
xvt_vobj_is_valid
xvt_vobj_move
xvt_vobj_raise
xvt_vobj_set_attr
xvt_vobj_set_data
xvt_vobj_set_enabled
xvt_vobj_set_formatter
xvt_vobj_set_palet
xvt_vobj_set_title
xvt_vobj_set_visible
xvt_vobj_translate_points
```

xvt_vobj_destroy

Close And Destroy Window

Summary

```
void xvt_vobj_destroy(WINDOW win)
```

`WINDOW win`

Window or control to close and destroy.

Description

This function closes and destroys any visible object, window, dialog, or control specified by `win`.

`win` can be any window created by the following:

```
xvt_win_create
xvt_win_create_res
xvt_win_create_def
xvt_print_create_win
```

Or, it can be any control created by the following:

```
xvt_ctl_create
xvt_ctl_create_def
```

Or, it can be a control created as a side-effect of calling the following:

```
xvt_win_create_res
xvt_win_create_def
```

Or, it can be a dialog created by the following:

```
xvt_dlg_create_res
xvt_dlg_create_def
```

For a regular window or dialog, you usually call `xvt_vobj_destroy` either in response to an `E_CLOSE` event, or when the user chooses a "Close" item from a menu.

Do not free the memory associated with a window or dialog before `xvt_vobj_destroy` is called, as additional events may be sent to the window or dialog during the closing process. You should free memory (application data) associated with a window or dialog in response to an `E_DESTROY` event for that window or dialog.

Once this function returns, you should not attempt to use the window any longer (not even to call `xvt_vobj_get_data`). An `E_DESTROY` event may be dispatched before the call to `xvt_vobj_destroy` returns. The safest thing to do is return from the event handler immediately after calling `xvt_vobj_destroy`.

See Also

```
E_CLOSE
E_DESTROY
WINDOW
xvt_app_create
xvt_ctl_create
xvt_ctl_create_def
xvt_dlg_create_def
xvt_dlg_create_res
xvt_pmap_create
xvt_print_create_win
xvt_win_create
xvt_win_create_def
xvt_win_create_res
xvt_vobj_get_data
```

Example

```
switch (cmd) {
case M_FILE_CLOSE:
     xvt_vobj_destroy(win);
    break;
    ...
```

xvt_vobj_get_attr

Retrieve Attribute Value

Summary

```
long xvt_vobj_get_attr(WINDOW win, long data)
```

`WINDOW win`

Window for which the attribute value is to be retrieved.

`long data`

Attribute for which the value is to be retrieved.

Description

This function retrieves the `attr` value from the attribute list for a window, `win`. If `win` is `NULL_WIN`, the `attr` is retrieved from the application attribute list. For certain attributes, this function can be called before `xvt_app_create` is called. For descriptions of these attributes, see the `XVT Portable Attributes` and the *XVT Platform-Specific Books*.

The type of return value from `xvt_vobj_get_attr` is a `long`, but this `long` value is the result of casting the true attribute type. In all cases where the true attribute type is *not* `long`, you need to cast the return of `xvt_vobj_get_attr` into the type that is appropriate for that particular attribute.

Return Value

The value of the attribute (cast to a `long`).

Parameter Validity and Error Conditions

XVT issues an error if the following parameter conditions are not met:

- If `attr` is an attribute that does require the use of `win`, then it must be a valid XVT `WINDOW`

- `attr` must be one of the portable `ATTR_*` constants defined in this reference, or one of the non-portable `ATTR_*` constants defined in the *XVT Platform-Specific Books*

Implementation Note

Platform-specific attributes, for example `ATTR_MAC_*`, are only defined on the platform for which they are intended and must be conditionally compiled with `XVTWS`. For example:

```
#if (XVTWS == MACWS)
     int min_size = (int)xvt_vobj_get_attr(NULL_WIN,
         ATTR_MAC_MIN_SIZE);
 #endif
```

Or, alternatively:

```
#ifdef ATTR_MAC_MIN_SIZE
     int min_size = (int)xvt_vobj_get_attr(NULL_WIN,
         ATTR_MAC_MIN_SIZE);
 #endif
```

See Also

```
XVT Portable Attributes
WINDOW
XVTWS, *WS Values
xvt_app_create
xvt_vobj_set_attr
```

xvt_vobj_get_client_rect

Get the Client Rectangle for a Window

Summary

```
RCT *xvt_vobj_get_client_rect(WINDOW win, RCT *rctp)
```

`WINDOW win`

Window whose client rectangle is being queried. This can be any XVT window.

`RCT *rctp`

Pointer to the client rectangle.

Description

This function gets the dimensions of the client area of any XVT WINDOW including regular windows, dialogs, controls, print windows, TASK_WINS, SCREEN_WINS, and XVT_PIXMAPS.

The client area of a WINDOW is the portion found inside the frame *excluding* the width of the border and any border decorations the WINDOW might have, such as the size borders, menubar, titlebars, and scrollbars.To get the dimensions of the window *including* the frame, menubar, and decorations, your application needs to call xvt_vobj_get_outer_rect.

The coordinates are relative to the coordinates of the WINDOW, and are stored in the RCT pointed to by rctp. The left and top members of the RCT structure are always zero, so the right member is equal to the width and the bottom member is equal to the height.

At creation time, the client area is set to the dimensions your application set when calling the creation function. If the window is resized by the user, your application receives an E_SIZE event. Upon receiving this event, your application does not need to call xvt_vobj_get_client_rect, as the new client size is contained in the E_SIZE event.

Note: Since controls and print windows do not get any events, calling xvt_vobj_get_client_rect is the *only* method for determining their client area.

Note: For the WC_HTML WINDOW type, the value returned from xvt_vobj_get_client_rect is identical to the value returned from xvt_vobj_get_outer_rect.

Return Value

RCT pointed to by rctp.

Parameter Validity and Error Conditions

XVT issues an error if win is not a valid WINDOW.

See Also

E_SIZE
RCT
SCREEN_WIN
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_vobj_get_outer_rect

The "Windows" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_vobj_move`.

xvt_vobj_get_data

Get Application Data Associated with a Window

Summary

```
long xvt_vobj_get_data(WINDOW win)
```

`WINDOW win`

Window, dialog, control, or pixmap whose application data is to be retrieved.

Description

This function retrieves the data for `win`. Every XVT `WINDOW` (regular, task, screen, print, dialog, control, or pixmap), can have a *long int* associated with it for your application's own use.

Frequently the application data is a pointer to a structure of your own design. In this case, your application should cast the return value from `xvt_vobj_get_data` into a pointer of the correct type.

You can associate application data with a `WINDOW` when it is created, with these creation functions:

```
xvt_ctl_create
xvt_ctl_create_def
xvt_dlg_create_def
xvt_dlg_create_res
xvt_tx_create_def
xvt_vobj_set_data
xvt_win_create
xvt_win_create_def
xvt_win_create_res
```

Alternatively, you can use `xvt_vobj_set_data` to associate application data once the `WINDOW` has been created (usually in the handling of `E_CREATE`). If you choose the latter approach, then remember not to use the application data before it has been set.

Note: When responding to an `E_DESTROY` event for a `WINDOW`, `xvt_vobj_get_data` is the only function you can call on the `WINDOW` being destroyed.

Return Value

`long` integer for application data associated with the `WINDOW`.

Parameter Validity and Error Conditions

XVT issues an error if `win` is not a valid `WINDOW` or `XVT_PIXMAP`.

See Also

```
E_CREATE
E_DESTROY
WINDOW
XVT_PIXMAP
xvt_ctl_create
xvt_ctl_create_def
xvt_dlg_create_def
xvt_dlg_create_res
xvt_font_get_app_data
xvt_tx_create_def
xvt_vobj_set_data
xvt_win_create
xvt_win_create_def
xvt_win_create_res
```

The "Windows" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_vobj_set_data`.

xvt_vobj_get_flags

Get Current State of the Creation Flags

Summary

```
long xvt_vobj_get_flags(WINDOW win)

WINDOW win
```

Window, dialog, or control which is being inquired.

Description

This function retrieves the *current state* of the creation flags of `win`.

The flags associated with a `vobj`'s visibility and enabled state reflect the *explicit* setting of those states, not the implicit values. This function does not traverse up `win`'s ancestors to see if one of them is

invisible or disabled. It is the application's responsibility to determine the enabled/visible status of any of `win`'s ancestors.

Return Value

The current state of the object creation flags;
an OR'd combination of `WSF_*` values if `win` is a window;
an OR'd combination of `DLG_FLAG_*` values if `win` is a dialog;
an OR'd combination of `CTL_FLAG_*` values if `win` is a control.

See Also

```
CTL_FLAG_* Options
DLG_FLAG_* Options
WIN_DEF
WINDOW
WSF_* Options Flags
xvt_ctl_create
xvt_ctl_create_def
xvt_dlg_create_def
xvt_win_create
xvt_win_create_def
```

Example

```
long flags;
flags = xvt_vobj_get_flags(win);
if(flags & WSF_DISABLED) {
     xvt_vobj_set_enabled(win, TRUE);
}
```

xvt_vobj_get_formatter

Get the Format Callback Function for a Window, Dialog, Or Control

Summary

```
XVT_FORMAT_HANDLER xvt_vobj_get_formatter(WINDOW win)
```

`WINDOW win`

XVT window, dialog, or control handle.

Description

The function returns a pointer to the format handler callback function assigned to the specified window by the application.

Return Value

Returns the pointer to the callback function assigned by the application.

Parameter and Validity Conditions

XVT returns NULL and issues an error if any of the following conditions occur:

- If any of the parameters is NULL.
- If win is not a valid XVT WINDOW.

See Also

XVT_FORMAT_HANDLER
XVT_PATTERN
xvt_pattern_create
xvt_pattern_destroy
xvt_pattern_match
xvt_vobj_set_formatter

xvt_vobj_get_outer_rect

Get Bounding Rectangle for Window

Summary

```
RCT *xvt_vobj_get_outer_rect(WINDOW win, RCT *rctp)
```

`WINDOW win`

Window whose bounding rectangle is to be retrieved.

`RCT *rctp`

Bounding rectangle.

Description

This function returns the outer rectangle for any XVT `WINDOW` including regular windows, dialogs, controls, print windows, `TASK_WIN`s, `SCREEN_WIN`s, and `XVT_PIXMAP`s.

In contrast to the client area of a `WINDOW`, the outer rectangle includes any border decorations the `WINDOW` might have. This is especially true for document windows that might have size borders, titlebars, menubars, and scrollbars.

The coordinates of the outer rectangle are relative to the container of the WINDOW. For example, if win is a control, the coordinates are in the coordinates of its parent window or dialog. If win is a dialog or TASK_WIN, the coordinates are relative to the SCREEN_WIN. If win is SCREEN_WIN, the coordinates are the size of the physical screen.

At creation time, the outermost rectangle is determined by XVT to be the client area plus the frame and border decorations. If the window is resized by the user, your application receives an E_SIZE event. Upon receiving this event, your application can call xvt_vobj_get_outer_rect to determine the size of the new outer rectangle.

Parameter Validity and Error Conditions

XVT issues an error if the any of the following parameter conditions are not met:

- win must be a valid XVT window or pixmap
- rctp must be non-NULL

See Also

E_SIZE
RCT
SCREEN_WIN
TASK_WIN
WINDOW
XVT_PIXMAP
xvt_vobj_get_client_rect
xvt_vobj_move

xvt_vobj_get_palet

Get a Visible Object's Palette

Summary

```
XVT_PALETTE xvt_vobj_get_palet(WINDOW win)

WINDOW win
```

Visible object from which to get the palette.

Description

This function gets the color palette associated with a visible object. If no palette has been explicitly set for this object, the palette associated with its nearest ancestor that has a palette is returned.

Return Value

The object's palette, or the palette associated with its nearest ancestor if available; otherwise NULL.

Parameter Validity and Error Conditions

XVT issues an error if win is NULL or invalid.

See Also

WINDOW
XVT_PALETTE
xvt_palet_create
xvt_vobj_set_palet

Example

See the example for xvt_palet_create.

---

xvt_vobj_get_parent

Get Parent of Window, Dialog, or Control

Summary

```
WINDOW xvt_vobj_get_parent(WINDOW win)
```

WINDOW win

Window, dialog, control, or pixmap whose parent is to be retrieved.

Description

This function returns the parent (container) window for any WINDOW that can be drawn on the screen including regular windows, dialogs, controls, TASK_WINs, SCREEN_WINs, and XVT_PIXMAPs.

The parent for a WINDOW is specified when the WINDOW is defined. Dialogs and TASK_WINs have SCREEN_WINs as their parents. Top-level windows have TASK_WINs or SCREEN_WINs as their parents. Child windows have other windows as their parents. If win is SCREEN_WIN or print window, NULL_WIN is returned.

Return Value

The parent of win.

Parameter Validity and Error Conditions

XVT issues an error if `win` is not a valid XVT window.

Implementation Note

If the application is running on XVT/Win32, and the attribute `ATTR_WIN_PM_DRAWABLE_TWIN` has been set, the parents of controls can be dialogs, regular windows, and `TASK_WIN`s.

See Also

`SCREEN_WIN`
`TASK_WIN`
`WINDOW`
`XVT_PIXMAP`
`xvt_ctl_create`
`xvt_ctl_create_def`
`xvt_dlg_create_def`
`xvt_dlg_create_res`
`xvt_pmap_create`
`xvt_vobj_get_outer_rect`
`xvt_vobj_move`
`xvt_win_create`
`xvt_win_create_def`
`xvt_win_create_res`

Example

See the example for `xvt_vobj_move`.

xvt_vobj_get_title

Get Title of Window or Control

Summary

```
char *xvt_vobj_get_title(WINDOW win, char *title,
     int sz_title)
```

`WINDOW win`

Window, dialog, or control whose title is to be retrieved.

`char *title`

Buffer to hold title.

`int sz_title`

Maximum buffer capacity.

Description

This function gets the title of a regular window, dialog, or control and stores it in `title`, whose maximum capacity (including the `NULL`-terminator) is `sz_title`. The title is truncated as needed to fit into `title`.

Not all control types have valid titles; only the following control types have valid title information:

```
WC_PUSHBUTTON
WC_RADIOBUTTON
WC_CHECKBOX
WC_EDIT
WC_TEXT
WC_LISTEDIT
WC_GROUPBOX
```

Calling `xvt_vobj_get_title` on other controls returns `NULL`.

Note: "Title" is loosely interpreted to mean, "whatever text is appropriate for the item in question." For example, for controls of type `WC_EDIT` and `WC_LISTEDIT`, this function returns the contents of the edit field.

Note: For the `WC_HTML WINDOW` type, `xvt_vobj_get_title` returns the title of the HTML page.

Return Value

Pointer to `title` if successful; `NULL` if the control type does not have a valid title. (See the above list of control types which have valid titles.)

Parameter Validity and Error Conditions

XVT issues an error if one of the following conditions is not met:

- `win` must be a valid `WINDOW`
- `title` must not be `NULL`
- `sz_title` must be greater than zero

See Also

```
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
xvt_vobj_set_title
xvt_win_set_doc_title
```

xvt_vobj_get_type

Get Type of Window

Summary

```
WIN_TYPE xvt_vobj_get_type(WINDOW win)
```

`WINDOW win`

Window, dialog, control, or pixmap whose type is retrieved.

Description

This function returns the type of a `WINDOW` specified by `win`, which can be any valid `WINDOW` object including regular, dialog, control, screen, print windows, and pixmaps.

The type of a `WINDOW` is defined implicitly by the creation function used to create the window. Regular windows are of type `W_DOC`, `W_PLAIN`, `W_DBL`, `WD_MODAL`, or `W_NOBORDER`. Dialogs are of type `WD_MODAL` or `WD_MODELESS`. Controls are of type `WC_*`. Task windows are of type `W_TASK`, and the screen windows of type `W_SCREEN`. Print windows are of type `W_PRINT`. Pixmaps are of type `W_PIXMAP`.

You could use `xvt_vobj_get_type`, for example, if you wanted to have the same drawing code to render a page layout on either the window or the printer, but you also wanted to be able to distinguish between the two for certain operations.

Return Value

The type of the window, which is one of the `WIN_TYPE` values.

See Also

```
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
WIN_TYPE
xvt_ctl_create
xvt_ctl_create_def
xvt_dlg_create_def
xvt_dlg_create_res
xvt_scr_list_wins
xvt_win_create
xvt_win_create_def
xvt_win_create_res
```

Example

See the example for `xvt_scr_list_wins`.

xvt_vobj_is_focusable

Check if Visible Object Can Receive Focus

Summary

```
BOOLEAN xvt_vobj_is_focusable(WINDOW win)
```

`WINDOW win`

Window, dialog, or control which is being inquired.

Description

This function finds out if `win` is capable of receiving focus. Focusability is a static property of a visible object and is not affected by its current visibility or enabled state.

Note: The `WC_HTML WINDOW` type, is not capable of gaining focus.

Return Value

`TRUE` if `win` is capable of receiving focus; `FALSE` otherwise.

See Also

`WINDOW`
`xvt_scr_set_focus_vobj`

xvt_vobj_is_valid

Check Whether a Window Handle is a Valid Window, Dialog, or Control

Summary

```
BOOLEAN xvt_vobj_is_valid(WINDOW win)
```

`WINDOW win`

Window, dialog, or control that is being inquired.

Description

This function tests to see whether `win` refers to a valid, visible object for a window, dialog, or control.

Return Value

`TRUE` if `win` is a valid object; `FALSE` otherwise.

See Also

`WINDOW`

xvt_vobj_move

Move and Resize Window

Summary

```
void xvt_vobj_move(WINDOW win, RCT *rctp)
```

`WINDOW win`

Window or control to be moved and/or resized.

`RCT *rctp`

New client rectangle.

Description

This function moves and/or resizes `win` so that its new client rectangle has coordinates as specified by `rctp`. `win` can be a regular window, dialog, or control. The coordinates specified by `rctp` are relative to the window's parent. The parent of dialogs is `SCREEN_WIN`. The parent of windows or controls is the `parent` that was passed to the function used to create the window or control.

Calling `xvt_vobj_move` on windows or dialogs generates `E_SIZE` events. This allows you to write your application in such a way that you only adjust window layout when an `E_SIZE` event is received.

Implementation Note

In the XVT/XM implementation, the window manager may not honor the request, so don't be surprised if the window doesn't show up at the requested location.

Normally, `TASK_WIN` is not a valid window for this function. However, on XVT/Win32, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before calling `xvt_app_create`. In that case, `TASK_WIN` would be a valid window for this function. On other platforms, any attempt to move the task window is simply ignored.

See Also

E_SIZE
RCT
SCREEN_WIN
WINDOW
xvt_app_create
xvt_ctl_create
xvt_ctl_create_def
xvt_dlg_create_def
xvt_dlg_create_res
xvt_vobj_get_client_rect
xvt_win_create
xvt_win_create_def
xvt_win_create_res

Example

This code illustrates how to change the height of a window without changing its position or width:

```
void
window_set_height(WINDOW window, short height)
{
     RCT client_rect;
    PNT position;
    position.h = 0;
    position.v = 0;
    xvt_vobj_translate_points(window,
         xvt_vobj_get_parent(window), &position, 1);
     xvt_vobj_get_client_rect(window, &client_rect);
    xvt_rect_set_pos(&client_rect, position);
    xvt_rect_set_height(&client_rect, height);
    xvt_vobj_move(window, &client_rect);
 }
```

xvt_vobj_raise

Raise the Given Window

Summary

```
void xvt_vobj_raise(WINDOW win)

WINDOW win
```

Window to be raised.

Description

This function raises the given WINDOW to the top of the stacking order among its siblings. For example, if the WINDOW is a child of a top-

level window, you can call this function to make WINDOW the top-most child of those parented to the top-level window. However, the top-level WINDOW is not raised above other windows on the display.

Return Value

None.

Parameter Validity and Error Conditions

Only windows and modeless dialogs are valid arguments to xvt_vobj_raise. For example, windows of type W_DOC, W_PLAIN, W_DBL, W_NO_BORDER, and WD_MODELESS are valid arguments to xvt_vobj_raise.

Implementation Note

Normally, TASK_WIN is not a valid window for this function. However, on XVT/Win32, if you set the ATTR_WIN_PM_DRAWABLE_TWIN attribute to TRUE, TASK_WIN is a valid window. On some platforms, it is not possible to raise windows above dialogs, especially modal dialogs.

See Also

TASK_WIN
WINDOW
W_*, WC_*, WD_*, Values for WIN_TYPE
xvt_scr_set_focus_vobj

---

xvt_vobj_set_attr

Set a Value in the System Attribute Table

Summary

```
void xvt_vobj_set_attr(WINDOW win, XVT_ATTR attr,
long value)
```

WINDOW win

Valid window or NULL_WIN.

XVT_ATTR attr

Attribute. It should be set to one of the ATTR_* constants.

long value

New value.

Description

This function sets a system-wide attribute value, which has an effect that depends on the attribute being set. Only a few of the XVT Attributes can be set by your application; the rest are "read-only." An example of some attributes that are settable by your application are `ATTR_ERRMSG_HANDLER` and `ATTR_EVENT_HOOK`.

For application-wide attributes, `win` is set to `NULL_WIN`. `attr` should be set to one of the `ATTR_*` constants.

`value` should be the new value. Although `value` is a `long` parameter, you can cast other data types, such as data pointers and function pointers, into a `long` and pass them to this function, as appropriate for the particular attribute you're setting.

For details of which attributes can be set, and the semantics of doing so, see `XVT Portable Attributes`.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are not met:

- `win` must be either a valid `WINDOW` or `NULL_WIN`, as appropriate for the attribute being set
- `attr` must be defined, and must not be a read-only attribute

Implementation Note

Platform-specific attributes, for example `ATTR_MAC_*`, are only defined on the platform for which they are intended and need to be conditionally compiled with `XVTWS`. For example:

```
    #if (XVTWS==MACWS)
        xvt_vobj_set_attr(NULL_WIN, ATTR_MAC_MIN_SIZE,
1OOL);
     #endif
```

Or, alternatively:

```
    #ifdef ATTR_MAC_MIN_SIZE
        xvt_vobj_set_attr(NULL_WIN, ATTR_MAC_MIN_SIZE
100L);
     #endif
```

See Also

XVT Portable Attributes
ATTR_ERRMSG_HANDLER
ATTR_EVENT_HOOK
NULL_WIN
WINDOW
XVTWS, *WS Values
xvt_font_map_using_default
xvt_vobj_get_attr

For information on the non-portable ATTR_* Constants, see the *XVT Platform-Specific Books*

Example

See the example for xvt_font_map_using_default.

xvt_vobj_set_data

Associate Application Data with Window

Summary

```
void xvt_vobj_set_data(WINDOW win, long data)
```

WINDOW win

Window, dialog, control, or pixmap whose data is to be set.

long data

Data to associate with the window.

Description

Every XVT WINDOW (regular, task, screen, print, dialog, control, or pixmap) has an associated long int for your application's own use. This function sets the value of that data word for win. You might use this long word to store a pointer to window state data containing information about the status of objects you are displaying in that window. For example, it could contain information needed for redrawing the client area of the window during an E_UPDATE event.

In addition to xvt_vobj_set_data, these creation functions associate application data with a WINDOW:

```
xvt_ctl_create
xvt_ctl_create_def
xvt_dlg_create_def
xvt_dlg_create_res
xvt_win_create
xvt_win_create_def
xvt_win_create_res
```

In contrast to the creation functions, you can only use `xvt_vobj_set_data` to re-associate application data once the `WINDOW` has been created.

After application data has been set, your application can retrieve it by calling `xvt_vobj_get_data` with the appropriate `WINDOW`.

See Also

```
WINDOW
xvt_ctl_create
xvt_ctl_create_def
xvt_dlg_create_def
xvt_dlg_create_res
xvt_tx_create_def
xvt_vobj_get_data
xvt_win_create
xvt_win_create_def
xvt_win_create_res
```

Example

This code illustrates the use of `xvt_vobj_set_data` and `xvt_vobj_get_data` for accessing application data associated with a window:

```
    typedef struct s_state {
         int contents;
        int source;
        int index;
        char str[100];
        int count;
        } State;/* Application state data for window */
     ...
    long XVT_CALLCONV1 win101_eh(WINDOW xdWindow,
         EVENT *xdEvent)
    {
         State *state;
        switch (xdEvent->type) {
        case E_CREATE:
             state = (State *)
xvt_mem_zalloc(sizeof(State));
            xvt_vobj_set_data(xdWindow, PTR_LONG(state));
            break;
            ...
         case E_UPDATE:
             state = (State *)
xvt_vobj_get_data(xdWindow);
            ...
         }
     }
```

xvt_vobj_set_enabled

Enable or Disable Window

Summary

```
void xvt_vobj_set_enabled(WINDOW win, BOOLEAN enabled)
```

`WINDOW win`

Window, dialog, or control to be enabled or disabled.

`BOOLEAN enabled`

Determines whether to enable or disable the window.

Description

This function enables or disables a window, dialog, or a control. If a window is disabled, its event handler cannot receive focus, character, or mouse events. If a dialog is disabled, its events handler cannot receive focus or character events. If a control is disabled, it cannot be operated by the user and does not generate `E_CONTROL` events.

Disabling a window or dialog also disables all controls (and child windows in the case of a window) contained in that `WINDOW`. Note that it might not be possible, or desirable, to disable modal dialogs or `TASK_WIN`s.

Parameter Validity and Error Conditions

XVT issues an error if the following conditions are not met:

- `win` must be a valid window, dialog, or conrol
- You must not call this function during an `E_UPDATE` event

See Also

```
CTL_FLAG_* Options
DLG_FLAG_* Options
E_CONTROL
E_UPDATE
TASK_WIN
WINDOW
WSF_* Options Flags
xvt_ctl_create
xvt_ctl_create_def
xvt_dlg_create_def
xvt_dlg_create_res
xvt_win_create
xvt_win_create_def
xvt_win_create_res
```

xvt_vobj_set_formatter

Set the Format Callback Function for a Window, Dialog, or Control

Summary

```
void xvt_vobj_set_formatter(WINDOW win,
XVT_FORMAT_HANDLER handler, void *data)
```

`WINDOW win`

XVT window, dialog, or control handle.

`XVT_FORMAT_HANDLER handler`

Application-defined string format handler callback function.

`void *data`

Application-defined data passed through to the format handler callback function.

Description

This function attaches an application-defined string format callback function and application-defined function data to the specified WINDOW object.

Parameter and Validity Conditions

XVT issues an error if any of the following conditions occur:

- If win or handler is NULL.
- If win is not a valid XVT WINDOW.

See Also

```
XVT_FORMAT_HANDLER
XVT_PATTERN
xvt_pattern_create
xvt_pattern_destroy
xvt_pattern_match
xvt_vobj_get_formatter
```

xvt_vobj_set_palet

Set a Visible Object's Palette

Summary

```
void xvt_vobj_set_palet(WINDOW win, XVT_PALETTE palet)
```

`WINDOW win`

Visible object for which the palette is to be set.

`XVT_PALETTE palet`

Palette to associate with the object.

Description

This function associates a color palette with a visible object, overriding any palette previously associated with the window.

You can only set the palette on the top-level windows and pixmaps. Child windows inherit their parent's palette.

Return Value

None.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `palet` is `NULL` or invalid
- `win` is not a pixmap or a top-level window
- You call this function during an `E_UPDATE` event

See Also

```
E_UPDATE
WINDOW
XVT_PALETTE
xvt_palet_create
xvt_vobj_get_palet
```

Example

See the example for `xvt_palet_create`.

xvt_vobj_set_title

Set Title of Window or Control

Summary

```
void xvt_vobj_set_title(WINDOW win, char *title)
```

`WINDOW win`

Window, dialog or control whose title is to be set.

`char *title`

Title to be set.

Description

This function changes the title of a control, top-level window, or dialog to the `NULL`-terminated string pointed to by `title`.

Setting the title of a control really means setting the appropriate control-specific text. The effect of setting the text of a control varies with control type as follows:

- For `WC_EDIT` or `WC_LISTEDIT`, the content of the edit field is set.
- For `WC_GROUPBOX`, `WC_TEXT`, `WC_PUSHBUTTON`, `WC_RADIOBUTTON`, or `WC_CHECKBOX`, the control's label is

changed. (This function can also change the mnemonic characters of these controls.)

- For other controls, this function has no effect.

If an application-defined format function is attached to the WINDOW object, xvt_vobj_set_title may result in setting the title to a different string, as specified by the format handler function, or may cause xvt_vobj_set_title to return without taking any action.

This function cannot be used to change the text of a text edit object; you must use the `xvt_tx_*` functions instead.

Note: If you want the title of a top-level window to obey the user interface style guidelines for each platform you're planning to run on, use the similar function `xvt_win_set_doc_title` instead.

See Also

```
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
xvt_vobj_get_title
xvt_tx_*
xvt_win_get_ctl
xvt_win_set_doc_title
```

xvt_vobj_set_visible

Show or Hide Window or Control

Summary

```
void xvt_vobj_set_visible(WINDOW win, BOOLEAN show)
```

`WINDOW win`

Window, dialog, or control to be shown or hidden.

`BOOLEAN show`

Determines whether to show or hide the window.

Description

This function shows or hides the window, dialog, or control specified by `win`. A hidden control, dialog, or window does not appear on the screen and can't receive user input. `win` is made visible if `show` is `TRUE`, or invisible otherwise.

Parameter Validity and Error Conditions

Regular windows, dialogs, and controls can be hidden (or reshown), but print windows, `TASK_WIN`s, `SCREEN_WIN`s, and `XVT_PIXMAP`s cannot be hidden (and reshown). Hiding a modal dialog is not allowed.

See Also

```
CTL_FLAG_* Options
DLG_FLAG_* Options
SCREEN_WIN
TASK_WIN
WINDOW
WSF_* Options Flags
XVT_PIXMAP
xvt_vobj_set_enabled
xvt_win_create
```

The "Windows" the and "Dialogs" chapters in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_win_create`.

xvt_vobj_translate_points

Translate Window Coordinates

Summary

```
void xvt_vobj_translate_points(WINDOW from_win,
     WINDOW to_win, PNT *pntp, int npnts)
```

`WINDOW from_win`

Source window.

`WINDOW to_win`

Destination window.

`PNT *pntp`

Array of points.

`int npnts`

Number of points, in`pntp`.

Description

This function translates npnts points in the pntp array from being relative to from_win to being relative to to_win. (It performs an in-memory replacement of the values in the PNT array.)

In addition to any regular window, dialog window, or control, you can use TASK_WIN and SCREEN_WIN. However, print windows and XVT_PIXMAPs are not allowed.

xvt_vobj_translate_points is especially useful when arranging windows to fit in their parent's client area.

Parameter Validity And Error Conditions

XVT issues an error if any of the following parameter conditions are not met:

- from_win and to_win must be valid WINDOWs and cannot be print windows or XVT_PIXMAPs
- pntp must be non-NULL
- npnts must be positive

See Also

PNT
SCREEN_WIN
TASK_WIN
WINDOW
xvt_vobj_get_client_rect
xvt_vobj_move

Example

See the example for xvt_vobj_move.

xvt_win_*

Window Functions

xvt_win_create
xvt_win_create_def
xvt_win_create_res
xvt_win_dispatch_event
xvt_win_enum_wins
xvt_win_get_ctl
xvt_win_get_ctl_color_component
xvt_win_get_ctl_colors
xvt_win_get_ctl_font
xvt_win_get_cursor
xvt_win_get_cxo
xvt_win_get_event_mask
xvt_win_get_handler
xvt_win_get_nav
xvt_win_get_tx
xvt_win_has_menu
xvt_win_list_cxos
xvt_win_list_wins
xvt_win_process_modal
xvt_win_release_pointer
xvt_win_set_caret_pos
xvt_win_set_caret_size
xvt_win_set_caret_visible
xvt_win_set_ctl_color_component
xvt_win_set_ctl_colors
xvt_win_set_ctl_font
xvt_win_set_cursor
xvt_win_set_doc_title
xvt_win_set_event_mask
xvt_win_set_handler
xvt_win_trap_pointer
xvt_win_unset_ctl_color_component

xvt_win_create

Create a Window

Summary

```
WINDOW xvt_win_create(WIN_TYPE wtype, RCT *rct_p,
     char *title, int menu_rid, WINDOW parent_win,
    long win_flags, EVENT_MASK mask, EVENT_HANDLER eh,
    long app_data)
```

`WIN_TYPE wtype`

Indicates which type of window is to be created. It should be set to `W_MODAL`, `W_DOC`, `W_DBL`, `W_NO_BORDER`, or `W_PLAIN`. The type of window you choose depends on the border style you prefer, and on whether the window is a top-level or child window. If you want to create a top-level window, you must set `wtype` to `W_MODAL`, `W_DOC`, `W_DBL`, or `W_PLAIN`. If you want to create a child window, you must set `wtype` to `W_PLAIN` or `W_NO_BORDER`.

`RCT *rct_p`

Points to a rectangle specifying the window's client area in its parent's coordinate system. The rectangle must be in pixels. `rct_p` can be specified as `XVT_MAX_WINDOW_RECT`, asking for a window as large as the parent window permits.

`char *title`

Points to a single-byte or multibyte string for the title of a window. When creating `W_NO_BORDER` windows, this can be set to `NULL`.

`int menu_rid`

Specifies a resource ID for a menubar resource as described in your URL file. This can be set to zero only if the `WSF_NO_MENUBAR` flag is set in the `win_flags` parameter, or the window being created is a child window.

`WINDOW parent_win`

Specifies the container in which the window is created. If you are creating a top-level window, set this to `TASK_WIN` or `SCREEN_WIN`. If you are creating a child window, then set this to an existing XVT window. This parameter cannot specify a dialog, control, pixmap, or print window.

`long win_flags`

Can contain an OR'd combination of `WSF_*` flags. These flags control the window's style and decoration. Certain flags are not valid for `W_PLAIN`, `W_DBL`, and `W_NO_BORDER`. For example, if `WSF_ICONIZED` is one of the flags, it should *not* be combined with `WSF_MAXIMIZED`, `WSF_INVISIBLE`, or `WSF_DISABLED`, but it *should* be combined with `WSF_ICONIZABLE`. For valid combinations of `WSF_*` flags, see `Window/Dialog/Control Creation Function Parameters`.

`EVENT_MASK mask`

Specifies which events should be sent to the window handler. This is an OR'd combination of any of the `EM_*` constants. You usually set this to `EM_ALL` indicating that all events should be sent to the window (no restriction). In some conditions, you can restrict the events sent to the window. For more details, see the "Event Masking" section of the "Events" chapter in the *XVT Portability Toolkit Guide*.

`EVENT_HANDLER eh`

The event handler function; it receives all of the events for the window.

`long app_data`

Contains any application data you wish to attach to the window when it is created. Normally, it is a pointer to a data structure cast into a `long`.

`long win_flags`

The flag WSF_DEFER_MODAL is only valid for W_MODAL.

Description

This function creates a top-level or child XVT window.

Return Value

A `WINDOW` if successful; `NULL_WIN` if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are not met:

- If `wtype` is `W_MODAL`, the `parent_win` parameter must be `TASK_WIN`, `SCREEN_WIN`, `W_MODAL`, `WD_MODAL`, `WD_MODELESS`, `W_DOC`, `W_PLAIN`, or `W_DBL`.
- `parent_win` must be a valid window

- If parent_win is TASK_WIN or SCREEN_WIN (implying a top-level window), then the wtype parameter must be W_DOC, W_PLAIN, W_DBL or W_MODAL
- If parent_win is not TASK_WIN or SCREEN_WIN (implying a child window), then the wtype parameter must be W_PLAIN, W_NO_BORDER, or W_MODAL
- eh must be a vaild pointer to a window event handler
- rct_p must either be XVT_MAX_WINDOW or a valid RCT pointer
- If you are creating a top-level window and WSF_NO_MENUBAR is not set, then menu_rid must be a valid menu resource ID; otherwise, menu_rid can be set to zero
- If WSF_ICONIZED is one of the flags in the win_flags parameter, then it cannot be combined with WSF_MAXIMIZED, WSF_INVISIBLE, or WSF_DISABLED; for other specifications of appropriate combinations of WSF_* flags, see Window/Dialog/Control Creation Function Parameters
- You must *not* call this function during an E_UPDATE event

See Also

EM_* Constants
E_UPDATE
EVENT_HANDLER
EVENT_MASK
NULL_WIN
RCT
SCREEN_WIN
TASK_WIN
W_*, WC_*, WD_*, Values for WIN_TYPE
WIN_TYPE
WINDOW
WSF_* Options Flags
XVT_MAX_WINDOW_RECT
xvt_vobj_get_data
xvt_vobj_set_data
xvt_win_create_def
xvt_win_process_modal
Window/Dialog/Control Creation Function Parameters

The "Creating Windows" section of the "Windows" chapter in the *XVT Portability Toolkit Guide*

Example

This code creates a plain window that is initially invisible, and then later displayed with xvt_vobj_set_visible:

```
WINDOW window;
RCT rct;
short width, height;
...
xvt_rect_set(&rct, 50, 50, 50 + width, 50 + height);
window = xvt_win_create(W_PLAIN, &rct, "New Window",
     0, TASK_WIN, WSF_NO_MENUBAR|WSF_INVISIBLE,
    EM_UPDATE|EM_CLOSE, window_eh, 0L);
 ...
/* show window after it is initialized */
xvt_vobj_set_visible(window, TRUE);
```

xvt_win_create_def

Create a Window with Controls from an Array of Data Structures

Summary

```
WINDOW xvt_win_create_def(WIN_DEF *win_def_p,
     WINDOW parent_win, EVENT_MASK mask,
    EVENT_HANDLER eh, long app_data)
```

`WIN_DEF *win_def_p`

Points to an array of data structures. The first element in the array defines the window itself. Subsequent elements of the array define the controls or text edit objects contained within the window. The last element of the array is a terminator whose `wtype` field is set to `W_NONE`.

`WINDOW parent_win`

Specifies the container in which the window is to be created. If you are creating a top-level window (types `W_DOC`, `W_MODAL`, or `W_PLAIN`) this parameter must be set to a `TASK_WIN` or a `SCREEN_WIN`. If you are creating a child window (types `W_PALIN` or `W_NO_BORDER`), then set this parameter to an existing XVT window. This parameter cannot specify a dialog, control, pixmap, or print window.

`EVENT_MASK mask`

Specifies which events are sent to the window event handler. This is an OR'd combination of any of the `EM_*` constants. You usually set this to `EM_ALL` indicating that all events would be sent to the window. For more details, see the "Event Masking" section of the "Events" chapter in the *XVT Portability Toolkit Guide*.

`EVENT_HANDLER eh`

The event handler function; it receives all of the events for the window.

`long app_data`

Contains any application data you wish to attach to the window when it is created. Normally, it is a pointer to a data structure cast into a `long`.

`long win_flags`

The flag WSF_DEFER_MODAL is only valid for W_MODAL.

Description

This function creates a window and its controls based on a description contained in an array of `WIN_DEF` data structures. When filling in the `WIN_DEF` structures that define the window, keep in mind the following:

- `win_def_p[0].wtype` should be set to `W_DOC`, `W_DBL`, `W_MODAL`, `W_NO_BORDER`, or `W_PLAIN`, to indicate the type of window you want to create. The type of window you choose depends on the border style you prefer and on whether the window is a top-level or child window. If you want to create a top-level window, you must set `wtype` member to `W_DOC`, `W_DBL`, `W_PLAIN`, or `W_MODAL`. If you want to create a child window, you must set the `wtype` member to `W_PLAIN` or `W_NO_BORDER`.
- `win_def_p[0].rct` should be set to a rectangle specifying the window's client area in its parent's coordinate system.
- `win_def_p[0].text` should point to a string containing the title of the window. If the window is of type `W_NO_BORDER`, `win_def_p[0].text` can be set to `NULL`.
- `win_def_p[0].units` should be set to `U_PIXELS`, `U_CHARS`, or `U_SEMICHARS` to indicate which type of coordinate system XVT will use to place the window.
- `win_def_p->ctlcolors` contains the array of `XVT_COLOR_COMPONENT` structures that define the colors of the controls in a window. If it is `NULL`, the controls in the window will use the default application control colors. The last element of the `XVT_COLOR_COMPONENT` array must have an `XVT_COLOR_TYPE` of `XVT_COLOR_NULL` to indicate the end of the array.

- win_def_p[0].v.win.menu_rid *or* win_def_p[0].v.win.menu_p should be set for any top-level window being created that does not have the WSF_NO_MENUBAR flag set. If win_def_p[0].menu_rid is set (non-zero), then it must be a resource ID for a menubar resource as described in your URL file. If win_def_p[0].menu_p is set (non-NULL), then it must point to a valid array of MENU_ITEMs. The format of this array is the same as if you were making a call to xvt_menu_set_tree.
- win_def_p[0].v.win.flags can contain an OR'd combination of WSF_*. These flags control the window's style and decoration. Certain flags are not valid for W_PLAIN, W_DBL, and W_NO_BORDER. For valid combinations of WFS_* flags, see Window/Dialog/Control Creation Function Parameters.
- win_def_p[0].v.win.ctl_font_id is the XVT_FNTID that defines the font used in all the controls in the window.
- win_def_p[1...n] (where n is the number of controls or text edit objects to be created in the window) should be filled with valid descriptions for either a control or a text edit object. A valid description for a control is one that would be correct if passed to xvt_ctl_create_def. A valid description for a text edit object is one that would be correct if passed to xvt_tx_create_def.
- win_def_p[n+1].wtype should be set to W_NONE indicating the end of the array.

Return Value

A WINDOW if successful; NULL_WIN if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are not met:

- win_def_p must point to a valid array of WIN_DEF structures.
- win_def_p[0].wtype must be one of W_DOC, W_DBL, W_NO_BORDER, W_MODAL, or W_PLAIN. W_DOC and W_DBL are allowed only for top-level windows. W_NO_BORDER is only valid for child windows.
- win_def_p[i].ctlcolors must be either NULL or a valid array of XVT_COLOR_COMPONENT structures.
- win_def_p[0].v.win.ctl_font_id must be either NULL_FNTID or a valid logical font.

- `win_def_p_[i].v.ctl.font_id` must be either `NULL_FNTID` or a valid logical font.
- `win_def_p[0].text` must point to a valid string if you are creating a top-level window. If you are creating a child window of type `W_NO_BORDER`, this field can be `NULL`.
- `win_def_p[0].units` must be one of `U_PIXELS`, `U_CHARS`, or `U_SEMICHARS`.
- `win_def_p[0].v.win.flags` must contain valid flags for the window to be created. If `WSF_ICONIZED` is one of the flags in the `win_flags` parameter, then it cannot be combined with `WSF_MAXIMIZED`, `WSF_INVISIBLE`, or `WSF_DISABLED`. However, if `WSF_ICONIZED` is one of the flags, `WSF_ICONIZABLE` must also be used. For other specifications of appropriate combinations of `WSF_*` flags, see `Window/Dialog/Control Creation Function Parameters`.
- All top-level windows being created that do not have the `WSF_NO_MENUBAR` flag set must have a valid menu specification. A valid menu specification means that exactly one of the fields `win_def_p[0].v.win.menu_rid` *or* `win_def_p[0].v.win.menu_p` is set to a valid menu specification. If `win_def_p[0].v.win.menu_rid` is set, then it must be a resource ID for a menubar resource as described in your URL file. If `win_def_p[0].v.win.menu_p` is set, then it must point to a valid array of `MENU_ITEM`s.
- `win_def_p[1...n]` (where `n` is the number of controls or text edit objects to be created in the window) must be filled with valid descriptions for either a control or a text edit object. A valid description for a control is one that would be correct if passed to `xvt_ctl_create_def`. A valid description for a text edit object is one that would be correct if passed to `xvt_tx_create_def`.
- `win_def_p[n+1]` must have its `wtype` field set to `W_NONE` to signal termination of the array.

See Also

```
EVENT_HANDLER
EVENT_MASK
MENU_ITEM
NULL_WIN
U_* Values for UNIT_TYPE
W_*, WC_*, WD_*, Values for WIN_TYPE
WIN_DEF
WINDOW
WSF_* Options Flags
XVT_COLOR_COMPONENT
XVT_COLOR_*
XVT_COLOR_TYPE
XVT_FNTID
xvt_ctl_create_def
xvt_menu_set_tree
xvt_tx_create
xvt_vobj_get_data
xvt_vobj_set_data
xvt_win_create
xvt_win_create_res
xvt_win_process_modal
Window/Dialog/Control Creation Function Parameters
```

The "Event Masking" section of the "Events" chapter in the *XVT Portability Toolkit Guide*

xvt_win_create_res

Create a Window from a Resource File

Summary

```
WINDOW xvt_win_create_res(int rid, WINDOW parent_win,
     EVENT_MASK mask, EVENT_HANDLER eh, long app_data)
int rid
```

Resource ID corresponding to a window statement in your URL resource file. The window is created as if this resource were loaded via `xvt_res_get_win_def`, and then instantiated via `xvt_win_create_def`.

```
WINDOW parent_win
```

Specifies the container in which the new window is to be created. It can be `TASK_WIN`, `SCREEN_WIN`, or any XVT window you have previously created. It cannot be a dialog, control, pixmap, or print window.

```
EVENT_MASK mask
```

Specifies which events should be sent to your window's event handler. Normally, you would set this to `EM_ALL`. To restrict the events set to the handler to a subset of the possible events, set `mask` to an OR'd combination of the `EM_*` constants.

`EVENT_HANDLER eh`

Event handler function; it receives all of the events for the window.

`long app_data`

Can be used to attach arbitrary application data to your window. This is normally set to a pointer to a data structure containing the information for your window, cast into a `long`.

Description

This function creates a window based on an URL window definition.

Return Value

A `WINDOW` if successful; `NULL_WIN` if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are not met:

- `parent_win` must be a valid XVT window. It cannot be a print window, dialog, or control.
- `eh` must be set to a valid function pointer.
- `rid` specifies a window resource in the URL file. All of the parameter checking and validity conditions that apply to `xvt_win_create_def` also apply to the window resource as specified in URL. Note that **curl** does not necessarily catch all parameter validity problems.

Note: `rid` must be a valid window resource specified in the URL file or by some non-portable means.

See Also

```
EM_* Constants
EVENT_HANDLER
EVENT_MASK
NULL_WIN
SCREEN_WIN
TASK_WIN
WINDOW
xvt_ctl_create_def
xvt_res_get_win_def
xvt_tx_create_def
xvt_vobj_get_data
xvt_vobj_set_data
xvt_win_create
xvt_win_create_def
```

The "Creating Windows" section of the "Windows" chapter in the *XVT Portability Toolkit Guide*

xvt_win_dispatch_event

Send an Event to a Window

Summary

```
long xvt_win_dispatch_event(WINDOW win, EVENT *event_p)
```

`WINDOW win`

Window or dialog to which an event is being sent.

`EVENT *event_p`

Event to be sent.

Description

This function allows you to send events to `win`'s event handler. Note that the event is not queued on the native system's event queue. The most common use for `xvt_win_dispatch_event` is sending `E_USER` events to your application's event handlers.

Return Value

The return value is that of the event handler invoked by this call. In this way, an event handler can return values to your application. If the event handler for `win` has been masked for the event type of `event_p`, this function returns -1.

Parameter Validity and Error Conditions

XVT issues an error if:

- The specified window is not a valid XVT `WINDOW` as created by `xvt_win_create_*`, `xvt_dlg_create_*`, or `TASK_WIN`
- The event structure's `type` field is not set to a valid event type
- The window parameter is `SCREEN_WIN`
- The window does not have an event handler

See Also

`E_USER`
`EVENT`
`WINDOW`

xvt_win_enum_wins

List Windows and Controls

Summary

```
BOOLEAN xvt_win_enum_wins (WINDOW parent_win,
     XVT_ENUM_CHILDREN func, long data,
    unsigned long reserved)
WINDOW parent_win
```

Parent window whose children will be enumerated.

`XVT_ENUM_CHILDREN func`

Address of function to be called for each child.

`long data`

Application-defined data to pass to callback function.

`unsigned long reserved`

Reserved for future use.

Description

This function enumerates (i.e., invokes an application-supplied callback function) the child windows and controls (excluding text edits) contained in the specified parent window. It passes the `WINDOW` ID of each child window or control, in creation order, to an application-defined callback function. It continues until the last child window and control is enumerated or until the callback function returns `FALSE`.

This function does not enumerate a child window or control that is destroyed during a call to this function before the child window or control are enumerated. Also any child window or control created in the `parent_win` will not be enumerated. These measures ensure that `xvt_win_enum_wins` is reliable even when the application causes odd side effects.

Return Value

`TRUE` if successful; `FALSE` otherwise.

Parameter Validity and Error Conditions

XVT issues an error if:

- `parent_win` is invalid
- `reserved` is not `NULL`

See Also

`WINDOW`
`XVT_ENUM_CHILDREN`
`xvt_win_list_wins`

xvt_win_get_ctl

Retrieve Control Window Based on ID

Summary

`WINDOW xvt_win_get_ctl(WINDOW parent_win, int cid)WINDOW parent_win`

Window for the container.

`int cid`

Control ID for the control.

Description

This function returns the `WINDOW` for a control given its parent `WINDOW` and control ID. This function can only be used on controls that were given a unique ID at creation time.

An alternative to calling `xvt_win_get_ctl` is to store the `WINDOW` returned by `xvt_ctl_create` or `xvt_ctl_create_def`.

Return Value

The WINDOW of the control if successful, or NULL_WIN if no such control exists.

Parameter Validity and Error Conditions

XVT issues an error if parent_win is not of wtype W_* or WD_*, or is a screen or print window.

Implementation Note

On XVT/Win32, parent_win can be TASK_WIN if you have previously set the attribute ATTR_WIN_PM_DRAW_TWIN.

See Also

NULL_WIN
SCREEN_WIN
TASK_WIN
WINDOW
W_*, WC_*, WD_*, Values for WIN_TYPE
xvt_ctl_create
xvt_ctl_create_def

Example

This code gets pointers to windows using xvt_win_get_ctl:

```
WINDOW xdWindow;
WINDOW family_lbox;
WINDOW size_lbox;
WINDOW style_lbox;
...
family_lbox = xvt_win_get_ctl(xdWindow, FAMILY_LBOX);
size_lbox = xvt_win_get_ctl(xdWindow, SIZE_LBOX);
style_lbox = xvt_win_get_ctl(xdWindow, STYLE_LBOX);
```

xvt_win_get_ctl_color_component

Gets the Control Color for a Color Type From a Single Control

Summary

```
COLOR xvt_win_get_ctl_color_component(WINDOW win,
XVT_COLOR_TYPE ctype)
```

WINDOW win

WINDOW ID of a window or dialog.

XVT_COLOR_TYPE ctype

Control component to get the color for.

Description

This function gets the control color for a specific control component from the defined container control colors. The control color returned reflects the latest container-level control color setting. This color is either the control color set in the WIN_DEF structure during the window's creation, or the control colors set by xvt_win_set_ctl_colors or xvt_win_set_ctl_color_component.

Return Value

INVALID_COLOR if no color is set for the requested component of this window or if an error occurs, otherwise the COLOR value of the control component.

Parameter and Validity Conditions

XVT issues an error if win is not a valid WINDOW or ctype is not a valid XVT_COLOR_TYPE.

See Also

```
WIN_DEF
XVT_COLOR_TYPE
XVT_COLOR_COMPONENT
xvt_ctl_set_colors
xvt_ctl_set_color_component
xvt_ctl_unset_color_component
xvt_ctl_get_colors
xvt_win_set_ctl_colors
xvt_win_set_ctl_color_component
xvt_win_unset_ctl_color_component
xvt_win_get_ctl_colors
ATTR_APP_CTL_COLORS
COLOR
```

xvt_win_get_ctl_colors

Get Default Container Control Colors

Summary

```
XVT_COLOR_COMPONENT *xvt_win_get_ctl_colors(WINDOW win)

WINDOW win
```

WINDOW ID of window or dialog.

Description

This function provides the application with a copy of the defined container control colors. The control colors returned reflect the latest container-level control colors setting. These colors can either be the container control colors set in the `WIN_DEF` structure during the window's creation, or the container-level control colors set by `xvt_win_set_ctl_colors`. The returned colors are those that render controls in the container for which no control-specific colors have been defined.

Return Value

A pointer to an array of `XVT_COLOR_COMPONENT` structures; `NULL` if an error occurs or if no colora have been set. The application owns this array and must destroy it when finished with it.

Parameter Validity and Error Conditions

XVT issues an error if:

- `win` is not valid
- `win` is not type `W_DOC`, `W_PLAIN`, `W_DBL`, `W_NO_BORDER`, `W_MODAL`, `WD_MODAL`, or `WD_MODELSS`

See Also

```
COLOR
COLOR_*, COLOR_INVALID Constants
NULL
WIN_DEF
WINDOW
XVT_COLOR_*
XVT_COLOR_COMPONENT
xvt_ctl_get_colors
xvt_win_set_ctl_colors
```

Example

```
XVT_COLOR_COMPONENT* colors;
COLOR bkgnd = COLOR_BLACK;
int = icolors = xvt_win_get_ctl_colors(win);
if(NULL == colors) return;
for(i=0; colors[i].type! = XVT_COLOR_NULL {
     if(colors[i].type ==
            XVT_COLOR_BACKGROUND) {
         bkgnd = colors[i].color;
        break;
     }
 }
xvt_mem_free(colors);
```

xvt_win_get_ctl_font

Get Logical Font of Controls

Summary

```
XVT_FNTID xvt_win_get_ctl_font(WINDOW win)
```

`WINDOW win`

`WINDOW ID` of window or dialog.

Description

This function returns a copy of the logical font used by all of the controls in a container for which no individual control fonts have been established. The application must delete the logical font when finished with it.

The logical font returned reflects the latest container level control font setting. This font can either be the container control font set in the `WIN_DEF` structure during the window's creation, or a container level control font set by `xvt_win_set_ctl_font`.

Return Value

A copy of the logical font used to render all of the controls in the container for which no individual control fonts have been established; `NULL_FNTID` if an error occurs or if no font was set. The application owns this logical font and must destroy it when finished with it.

Parameter Validity and Error Conditions

XVT issues an error if:

- `win` is not valid
- `win` must be of type `W_DOC`, `W_PLAIN`, `W_NO_BORDER`, `W_MODAL`, `WD_MODAL`, or `WD_MODELSS`

See Also

```
XVT_FNTID
NULL_FNTID
WIN_DEF
WINDOW
xvt_ctl_get_font
xvt_win_set_ctl_font
```

xvt_win_get_cursor

Get the Cursor Shape

Summary

```
CURSOR xvt_win_get_cursor(WINDOW win)
```

`WINDOW win`

Window whose cursor shape is being inquired.

Description

This function gets the `CURSOR` that identifies the symbol used to locate the mouse pointer or cursor. Recall that every XVT regular window has a cursor shape associated with it, and XVT automatically changes the shape of the cursor as the user moves it from window to window. Therefore, this function returns the cursor shape associated with a particular window. You can set the standard XVT cursor with a call to `xvt_win_set_cursor`. For a list of the standard XVT cursors, see `CURSOR_* Options`.

The main reason for calling `xvt_win_get_cursor` is to save the `CURSOR` for later restoration with `xvt_win_set_cursor`. The value returned will be the value last set by a call to `xvt_win_set_cursor`. If no previous call to `xvt_win_set_cursor` for that window has been made, then the `CURSOR_ARROW` cursor is returned.

Return Value

The current `CURSOR` associated with `win` if successful; `CURSOR_ARROW` if no previous call to `xvt_win_set_cursor` was made.

Parameter Validity and Error Conditions

XVT issues an error if `win` is not a valid `WINDOW` of type `W_*`. Dialogs, controls, pixmaps, and print windows are not valid values for `win`.

See Also

```
CURSOR
CURSOR_* Options
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
xvt_win_set_cursor
xvt_win_trap_pointer
```

Example

See the example for `xvt_win_trap_pointer`.

xvt_win_get_cxo

Retrieve a CXO

Summary

```
XVT_CXO xvt_win_get_cxo( WINDOW win, char * class_name,
long cxo_id )
```

`WINDOW win,`

Window whose CXO of class name and id are to be retrieved.

`char * class_name,`

The class name of the CXO to be found.

`long cxo_id`

The ID of the CXO of class name to be found. This value can be zero.

Description

This function retrieves a CXO from a container's CXO list using a class name and, if supplied, a `cxo_id`. When created using `xvt_cxo_create`, CXO's minimally need a class name so that they can be retrieved. If more than one CXO of a class can be contained in a window, then id values are also needed. If no unique ID's are used with `xvt_win_get_cxo`, then the first CXO of `class_name` will be returned.

Return Value

A valid `XVT_CXO`; `NULL` if unsuccessful or on error.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are not met:

- `win` must be valid.
- The `class_name` must be valid.

See Also

```
WINDOW
XVT_CXO
xvt_cxo_create
```

Example

See example for `xvt_cxo_destroy`.

xvt_win_get_event_mask

Get Event Mask for Window

Summary

```
EVENT_MASK xvt_win_get_event_mask(WINDOW win)
```

`WINDOW win`

Window, dialog, or task window whose event mask is to be gotten.

Description

This function returns the `EVENT_MASK` for a window, dialog, or task window. Controls and screen windows are not allowed. Recall that the `EVENT_MASK` specifies the events the event handler for a window can receive, and is set by ORing together the `EM_*` constants, and then calling `xvt_win_set_event_mask` or the appropriate `WINDOW` creation function.

Return Value

The `EVENT_MASK` for the `WINDOW`.

Parameter Validity And Error Conditions

XVT issues an error if `win` is not a valid window or dialog of type `W_*` or `WD_*`. `win` cannot be a screen window, control, pixmap, or print window.

See Also

```
EM_* Constants
EVENT_MASK
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
xvt_timer_destroy
xvt_win_set_event_mask
```

Example

See the example for `xvt_timer_destroy`.

xvt_win_get_handler

Retrieve Event Handling Function for Window

Summary

```
EVENT_HANDLER xvt_win_get_handler(WINDOW win)
```

`WINDOW win`

Window, dialog, or task window whose event handler is to be gotten.

Description

This function gets the current event handler for a window, dialog, or task window. The following are typical uses for `xvt_win_get_handler`:

- You can determine a `WINDOW`'s "class" by comparing the value returned by `xvt_win_get_handler` with the known event handlers used by your application
- You can temporarily save the current event handler for a window or dialog and restore it later with a call to `xvt_win_set_handler`; you may want to do this if you temporarily override the event handler for a window or dialog and wish to put it back later
- You can save the event handler for a window or dialog, reassign a new handler, and "preprocess" events with the new handler before it invokes the original handler, effectively chaining together event handlers

Return Value

The `EVENT_HANDLER` registered to the `WINDOW`.

Parameter Validity and Error Conditions

The `WINDOW` must be of `WIN_TYPE`, `W_*`, or `WD_*`. Screen windows, controls, pixmaps, and print windows are not allowed.

See Also

XVT Events
EVENT
EVENT_HANDLER
XVT_CALLCONV*
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
WIN_TYPE
xvt_win_set_handler

Example

This code uses `xvt_win_get_handler` to save a window's event handler so that it can be overridden by another event handler that preprocesses the window's events. The preprocessor event handler then invokes the original event handler when it is finished.

```
/* Save the original window event handler */
EVENT_HANDLER save_eh = xvt_win_get_handler(win);
xvt_win_set_handler(win, preprocess_eh);

...

/* Preprocess window events, then invoke original event
handler */

long XVT_CALLCONV1 preprocess_eh(WINDOW win, EVENT *ep) {

    switch(ep->type) {
    case E_CREATE:
    ...
    break;
    case E_FOCUS
    ...
    break;
    ...
    }
    return (*save_eh)(win, ep);

}
```

xvt_win_get_nav

Retrieves the Navigation Object Associated with a Window

Summary

```
XVT_NAV xvt_win_get_nav(WINDOW win);

WINDOW win
```

The window from which to obtain the navigation object.

Description

`xvt_win_get_nav` returns a navigation object from the specified window if one exists.

Return Value

A valid `XVT_NAV` or `NULL` if no navigation object is associated with the window.

Parameter Validity and Error Conditions

XVT issues an error if your application does not meet the following conditions for the parameter passed to `xvt_win_get_nav`:

- `win` is a valid window
- `win` is of type `W_DOC`, `W_PLAIN`, `W_DBL`, `W_MODAL`, or `W_NO_BORDER`

See Also

`W_*, WC_*, WD_*, Values for WIN_TYPE`
`WINDOW`
`XVT_NAV`
`xvt_nav_create`

---

xvt_win_get_tx

Get Text Edit Object from ID

Summary

```
TXEDIT xvt_win_get_tx(WINDOW parent_win, int cid)WINDOW parent_win
```

Window containing the text edit object to be retrieved.

```
int cid
```

Control ID of the text edit object.

Description

`xvt_win_get_tx` returns the text edit object whose control ID is `cid`, and which is contained in `parent_win`.

Note: You cannot assign an ID to a text edit object via `xvt_tx_create`. Instead, you must call `xvt_tx_create_def`. You can also call

xvt_win_create_def or xvt_win_create_res, which will in turn call the equivalent of xvt_tx_create_def.

Return Value

The TXEDIT corresponding to cid if successful; NULL_TXEDIT if no such text edit object exists.

Parameter Validity and Error Conditions

XVT issues an error if parent_win is not of wtype W_*. Pixmaps, print windows, and screen windows are not valid values for parent_win.

Implementation Note

On XVT/Win32, parent_win can be TASK_WIN if the attribute ATTR_WIN_PM_DRAW_TWIN has been set.

See Also

```
NULL_TXEDIT
TASK_WIN
TXEDIT
NULL_WIN
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
xvt_tx_create
xvt_tx_create_def
xvt_win_create_def
xvt_win_create_res
xvt_win_get_ctl
```

Example

This code gets pointers to a text edit object using xvt_win_get_tx:

```
WINDOW xdWindow;
TXEDIT tx;
...
tx = xvt_win_get_tx(xdWindow, WIN_PALETTE_TEDIT);
if (tx == NULL_TXEDIT)
     xvt_dm_post_error("Could not retrieve text edit");
 ...
```

xvt_win_has_menu

Determine if the Window has a Menubar

Summary

```
BOOLEAN xvt_win_has_menu(WINDOW win)
```

Description

This function determines if a window has a menubar. A window has a menubar if, at its creation time, the following are true:

- The call to `xvt_win_create` contains a menubar resource ID, *and* the window flags parameter does *not* contain `WSF_NO_MENUBAR`
- The call to `xvt_win_create_def` contains a `WIN_DEF` array whose first (zeroth) element does *not* use the `WSF_NO_MENUBAR` flag in the `v.win.flags` member, *and* either a valid `MENU_ITEM` pointer is in `v.win.menu_p` or a valid `MENU_BAR` resource ID is in `v.win.menu_rid`
- The URL `WINDOW` resource used in the call to `xvt_win_create_res` does not use the `NO_MENUBAR` flag, *and* it provides a valid a `MENU_BAR` resource ID

Return Value

`TRUE` if the window has a menubar; `FALSE` if the window does not have a menubar.

Parameter Validity and Error Conditions

XVT issues an error if `win` is not valid and of type `W_DOC`.

See Also

```
MENU_ITEM
WIN_DEF
WINDOW
WSF_* Options Flags
xvt_win_create
xvt_win_create_def
xvt_win_create_res
menu and menubar URL Statement
window URL Statement
```

The "Resources and URL" chapter in the *XVT Portability Toolkit Guide*

xvt_win_list_cxos

List All CXO's Associated With a Window

Summary

```
SLIST xvt_win_list_cxos( WINDOW win )
```

`WINDOW win,`

Window whose list of CXO's are to be retrieved.

Description

This function retrieves a list of all the CXO's contained within a window. The resulting list is returned in an `SLIST` with the class name set to the string portion of the `SLIST_ELT`. The application is responsible for destroying the `SLIST`.

Return Value

A valid `SLIST`; `NULL` if unsuccessful or if there are no CXO's in the window.

Parameter Validity and Error Conditions

XVT issues an error if the following condition is not met:

`win must be valid.`

See Also

`SLIST`
`WINDOW`

Example

This example shows how to retrieve a list of CXO's in a window, and retrieve a class name from the list.

```
SLIST slist;
SLIST_ELT elt;
char* class_name;

slist=xvt_win_list_cxos(win);
if( slist )
{
    for( elt = xvt_slist_get_first( slist ); elt != NULL;
    xvt_slist_get_next( slist, elt ) )
    {
    xvt_slist_get( slist, elt, class_name );
```

```
    /* Code to do something with class_name here */
    ...
    }
    xvt_slist_destroy( slist );

}
```

xvt_win_list_wins

List Titles

Summary

```
SLIST xvt_win_list_wins(WINDOW parent_win,
     unsigned long reserved)
WINDOW parent_win
```

Parent window of children returned in list.

`unsigned long reserved`

Reserved for future use.

Description

This function gets an `SLIST` containing `WINDOW` IDs and the titles of each control, window, and dialog which is parented by `parent_win` (excluding text edit objects, which have no `WINDOW` identifier). It is non-recursive, listing only the immediate descendants. The descendants are listed in creation order.

The data word in the `SLIST` associated with each descendent is its `WINDOW` object identifier. The data word may be used as an argument to `xvt_vobj_get_type` to determine the decendent's window type.

`TASK_WIN` and `SCREEN_WIN` are valid arguments to this function. When they are used as `parent_win`, the top-level windows and dialogs that are parented by them are returned.

This function is *not* a replacement for `xvt_scr_list_wins`. That function is useful for listing all top-level windows and dialogs.

Return Value

`SLIST` containing titles and Window IDs if successful; `NULL` on error. The application must free the returned `SLIST`, when no longer needed, using `xvt_slist_destroy`.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are not met:

- `reserved` must be `NULL`
- `parent_win` must be a valid window

See Also

```
SCREEN_WIN
SLIST
TASK_WIN
WINDOW
xvt_scr_list_wins
xvt_slist_destroy
xvt_vobj_get_type
xvt_win_enum_wins
```

xvt_win_process_modal

Sets the Control Color for a Color Type Used by a Single Control

Summary

```
void xvt_win_process_modal(WINDOW win)
```

`WINDOW win`

WINDOW ID of modal deferred W_MODAL window.

Description

This function enters a modal processing loop when passed a valid WINDOW of type W_MODAL that was created with a WSF_DEFER_MODAL flag. The function returns when the WINDOW is destroyed. Normally a W_MODAL window not created with WSF_DEFER_MODAL will enter the modal state inside the xvt_win_create* function and will only return after the window is destroyed and modal processing is complete. The function cannot be called twice with the same window or an error will occur. XVT recommends that you call xvt_vobj_is_valid on the window before calling xvt_win_process_modal as the window may have been destroyed in the window E_CREATE event.

Parameter and Validity Conditions

XVT issues an error if win is not a modal deferred W_MODAL WINDOW or if xvt_win_process_modal is called twice with the same win. XVT returns if win is already closed (invalid).

See Also

```
WINDOW
WIN_TYPE
WSF_* Options Flags
W_*, WC_*, WD_*, Values for WIN_TYPE
xvt_win_create
xvt_win_create_def
xvt_vobj_is_valid
```

xvt_win_release_pointer

Release Trapped Mouse

Summary

```
void xvt_win_release_pointer(void)
```

Description

This function releases the mouse pointer that was trapped with an earlier call to `xvt_win_trap_pointer`. Calling this function allows mouse events to be sent, once again, to all windows in the window system.

Parameter Validity and Error Conditions

If the window is invalid, XVT issues an error.

See Also

```
xvt_win_trap_pointer
```

Example

See the example for `xvt_win_trap_pointer`.

xvt_win_set_caret_pos

Reposition the Caret

Summary

```
void xvt_win_set_caret_pos(WINDOW win, PNT pos)
```

`WINDOW win`

Window whose caret is to be repositioned.

`PNT pos`

Position of caret (lower-left corner).

Description

This function repositions the caret. The caret doesn't move automatically when you draw text or process keyboard events; you must position the caret yourself by calling this function.

The `h` and `v` members of the `pos` parameter specify respectively the horizontal and vertical positions of the caret in the window's client area. If the caret size has not been set, or if the caret height has been explicitly set to zero, the bottom of the caret is placed as `pnt.v + size` of the window font descent. Otherwise, the bottom of the caret is exactly at `pnt.v`. The `pnt.h` horizontal component identifies the left coordinate of the caret. The caret's height is determined by the current logical font your application sets by calling `xvt_dwin_set_font`. Therefore, you should position the caret so that it lines up with the baseline of the text drawn next to the caret.

You can also position the caret so that the window's current logical font has no effect on the caret's height or position. For details, see `xvt_win_set_caret_size`. Typically, you must use `xvt_win_set_caret_size` if you are using multiple logical fonts within a single window, because, in this case, XVT cannot accurately determine what the current size should be.

Caution: Do not call `xvt_win_set_caret_pos` during an update event. Doing so interferes with the way the underlying window systems implement carets and causes XVT to issue an error.

Parameter Validity and Error Conditions

XVT issues an error if any of the following are true:

- You call this function during the processing of an `E_UPDATE` event
- `win` is `NULL_WIN`
- `win` is not of type `W_*`
- `win` is a print window, screen window, pixmap, or control

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on XVT/Win32, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before calling `xvt_app_create`. In that case, `TASK_WIN` is a valid window for this function.

See Also

```
E_UPDATE
NULL_WIN
PNT
TASK_WIN
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
xvt_app_create
xvt_dwin_set_font_*
xvt_win_set_caret_size
xvt_win_set_caret_visible
```

The "Cursors and Carets" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_win_set_caret_visible`.

xvt_win_set_caret_size

Set Caret's Width and Height

Summary

```
void xvt_win_set_caret_size(WINDOW win, int width,
     int height)
```

`WINDOW win`

Window whose caret's size is to be set.

`int width`

Width of caret.

`int height`

Height of caret.

Description

This function sets the size of the caret for `win`. The size is expressed in pixels. `win` must be a regular window; it *cannot* be a control, pixmap, or dialog. This function has no effect until a succeeding call to `xvt_win_set_caret_pos` is made.

If `width` is zero, a default platform-specific caret width is used. Otherwise the caret width is set to `width`.

If `height` is zero, the caret height changes dynamically according to the mapped height of the window's current logical font each time you call `xvt_win_set_caret_visible`. In addition, when `height` is zero, XVT increases the y-coordinate by the current physical font's descent value. If `height` is non-zero, then that value becomes the new caret height.

If you never call `xvt_win_set_caret_size` for a window, the caret assumes a height appropriate for the current logical drawing font as set for the window. Therefore, if you use only one logical font in a window, you don't need to call `xvt_win_set_caret_size`.

However, if you use multiple logical fonts in a window, then the caret height should match the mapped height of the logical font that the caret is adjacent to, which may not be the same as the current window's logical font. In that case, you should call `xvt_win_set_caret_size`.

Caution: Do not call `xvt_win_set_caret_size` during an update event. Doing so interferes with the way the underlying window systems implement carets and causes XVT to issue an error.

Return Value

None.

Parameter Validity and Error Conditions

XVT issues an error if any of the following are true:

- You call this function during the processing of an `E_UPDATE` event
- `win` is `NULL_WIN`
- `win` is not of type `W_*`
- `win` is a print window, screen window, pixmap, or control

Implementation Note

Normally, `TASK_WIN` is not a valid window for this function. However, on XVT/Win32, you can set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN` before calling `xvt_app_create`. In that case, `TASK_WIN` is a valid window for this function.

See Also

```
E_UPDATE
NULL_WIN
TASK_WIN
WINDOW
W_*, WC_*, WD_*, Values for WIN_TYPE
xvt_app_create
xvt_win_set_caret_pos
xvt_win_set_caret_visible
```

The "Cursors and Carets" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_win_set_caret_visible`.

xvt_win_set_caret_visible

Change the Position of the Caret

Summary

```
void xvt_win_set_caret_visible(WINDOW win,
     BOOLEAN visible)
```

`WINDOW win`

Window whose caret's visibility is being set.

`BOOLEAN visible`

If `TRUE`, makes caret visible; if `FALSE`, makes caret invisible.

Description

This function changes the visibility of the "logical" caret for a particular window. The visibility of the logical caret affects the visibility of the "physical" caret. A physical caret is a vertical bar (possibly blinking) that you use to indicate the location for text insertion.

Any window that has a visible logical caret displays a physical caret when that window has the focus. Any window that has an invisible logical caret does not display the physical caret, even when that window has focus.

You should call this function with `visible` equal to `FALSE` when the window leaves text insertion mode. Calling this function with

visible equal to FALSE cancels any previous calls to this function with visible equal to TRUE for this window.

You do not have to pair calls to this function with visibility values of TRUE and FALSE. In other words, you can call xvt_win_set_caret_visible(win, FALSE) or xvt_win_set_caret_visible(win, TRUE) repeatedly without confusing XVT.

You call xvt_win_set_caret_visible(win, TRUE) when the user enters text entry mode for a particular window. If a particular window is always in text entry mode, then you can call this function with TRUE from within the E_CREATE case of the window's event handler. The caret isn't shown automatically when you draw text or process keyboard events; you make it visible by calling this function.

XVT automatically hides the caret when a window becomes inactive and shows it again (if its visibility is TRUE) when the window becomes active.

Caution: Do not call xvt_win_set_caret_visible during an update event. Doing so interferes with the way the underlying window systems implement carets and causes XVT to issue an error.

Parameter Validity and Error Conditions

XVT issues an error if:

- win is NULL_WIN
- win is not of type W_*
- win is a print window, screen window, pixmap, or control
- This function is called during the processing of an E_UPDATE event

Implementation Note

Normally, TASK_WIN is not a valid window for this function. However, on XVT/Win32, you can set the non-portable attribute ATTR_WIN_PM_DRAWABLE_TWIN before calling xvt_app_create. In that case, TASK_WIN is a valid window for this function.

See Also

E_CREATE
E_UPDATE
NULL_WIN
PNT
TASK_WIN
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
xvt_app_create
xvt_dwin_draw_text
xvt_dwin_get_text_width
xvt_scr_beep
xvt_win_set_caret_pos
xvt_win_set_caret_size

The "Cursors and Carets" chapter in the *XVT Portability Toolkit Guide*

Example

```
switch(ep->type)
     {
    ...
    case E_CHAR:
         PNT pnt;
        static char str[MAX_CHARS +1];
        static int next = 0;if (next < MAX_CHARS) {
             next+= xvt_str_convert_wc_to_mb(&str[next],
ep->v.chr.ch);
           str[next] = NULL
           xvt_dwin_draw_text(edit_win, start_x,
start_y, str, -1);
           pnt.v = start_y;
           pnt.h = start_x +
xvt_dwin_get_text_width(edit_win, str, -1);
           xvt_win_set_caret_pos(edit_win, pnt);
           xvt_win_set_caret_visible(edit_win, TRUE);
         }
        else {
             xvt_scr_beep();
          }
     }
```

xvt_win_set_ctl_color_component

Sets the Control Color for a Color Type Used by all Controls in a Window

Summary

```
void xvt_win_set_ctl_color_component(WINDOW win,
XVT_COLOR_TYPE ctype, COLOR color)
```

`WINDOW win`

WINDOW ID of a window or dialog.

`XVT_COLOR_TYPE ctype`

Control component to set the color for.

`COLOR color`

Color to assign.

Description

This function sets the control color for a specific control component in all the existing and subsequently created controls contained in win. A call to this function overrides any previous control color setting only for the specified ctype. Excluded from this effect are those controls for which control colors have been established individually. All other colors used by the designated control not specified by a call to this function are unaffected.

Parameter and Validity Conditions

XVT issues an error if win is not a valid WINDOW or ctype is not a valid XVT_COLOR_TYPE.

See Also

```
WIN_DEF
XVT_COLOR_TYPE
XVT_COLOR_COMPONENT
xvt_ctl_set_colors
xvt_ctl_set_color_component
xvt_ctl_unset_color_component
xvt_ctl_get_colors
xvt_win_set_ctl_colors
xvt_win_get_ctl_colors
xvt_win_get_ctl_color_component
xvt_win_unset_ctl_color_component
ATTR_APP_CTL_COLORS
```

xvt_win_set_ctl_colors

Set Control Colors

Summary

```
void xvt_win_set_ctl_colors(WINDOW win,
    XVT_COLOR_COMPONENT* colors,
    XVT_COLOR_ACTION action)
```

`WINDOW win`

WINDOW ID of window or dialog.

`XVT_COLOR_COMPONENT* colors`

Colors to set or unset.

`XVT_COLOR_ACTION action`

Either set or unset the colors.

Description

This function sets (or unsets) the control colors in all the existing and subsequently created controls contained in `win`. A call to this function overrides any previous control color settings *only* for the specified `XVT_COLOR_COMPONENT`s in the colors array. Excluded from this effect are those controls for which control colors have been established individually. All colors which are *not* specified in a call to this function are unaffected.

If `NULL` is passed as the value of the `colors` array, the controls in the container which depend on the container level control colors revert to using the application default control colors. If no application default control colors are defined, the platform-specific default control colors are used.

If the `action` parameter is `XVT_COLOR_ACTION_SET`, the container control colors for the specified color components is set to the color values in the `colors` parameter. If the `action` parameter is `XVT_COLOR_ACTION_UNSET`, the container control colors for the specified color components are inherited from the application default control `color_set`.

This function does not affect the colors of the container decorations or any other colors that appear in the container itself.

Parameter and Validity Conditions

XVT issues an error if:

- win is NULL_WIN
- win is not a valid window
- win is not of type W_PLAIN, W_DBL, W_NO_BORDER, W_MODAL, WD_MODAL, WD_MODELESS or (for XVT/Win32) W_TASK
- This function is called during an E_UPDATE event

Implementation Note

Normally, TASK_WIN is not a valid window for this function. However, on XVT/Win32, you can set the non-portable attribute ATTR_WIN_PM_DRAWABLE_TWIN before calling xvt_app_create. In that case, TASK_WIN is a valid window for this function.

See Also

ATTR_APP_CTL_COLORS
XVT_COLOR_ACTION
XVT_COLOR_COMPONENT
E_UPDATE
WINDOW
xvt_ctl_set_colors
xvt_win_get_ctl_colors

xvt_win_set_ctl_font

Set Logical Control Font

Summary

```
void xvt_win_set_ctl_font(WINDOW win,
     XVT_FNTID font_id)

WINDOW win
     WINDOW ID of window or dialog.

XVT_FNTID font_id
```

Logical font.

Description

This function sets the logical font in all the existing and subsequently created controls contained in win.

A call to this function overrides any previous font setting for the contained controls, *excluding* those controls for which fonts have been established individually. If `NULL_FNTID` is passed to this function, the controls in the container which depend on the container level control font revert to using the application default control font. If no application default control font has been defined, the platform-specific default control font is used.

This function does not affect the font of the window title, menu items, or any other text in the container.

Parameter and Validity Conditions

XVT issues an error if any of the following conditions exist:

- `win` is `NULL_WIN`
- `win` is not a valid window

See Also

`ATTR_APP_CTL_FONT_RID`
`E_UPDATE`
`NULL_FNTID`
`XVT_FNTID`
`WINDOW`
`xvt_ctl_set_font`
`xvt_win_get_ctl_font`

xvt_win_set_cursor

Set Window's Cursor Shape

Summary

```
void xvt_win_set_cursor(WINDOW win, CURSOR cur)
```

`WINDOW win`

Window whose cursor's shape is to be set.

`CURSOR cur`

Cursor to be set.

Description

This function sets the `CURSOR` that identifies the symbol used to locate the mouse pointer or cursor. Recall that every XVT regular window has a cursor shape associated with it; XVT automatically changes the shape of the cursor as the user moves it from window to

window. This function sets the cursor shape associated with a particular window. For a list of the standard XVT cursors, see CURSOR_* options. You can also use a cursor that you have designed specifically for your application.

If you specify a non-standard cursor value, then that value must be the resource ID of a CURSOR resource that you have defined non-portably. For details on specifying CURSOR resources, see the *XVT Platform-Specific Books*.

Each window remembers its cursor shape, and XVT automatically changes the shape as the mouse pointer is moved around the screen, unless the mouse is trapped with a call to xvt_win_trap_pointer. Therefore, xvt_win_set_cursor should be thought of as setting the "tool cursor" for a particular window, rather than setting an application-wide cursor.

Parameter Validity and Error Conditions

Dialogs, controls, pixmaps, and print windows are not valid values for win. TASK_WIN is a valid value for win on XVT/Mac. If the attribute ATTR_WIN_PM_DRAWABLE_TWIN was set when the application was started, it is also valid for XVT/Win32.

Implementation Note

On XVT/XM, TASK_WIN is not a valid window for this function.

See Also

```
CURSOR
CURSOR_* Options
TASK_WIN
WINDOW
xvt_win_get_cursor
xvt_win_trap_pointer
```

The "Cursors and Carets" chapter in the *XVT Portability Toolkit Guide*
The *XVT Platform-Specific Books*

Example

See the example for xvt_win_trap_pointer.

xvt_win_set_doc_title

Set Document Window's Title

Summary

```
void xvt_win_set_doc_title(WINDOW win, char *title)
```

`WINDOW win`

Window whose title is being set.

`char *title`

Title to be set.

Description

This function is similar to `xvt_vobj_set_title`, except that it forms a window title that follows appropriate style guidelines for the local platform, by prepending the application name to `title`. XVT retrieves the application name from the `appl_name` field of the `XVT_CONFIG` structure passed to `xvt_app_create`. Use `xvt_win_set_doc_title` only for top-level windows.

If the window corresponds to a document with a title, use that title (e.g., "`Sales Chart`") for the `title` argument to this function. If the document is untitled, use a title of the form "`Untitledn`", where `n` is a number greater than or equal to 1. This function pulls off the number and uses a name for the untitled window that is appropriate for the local system.

Parameter Validity and Error Conditions

XVT issues an error if any of the following are true:

- `win` is invalid; print windows, screen windows, pixmaps, and controls are *not* valid values for `win`
- `title` is `NULL`
- This function is called during an `E_UPDATE` event

See Also

`XVT_CONFIG`
`E_UPDATE`
`WINDOW`
`xvt_app_create`
`xvt_vobj_set_title`

xvt_win_set_event_mask

Specify Event Restrictions

Summary

```
void xvt_win_set_event_mask(WINDOW win,
     EVENT_MASK mask)
```

`WINDOW win`

Window, dialog, or task window for which `EVENT_MASK` is to be set.

`EVENT_MASK mask`

Specified events that the window event handler can receive.

Description

This function sets the `EVENT_MASK` for a window, dialog, or task window. Controls, pixmaps, print windows, and screen windows are *not* allowed. The `EVENT_MASK` specifies the events the event handler for a window can receive, and is set by ORing together the `EM_*` constants. Only the events that have the corresponding `EM_*` values set in the event mask are sent to the event handler for the `WINDOW` (e.g., `E_CHAR` is the only event sent if only the `EM_CHAR` mask bit is set). Most of the time, you simply set the event mask to `EM_ALL`, which allows all events.

You can set an event mask either in the window/dialog creation functions, `xvt_win_create_*` and `xvt_dlg_create_*`, or by calling `xvt_win_set_event_mask`. Typically, your application would reset the event mask by calling `xvt_win_set_event_mask` to screen out events based on the current state of the `WINDOW`. For example, you might want to call `xvt_win_set_event_mask` to screen out queued `E_TIMER` events before you call `xvt_timer_destroy`. This guarantees that any timer events in the event queue will not be delivered.

Parameter Validity and Error Conditions

XVT issues an error if `win` is not a valid window, dialog, or `TASK_WIN`. Controls, pixmaps, print windows, and `SCREEN_WIN`s are not allowed.

See Also

E_CHAR
EM_* Constants
E_TIMER
EVENT_MASK
SCREEN_WIN
TASK_WIN
WINDOW
xvt_dlg_create_def
xvt_dlg_create_res
xvt_timer_destroy
xvt_win_create
xvt_win_create_def
xvt_win_create_res

Example

See the example for xvt_timer_destroy.

xvt_win_set_handler

Set Window or Dialog Event Handler

Summary

```
void xvt_win_set_handler(WINDOW win, EVENT_HANDLER eh)
```

WINDOW win

Window, dialog, or TASK_WIN whose event handler is being set.

EVENT_HANDLER eh

Event handler function.

Description

This function sets the event handler for win, which may be a regular window, dialog, or TASK_WIN. Setting the event handler for a window has the effect of changing the function that receives events for that window. You can use this function to drastically alter the behavior of a particular window. You might also use it to intercept events for a window before invoking the handler that the window was assigned when it was first created, effectively chaining together event handlers.

Another use for xvt_win_set_handler is if you want to temporarily override the event handler for a window or dialog and put it back later. In this case, you can call xvt_win_get_handler to save the

current event handler for a window or dialog, so that you can restore it later with a call to `xvt_win_set_handler`.

Parameter Validity and Error Conditions

XVT issues an error if `win` is not a valid window, dialog, or `TASK_WIN`.

See Also

`EVENT_HANDLER`
`TASK_WIN`
`WINDOW`
`xvt_win_get_handler`

Example

See the example for `xvt_win_get_handler`.

xvt_win_trap_pointer

Take Control of the Mouse

Summary

```
void xvt_win_trap_pointer(WINDOW win)
```

`WINDOW win`

Window to which the pointer is being trapped.

Description

This function traps all mouse events to the `WINDOW` specified by `win`. This means that the mouse can't be used to operate anything outside of the client area--not the scrollbars, not the menubar, and not anything in other windows. The user can still move the mouse outside of the client area, but the `CURSOR` shape of the mouse pointer won't change.

All subsequent mouse events are sent to the event handler of the window in which the mouse is trapped until the pointer is released with a call to `xvt_win_release_pointer`. However, the mouse coordinates for these events may fall outside the client area. If you want to constrain mouse events to the client rectangle, include code in your application to force them into the proper range.

You should always trap the mouse during dragging.

E_MOUSE_MOVE events are generated continuously while the mouse is trapped, to help in implementing automatic scrolling.

Do not keep the mouse trapped indefinitely, as it interferes with the user's ability to operate other applications.

Parameter Validity and Error Conditions

If win is not a valid window, XVT issues an error.

Implementation Note

On XVT/XM, the X server performs an automatic grab of the mouse when a mouse button is pressed. On these platforms, xvt_win_trap_pointer can fail depending on grabs in progress from other applications. In this case, XVT issues a warning, which your application can watch for in its error message handler.

See Also

CURSOR
CURSOR_* Options
E_MOUSE_MOVE
WINDOW
xvt_win_release_pointer
xvt_win_get_cursor
xvt_win_set_cursor

Example

This code routine can be used to toggle "rubber banding" in a window:

```
    static DRAW_CTOOLS rb_ctools = {...M_XOR...};
    ...
    void
    rubber_band(WINDOW window, int on) {
         static DRAW_CTOOLS save_tools;
        static CURSOR save_cursor;if (on) { /* turn rubber
band mode on */
            /* trap pointer, set cursor, set ctools */
           xvt_win_trap_pointer(window);
           xvt_win_get_cursor(window, &save_cursor);
           xvt_win_set_cursor(window, CURSOR_CROSS);
           xvt_dwin_get_draw_ctools(window,
&save_tools);
           xvt_dwin_set_draw_ctools(window, &rb_ctools);
        }
       else { /* turn rubber band mode off */
            /* release pointer, reset cursor & ctools */
           xvt_win_release_pointer();
           xvt_win_set_cursor(window, &save_cursor);
           xvt_dwin_set_draw_ctools(window,
&save_ctools);
        }
     }
```

xvt_win_unset_ctl_color_component

Unsets the Control Color for a Color Type Used by all Controls in a Window

Summary

```
void xvt_win_unset_ctl_color_component(WINDOW win,
XVT_COLOR_TYPE ctype)
```

`WINDOW win`

WINDOW ID of a window or dialog.

`XVT_COLOR_TYPE ctype`

Control component to unset the color for.

Description

This function unsets the control color for a specific control component in all the existing and subsequently created controls contained in win. A call to this function overrides any previous control color setting only for the specified ctype. Excluded from this effect are those controls for which control colors have been established individually. All other colors used by the designated control not specified by a call to this function are unaffected.

The control reverts to the default colors for the application or the native window system. The control colors for the specified color component is either inherited the application owned colors, or the system default.

Parameter and Validity Conditions

XVT issues an error if win is not a valid WINDOW or ctype is not a valid XVT_COLOR_TYPE.

See Also

WIN_DEF
XVT_COLOR_TYPE
XVT_COLOR_COMPONENT
xvt_ctl_set_colors
xvt_ctl_set_color_component
xvt_ctl_unset_color_component
xvt_ctl_get_colors
xvt_win_set_ctl_colors
xvt_win_set_ctl_color_component
xvt_win_get_ctl_colors
xvt_win_get_ctl_color_component
ATTR_APP_CTL_COLORS

URL Statements

Universal Resource Language Statements

URL Statement Components

Bounding Rectangle
Resource ID
Text Strings
userdata

URL Statements

accel
button Control
checkbox Control
dialog
edit Control
font
font_map
groupbox Control
icon Control
image
listbox Control
listbutton Control
listedit Control
menu and menubar
radiobutton Control
scrollbar Control
string
text Control
textedit Object
units
window

curl Resource Compiler Directives

#define
#include
#if, #elif, #else, and #endif
#ifdef and #ifndef
#scan
#transparent
#undef

Bounding Rectangle

URL Statement Component

A bounding rectangle is a sequence of four constant integer values (or expressions that evaluate to constants). The first and second

values are the x and y coordinate, respectively, of the upper-left corner of the rectangle. The third and fourth values are the width and height, respectively, of the rectangle. The values can be separated by blanks and/or commas.

Resource ID

URL Statement Component

Resource IDs are positive integer values. They can be specified as a constant or an expression that evaluates to a constant. Application should use values below 30,000.

See Also

`button Control` URL statement
`checkbox Control` URL statement
`dialog` URL statement
`edit Control` URL statement
`font URL` statement
`font_map` URL statement
`groupbox Control` URL statement
`icon Control` URL statement
`image` URL statement
`listbox Control` URL statement
`listbutton Control` URL statement
`listedit Control` URL statement
`radiobutton Control` URL statement
`scrollbar Control` URL statement
`string` URL statement
`text Control` URL statement
`textedit Object` URL statement
`window` URL statement

Text Strings

URL Statement Component

Text strings are specified inside double quotes (""). Two or more sequential strings are concatenated to form a single string. Backslashes inside a string escape certain characters as follows:

"\becomes "
\\ becomes \
\n becomes linefeed character
\t becomes tab character

In addition, a backslash followed by 1,2, or 3 octal digits is converted to the equivalent 8-bit character. A backslash at the end of a line continues the string on to the next line.

userdata

URL Statement Component

Summary

userdata *"string 1", "string 2", ... , "string n"*

Description

You can associate any number of strings with URL menus, dialogs, windows, images and controls. These strings become the user data for their associated objects, and can be retrieved by the application program using the functions `xvt_res_get_menu_data`, `xvt_res_get_image_data`, `xvt_res_get_dlg_data`, and `xvt_res_get_win_data`.

User data strings are each limited to 2048 characters. User data is allowed any of the following URL statements:

button
checkbox Control
dialog
edit
groupbox Control
icon Control
image
menu
item
listbox Control
listbutton Control
radiobutton Control
scrollbar Control
text Control

textedit Control
window

See Also

`button Control` URL statement
`checkbox Control` URL statement
`dialog` URL statement
`edit Control` URL statement
`groupbox Control` URL statement
`icon Control` URL statement
`image` URL statement
`menu and menubar` URL statement
`listbox Control` URL statement
`listbutton Control` URL statement
`listedit Control` URL statement
`radiobutton Control` URL statement
`scrollbar Control` URL statement
`text Control` URL statement
`textedit Object` URL statement
`window` URL statement

The "Resources and URL" chapter in the *XVT Portability Toolkit Guide*

accel

URL Statement

Summary

`accel` *`tag symbol`* `[modifiers]`

Description

This statement specifies a keyboard equivalent for a menu item. The `tag` usually appears in an `item` specification of a menu statement, but it need not.

The *key* must be either a single ASCII character enclosed in double quotes, or one of the following symbols. Note that these symbols are unlike XVT virtual key codes, in that they represent actual keys on the keyboard. Not all keyboards have all keys.

Symbol	Key	Symbol	Key
f1	F1	kp0	0 on keypad

...	...	...	...
f24	F24	kp9	9 on keypad
tab	Tab	ins	Insert
back	Backspace	del	Delete
return	Return	copy	Copy
esc	Escape	cut	Cut
mult	* on keypad	paste	Paste
sub	- on keypad		
add	+ on keypad		

Some of these symbols (e.g., del, back) duplicate ASCII keys and thus can be specified in two ways.

The optional *modifiers* are a sequence of the following symbols. They can be in any order. They represent the modifier keys held down while the key is pressed: `control`, `shift`, `alt`.

The modifier `alt` means the command (or Apple) key on the Macintosh.

Example

The standard accelerators for XVT/Win32 can be specified like this:

```
accel M_EDIT_CUT DEL SHIFT
accel M_EDIT_COPY INS CONTROL
accel M_EDIT_PASTE INS SHIFT
accel M_EDIT_CLEAR DEL CONTROL
accel M_EDIT_UNDO BACK ALT
accel M_HELP_CONTENTS F1
accel M_FONT_NORMAL F5
accel M_FONT_BOLD F6
accel M_FONT_ITALIC F7
accel M_FONT_UNDER F8
```

These are some typical accelerators for the Mac:

```
accel M_FILE_OPEN "o" alt
accel M_FILE_SAVE "s" alt
accel M_FILE_QUIT "q" alt
```

Accelerators are more look-and-feel dependent than menus, so conditional compilation is usually required.

See Also

`menu and menubar` URL statements

button Control

URL Statement

Summary

`button` *id rect "title" [options] [userdata]*

Description

This statement specifies a button control with resource ID *id*, bounding rectangle *rect*, and label text *label*. If the height is given as zero, the URL compiler automatically chooses the recommended height for the target platform. If a tilde (~) appears in the label, the letter that follows is a mnemonic for that control.

The *options* can be one or more of the following:

Option	Meaning
`default`	Default button
`disabled`	Initially disabled
`invisible or hidden`	Initially hidden
`native_just`	Native text justification (the default)
`left_just`	Left-justified text
`center_just`	Center-justified text
`right_just`	Right-justified text

Note: If you specify more than one of the `*_just options`, results are unpredictable.

See Also

`Bounding Rectangle` URL statement component
`Resource ID` URL statement component
`Text Strings` URL statement component
`userdata` URL statement component
`dialog` URL statement
`window` URL statement

The "Resources and URL" chapter in the *XVT Portability Toolkit Guide*

checkbox Control

URL Statement

Summary

checkbox *id rect "title" [options] [userdata]*

Description

This statement is the same as the button statement, except that it specifies a check box. It can use tilde (~) mnemonics; see `button Control`. The options can be one or more of the following:

Option	Meaning
`checked`	Initially checked (Mac only)
`disabled`	Initially disabled
`invisible or hidden`	Initially hidden
`native_just`	Native text justification (the default)
`left_just`	Left-justified text
`center_just`	Center-justified text
`right_just`	Right-justified text

Note: If you specify more than one of the `*_just` options, results are unpredictable.

See Also

`Bounding Rectangle` URL statement component
`Resource ID` URL statement component
`Text Strings` URL statement component
`userdata` URL statement component
`button Control` URL statement
`dialog` URL statement
`window` URL statement

The "Resources and URL" chapter in the *XVT Portability Toolkit Guide*

dialog"

URL Statement

Summary

`dialog` *id rect ["title"] [type] [options] [userdata]*

... control statements ...

Description

This statement defines a dialog box whose resource ID is given by *id*. Control statements (e.g., `button`, `listbox`) follow.

The dialog box's location and size are specified by *rect*, as described in `window`. The location of a dialog is relative to the screen.

The *title*, which must be enclosed in double quotes, specifies the dialog's title. The *type* is either `modal` or `modeless`. If type is omitted, the dialog is `modal`.

The `options` for dialogs can be one or more of:

Option	Meaning
`disabled`	Initially disabled
`invisible or hidden`	Initially hidden

These options should only be used with modeless dialogs, since they don't make sense for modal dialogs.

The order of control statements following a dialog statement usually matters. When the keyboard is used to navigate within the dialog, it visits the controls in the order in which they appear. Also, the first control that appears is the one that gets the initial keyboard focus. On systems that do not use the keyboard to navigate, such as the Mac, the first edit control that appears gets the focus.

XVT defines a number of standard dialogs. These dialogs have IDs in the range 9050 to 9099, so these numbers should not be used for the dialog ID.

Within a dialog statement control IDs have to follow XVT's numbering requirements. The Default and Cancel buttons must have IDs equal to `DLG_OK` and `DLG_CANCEL`, respectively. Other numbers can be chosen at will, but they must be unique within a single dialog and they must be less than 9000.

See Also

`DLG_* Control IDs`
`Bounding Rectangle` URL statement component
`Resource ID` URL statement component
`Text Strings` URL statement component
`userdata` URL statement component
`window` URL statement

The "Dialogs" and the "Resources and URL" chapter in the *XVT Portability Toolkit Guide*

edit Control

URL statement

Summary

edit *id rect ["text"] [options] [userdata]*

Description

This statement specifies an edit control, with the given resource ID *id*, bounding rectangle *rect*, and initial text *text*. A zero height tells the compiler to pick the default height for the target platform. The *text* can be omitted, in which case the edit control will be initially empty.

The `options` can be one or more of the following:

Option	Meaning
`disabled`	Initially disabled
`invisible or hidden`	Initially hidden
`native_just`	Native text justification (the default)
`left_just`	Left-justified text
`center_just`	Center-justified text
`right_just`	Right-justified text
`opt1`	Use 9-point Geneva font (Mac only)
`opt2`	Use 9-point Monaco font (Mac only)
`opt3`	Multiline (Mac only)
`opt4`	Wordwrap (Mac only)
`password`	Password style edit field

Note: If you specify `opt1` with `opt2`, results are unpredictable.

See Also

`Bounding Rectangle` URL statement component
`Resource ID` URL statement component
`Text Strings` URL statement component
`userdata` URL statement component
`dialog` URL statement
`window` URL statement

The "Resources and URL" chapter in the *XVT Portability Toolkit Guide*

font

URL Statement

Summary

font *id family size [style] [***map** *native_desc]*

Description

This statement defines a logical font whose resource ID is given by *id*. This logical font can specify the font for an object (currently, only text edit objects support this). Or it can specify the mapping from a logical font to a physical font or both.

In the font resource, *family*, *size*, and *[style]* contain the XVT portable attributes (family, style, size). The *family* is a string in double quotes; it can be any value. The size must be a constant expression or the keyword `any` (used to specify wildcarding for mapping purposes). You can also use the wildcard "`any`" for the *[style]* field. If the URL font mapper encounters a wildcard, it allows any size or style, respectively, to be mapped to the specified native descriptor.

[style] specifies the font styles, and can be any of the following:

- any
- printer
- user1
- blink
- scalable
- user2

- bold
- shadow
- user3
- inverse
- strikeout
- user4
- italic
- underline
- user5
- outline

The any keyword indicates wildcarding for the font style, and therefore does not make sense with other style options. The keywords `user1 - user5` are available for user-defined styles if the application supplies its own font mapper.

The **`map`** `keyword` and the string `native_desc` (which must be in double quotes) indicate a mapping from the logical font to a physical font.

Example

Here is an example of how you would define "`MYFONT101`" in your URL file:

```
#define MYFONT101 1
font MYFONT101 "helvetica" 12 bold italic
```

See Also

`XVT_FONT_STYLE_MASK`
`xvt_res_get_font`
`Resource ID` URL statement component
`font_map` URL statement

font_map

URL Statement

Summary

`font_map` *`id native_desc`*

Description

This statement defines a mapping from the logical font given by the resource ID *id* (previously defined in the URL file by a font statement) to the physical font specified in the string *native_desc*, which must be in double quotes.

You can use wildcards ("*") in the *native_desc* field. If a native descriptor contains wildcards, the corresponding portable attributes are used as the value of the native attribute.

Example

Here is an example of defining a native mapping for "`MYFONT101`" on XVT/XM:

```
font_map MYFONT101 "X1101/adobe/helvetica/bold/i/\
*/*/*/120/*/*/*/*/*/*"
```

Tip: You can avoid using the `font_map` statement altogether by appending the native descriptor to the end of the font statement, preceded by the keyword map, like this:

```
font MYFONT101 "helvetica" 12 bold italic map "X1101\
adobe/helvetica/..."
```

See Also

`Resource ID` URL statement component
`font` URL statement

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*
The *XVT Platform-Specific Books*

groupbox Control

URL Statement

Summary

groupbox *id rect ["label"] [options] [userdata]*

Description

This statement specifies a rectangle that causes other controls within a dialog box to appear as a set. The text string *label* appears at the

top of the group box. The *options* can be one or more of the following:

Option	Meaning
`disabled`	Initially disabled
`invisible or hidden`	Initially hidden
`native_just`	Native text justification (the default)
`left_just`	Left-justified text
`center_just`	Center-justified text
`right_just`	Right-justified text
`opt1`	Use 9-point Geneva font (Mac only)
`opt2`	Use 9-point Monaco font (Mac only)

Note: If you specify more than one of the `*_just` options, or if you specify `opt1` with `opt2`, results are unpredictable.

See Also

`Bounding Rectangle` URL statement component
`Resource ID` URL statement component
`Text Strings` URL statement component
`userdata` URL statement component
`dialog` URL statement
`window` URL statement

The "Resources and URL" chapter in the *XVT Portability Toolkit Guide*

icon Control

URL Statement

Summary

`icon` *`id rect ref-id [options] [userdata]`*

Description

This statement specifies an icon to be placed into the specified bounding rectangle. The program uses the *id* to refer to the control. The *ref-id* refers to the ID of a platform-specific icon resource not specifiable in URL. (You can define it with native resource statements within an URL `#transparent` statement.) On some platforms, the height and/or width of the `rect` is ignored, and is taken from the definition of the icon itself.

The *options* can be any of the following:

Option	Meaning
`disabled`	Initially disabled
`invisible or hidden`	Initially hidden

See Also

`Bounding Rectangle` URL statement component
`Resource ID` URL statement component
`userdata` URL statement component
`dialog` URL statement
`#transparent` URL statement
`window` URL statement

The "Resources and URL" chapter in the *XVT Portability Toolkit Guide*
The *XVT Platform-Specific Books*

image

URL Statement

Summary

`image` *`id "filename" [options] [userdata]`*

Description

This statement defines a portable image resource whose resource ID is given by *id*. The *filename* is a string in double quotes.

Tip: To get an image from a resource file:

Call `xvt_res_get_image`.

Currently the only available option is `reference`, which indicates that only the filename is stored in the resource file. This file is searched for at runtime only if a reference is made to this image resource.

If the `reference` option is not used, the entire image file is copied into the resource file. **curl** searches for this file in the same way as files specified in the `#include` or `#scan` directives.

curl does no validity checking on this file and assumes it is in MS-Windows BMP format.

See Also

`xvt_res_get_image`
`Resource ID` URL statement component
`userdata` URL statement component
`#include` preprocessor directive
`#scan` preprocessor directive

listbox Control

URL Statement

Summary

`listbox` *`id rect [options] [userdata]`*

Description

This statement specifies a list box, with given ID and rectangle. The rectangle must include space for the scrollbar. The options can be one or more of the following:

Option	Meaning
`disabled`	Initially disabled
`invisible or hidden`	Initially hidden
`readonly`	Read only; nothing can be selected
`multiple`	Multiple items can be selected
`opt1`	Use 9-point Geneva font (Mac only)
`opt2`	Use 9-point Monaco font (Mac only)

Note: If you specify `opt1` with `opt2`, results are unpredictable. If no option is chosen, the list box allows one selection at a time.

Implementation Note

On some XVT-supported platforms, you may encounter native (not XVT-imposed) memory limitations that apply to individual list boxes. For example, the number of items that you can add to a list box is determined by the combined size of the items and the memory available on the global heap.

See Also

Bounding Rectangle URL statement component
Resource ID URL statement component
`userdata` URL statement component
`dialog` URL statement
`window` URL statement

The "Resources and URL" chapter in the *XVT Portability Toolkit Guide*

listbutton Control

URL Statement

Summary

`listbutton` *`id rect [options] [userdata]`*

Description

This statement specifies a list button control, with the given ID and bounding rectangle. The bounding rectangle is the size of the control when the list is displayed. The *options* can be one or more of the following:

Option	Meaning
`invisible or hidden`	Initially invisible
`disabled`	Initially disabled
`native_just`	Native text justification (the default)
`center_just`	Center-justified text
`right_just`	Right-justified text

Note: If you specify more than one of the `*_just` options, results are unpredictable.

Implementation Note

On some XVT-supported platforms, you may encounter native (not XVT-imposed) memory limitations that apply to individual list button controls. For example, the number of items that you can add to a list button control is determined by the combined size of the items and the memory available on the global heap.

See Also

`Bounding Rectangle` URL statement component
`Resource ID` URL statement component
`userdata` URL statement component
`dialog` URL statement
`window` URL statement

The "Resources and URL" chapter in the *XVT Portability Toolkit Guide*

listedit Control

URL Statement

Summary

`listedit` *`id rect [options] [userdata]`*

Description

This statement specifies a list edit control, with the given ID and bounding rectangle. The bounding rectangle is the size of the control when the list is displayed. The options can be one or more of the following:

Option	Meaning
`invisible or hidden`	Initially invisible
`disabled`	Initially disabled
`native_just`	Native text justification (the default)
`left_just`	Left-justified text
`center_just`	Center-justified text
`right_just`	Right-justified text

Note: If you specify more than one of the `*_just options`, results are unpredictable.

Implementation Note

On some XVT-supported platforms, you may encounter native (not XVT-imposed) memory limitations that apply to individual list edit controls. For example, the number of items that you can add to a list edit control is determined by the combined size of the items and the memory available on the global heap.

See Also

`Bounding Rectangle` URL statement component
`Resource ID` URL statement component
`userdata` URL statement component
`dialog` URL statement
`window` URL statement

The "Resources and URL" chapter in the *XVT Portability Toolkit Guide*

menu and menubar

URL Statement

Summary

```
menu menu_tag ["title"] [options] [userdata]

submenu menu_tag "title" [options]
item menu_tag "title" [options] [userdata]
item menu_tag "title" [options] [userdata]
...
separator
...
item menu_tag label [options] [userdata]
...

menubar menu_tag
```

Description

The `menubar` statement defines the contents of a menubar and all of its submenus. There may be multiple menubar statements, each defining a different menubar, but at least one menubar is required.

The `menu` statement defines a menu consisting of a list of submenus, separators, and menu items. The `submenu` statement refers to another `menu` statement which defines the appearance of the submenu. For portability, a `menu` statement referenced by a `menubar` statement should only contain submenu statements, not items or separators.

Each *menu_tag* must be a constant expression. In practice, a `#defined` symbol is usually coded. User-defined menu tags must be in the range 1 to 32000; tags above 32000 are reserved by XVT for internal use.

A *label* must be a string enclosed in double quotes. If a tilde (~) appears in a label, the letter following it is the mnemonic for that menu or item; the tilde itself does not appear on the screen. On the screen menu, the mnemonic may be indicated to the user via some other means, such as underlining.

As indicated, the *label* is required for the submenu and item statements. It is optional for the `menu` statement, and is provided for compatibility with previous releases of XVT. If a `submenu` statement refers to a `menu` statement having a label, the label appearing on the `menu` statement is ignored.

These are the `options` for the `menu` and `submenu` statements:

Option	Meaning
`disabled`	Item is disabled (enabled is the default)

These are the `options` for the `item` statement:

Option	Meaning
`disabled`	Item is initially disabled
`checkable`	Item can be checked, but isn't necessarily checked
`checked`	Item is initially checked (implies checkable)

XVT defines platform-specific versions of certain standard menus. These are referred to by the following URL macros:

`DEFAULT_FILE_MENU`	Platform-specific default file menu
`DEFAULT_FONT_MENU`	Platform-specific default font menu
`DEFAULT_EDIT_MENU`	Platform-specific default edit menu
`DEFAULT_HELP_MENU`	Platform-specific default help menu

For example, the following URL fragment defines a menubar containing the standard File and Edit menus; it also contains the standard Font/Style menus, which may expand to zero, one, or two submenus:

```
MENUBAR 1000 MENU 1000
DEFAULT_FILE_MENU
DEFAULT_EDIT_MENU
DEFAULT_FONT_MENU
```

You can use these macros in place of submenu statements in a menubar. The result is the inclusion of the appropriate menu for each platform. This allows you to include or exclude the standard menus as desired.

If you wish to replace (as opposed to exclude) a standard menu, use one or more of the following statements in your URL file, before

including **url.h** (or define them on the **curl** command line as shown in "Resources and URL" of the *XVT Portability Toolkit Guide*.

```
#define NO_STD_FILE_MENU
#define NO_STD_EDIT_MENU
#define NO_STD_FONT_MENU
#define NO_STD_HELP_MENU
```

These suppress the corresponding menu statements in **url.h**, allowing you to redefine the standard menus.

See Also

`accel` URL statement

The "Menus" chapter in the *XVT Portability Toolkit Guide*

radiobutton Control

URL Statement

Summary

radiobutton *id rect "title" [options] [userdata]*

Description

This statement is the same as the checkbox statement, except that it specifies a radio button. It can use tilde (~) mnemonics; see `button Control`. The options can be one or more of the following:

Option	Meaning
`checked`	Initially checked (Mac only)
`disabled`	Initially disabled
`invisible or hidden`	Initially hidden
`group`	First or last in its group
`native_just`	Native text justification (the default)
`left_just`	Left-justified text
`center_just`	Center-justified text
`right_just`	Right-justified text

Note: If you specify more than one of the `*_just` options, the results are unpredictable.

The first and last radio buttons of a group that are meant to operate together must have the group option, and the radio group must be defined with consecutive `radiobutton` statments. These grouped radio buttons cannot be separated by radio buttons that are not part of the group, or by any other type of control. This does not make the

radio buttons behave as a group, but affects keyboard navigation on platforms that support it.

See Also

Bounding Rectangle URL statement component
Resource ID URL statement component
Text Strings URL statement component
`userdata` URL statement component
`dialog` URL statement
`window` URL statement

The "Resources and URL" chapter in the *XVT Portability Toolkit Guide*

scrollbar Control

URL Statement

Summary

`scrollbar` *`id rect [options] [userdata]`*

Description

This statement specifies a vertical or horizontal scrollbar, with the given ID and bounding rectangle. To specify a vertical scrollbar, make the width less than the height; for a horizontal scrollbar, make the height less than the width.

The *options* can be any of the following:

Option	Meaning
`disabled`	Initially disabled
`invisible or hidden`	Initially hidden

See Also

`B`ounding Rectangle URL statement component
Resource ID URL statement component
`userdata` URL statement component
`dialog` URL statement
`window` URL statement

The "Resources and URL" chapter in the *XVT Portability Toolkit Guide*

string

URL Statement

Summary

string *id "string"*

Description

This statement specifies a string resource whose resource ID is *id*. Strings should be numbered less than 30000. The string itself must be in double quotes. Note that a backslash () can be used to continue it onto the next line.

Implementation Note

For the XVT/Mac only, if consecutive `string` statements appear with consecutive `ids`, **curl** generates a `STR#` resource instead of multiple `STR` resources. This means that the strings can be retrieved only via `xvt_res_get_str_list`, not by `xvt_res_get_str`. Because `xvt_res_get_str_list` works for strings with consecutive ids on platforms other than the Macintosh (i.e., those without `STR#` resources), it is the portable routine to call for such strings.

See Also

Resource ID URL statement component
Text Strings URL statement component

The "Resources and URL" chapter in the *XVT Portability Toolkit Guide*

text Control

URL Statement

Summary

text *id rect ["title"] [options] [userdata]*

Description

This statement specifies a static text control, with the given ID, bounding rectangle, and initial text. A zero height tells the compiler to pick the default. The *text* can be omitted, in which case the static

control is initially blank. The text can be set or changed by the application using `xvt_vobj_set_title`.

The *options* can be one or more of the following:

Option	Meaning
`disabled`	Initially disabled
`invisible or hidden`	Initially hidden
`native_just`	Native text justification (the default)
`left_just`	Left-justified text
`center_just`	Center-justified text
`right_just`	Right-justified text
`opt1`	Use 9-point Geneva font (Mac only)
`opt2`	Use 9-point Monaco font (Mac only)

Note: If you specify more than one of the `*_just` options, or if you specify `opt1` with `opt2`, results are unpredictable.

See Also

`Bounding Rectangle` URL statement component
`Resource ID` URL statement component
`Text Strings` URL statement component
`userdata` URL statement component
`dialog` URL statement
`window` URL statement

The "Resources and URL" chapter in the *XVT Portability Toolkit Guide*

textedit Object

URL statement

Summary

`textedit` *`id rect ["title"] [options] [userdata]`*

Description

This statement specifies a text edit object with resource ID *id*, bounding rectangle *rect*, and optional initial text *text*. A text edit object can only be a control in a window, not in a dialog.

The options may be one or more of the following:

```
autohscroll
autovscroll
border
disabled
enableclear
invidible
nocut
nocopy
onepar
nomenu
nopaste
overtype
readonly
wrap

limit integer
margin integer
font <fid>
```

If you specify the optional font option, `fid` must reference a font resource defined previously in the URL file.

See Also

`xvt_tx_create`
Bounding Rectangle URL statement component
Resource ID URL statement component
Text Strings URL statement component
`userdata` URL statement component
`dialog` URL statement
`window` URL statement

The "Resources and URL" chapter in the *XVT Portability Toolkit Guide*

units

URL Statement

Summary

`units` *`unit`*

This optional statement, which can appear multiple times, generates no resources but merely specifies what units are used in subsequent `dialog` or `window` statements. `units` can be one of the following:

```
pixels
```

Units are device pixels (the default).

`chars`

Units are characters, in the system font.

`semichars`

Units are quarter characters horizontally and eighth characters vertically in the system font.

For those platforms whose resource managers don't naturally work in chars or semichars, the URL compiler assumes that characters are 6 pixels wide and 13 pixels high. These numbers are used whenever **curl** has to translate between char/semichars and pixel units.

Implementation Note

If you design a dialog box in terms of pixels with XVT/Mac, it will convert nicely when moved to XVT/Win32.

Coordinates are always relative to an origin at the upper-left.

See Also

`curl`

---

window

URL Statement

Summary

window *id rect ["title"] [type] [options] [menu_id] [userdata]*

...control statements...

Description

This statement defines a window whose resource ID is given by *id*. Control statements (e.g., `button`, `listbox`) follow.

The window's client area size and location are specified by *rect*, which is a sequence of four constant expressions. The first two specify the x- and y-coordinates relative to the window's parent.The next two are the width and height of the rectangle. Thus, if the units are pixels, a window 200 pixels wide by 100 high, with its upper left corner at (50, 50) is specified as:

```
50 50 200 100
```

An integer in a `rect` can be followed by a comma if desired. This is essential if the next coordinate is negative:

```
50, -50, 200, 100
```

Without the comma, the negative sign would be interpreted as a subtraction operator.

The *title*, which must be enclosed in double quotes, specifies the window's title. The `type` specifies an XVT window type, which is one of these:

- doc
- dbl_border
- plain
- no_border
- modal

If no `type` is given, the default is `doc`. If the window type does not support a title, then `title` is ignored.

options specifies the window style flags, which can be one or more of these:

- size
- iconizable
- close
- iconized
- hscroll
- sizeonly
- vscroll
- maximized
- invisible or hidden
- no_menubar
- disabled
- place_exact

The `menu_id` refers to an URL menubar tag, and it should be given if `no_menubar` was *not* used. Menubars are not supported for modal windows.

See Also

`WSF_* Options Flags`
Bounding Rectangle URL statement component
Resource ID URL statement component
Text Strings URL statement component
`menu and menubar` URL statements

The "Resources and URL" chapter in the *XVT Portability Toolkit Guide*

define

curl Resource Compiler Preprocessor Directive

Summary

`#define` *`identifier substitution-text`*

-OR-

`#define`*`identifier (identifier-list) substitution text`*

Description

The `#define` directive defines symbols or macros with arguments. In the first form, this directive causes all subsequent appearances of the identifier in non-preprocessor lines to be replaced by the substitution text. The identifier is replaced only if it appears as a token; it is not replaced if it appears in a comment, a string, or as part of a larger token.

In the second form, there must be no space between the `identifier` and the first parenthesis. Subsequent occurrences of the identifier, followed by optional white space, followed by an open parenthesis (`(`), followed by a sequence of comma-separated tokens, followed by a closed parenthesis (`)`) constitute a call of the macro. The usual C rules apply for replacement of the identifiers in the `identifier-list` by the token sequences in the call.

Only parts of some preprocessor lines are subject to token replacement before processing, as follows:

Lines	Replacement
`#include`	The part following the token include
`#define`	None

`#undef`	None
`#if`	The part following the token if
`#ifdef`	None
`#ifndef`	None
`#elif`	The part following the token elif
`#else`	None
`#endif`	None

Whenever a token is replaced, the substitution text is re-scanned for further replacements. But, upon re-scanning, an identifier that was previously replaced isn't replaced again--it is left as is.

White space (tabs or spaces) must separate the identifier and the substitution text. This white space and any that follows the substitution text is not considered to be part of the substitution text.

If there is no substitution text, the identifier is effectively removed when it appears, but it is still considered to be defined and it has the value 1 when tested with the `#if` directive. Undefined identifiers have a value of 0 in `#if` and `#elif` directives.

A `#defined` identifier can't be redefined to anything other than its original definition. If its definition has to be changed, it must first be undefined with an `#undef` directive.

See Also

`#undef` preprocessor directive
`#ifdef and #ifndef` preprocessor directives
`curl`

#include

curl Resource Compiler Preprocessor Directive

Summary

The `#include` directive has one of these forms:

`#include <`*`path-spec`*`>`

-OR-

`#include "`*`path-spec`*`"`

Description

This directive tells the compiler to treat the contents of the named file as if they appeared literally at the point of the `#include` directive. `#include` directives can nest to a depth of 32.

If the *path-spec* contains directories (e.g., **../incl/rsrc.inc**), it is taken as specified. If not and if it is surrounded by angled brackets, it is looked for in the standard places. If *path-spec* is surrounded by quotation marks, the compiler looks for it first in the directory containing the file containing the `#include` directive, and then in the standard places.

The "standard places" are the directories specified by `ëi` arguments on the command line (in the order in which they are listed), then the directories specified by the include list (see below), and then (on UNIX only) the directory **/usr/include**.

Implementation Note

For the **curl** line compiler, the "include list" can be specified by the value of the `INCLUDE` environment variable. That value must be a semicolon-separated list of directories. If a pathname contains backslashes, they are taken literally. They do not have to be doubled as they might if they appeared elsewhere in string constants.

See Also

`#scan` preprocessor directive
`curl`

#if, #elif, #else, and #endif

curl Resource Compiler Preprocessor Directives

Summary

`#if` *`const-expr`*

... text ...

`#elif` *`const-expr`*

... text ...

`#elif` *`const-expr`*

... text ...

`#else`

```
    ... text ...
#endif
```

Description

These directives control the compilation of portions of a source file. The `#elif` and `#else` clauses are optional. If present, they can nest to a depth of 50.

If a *const-expr* evaluates to a zero value, it is considered to be false. Otherwise, it is true. When encountering an `#if...endif` construction (possibly including `#elif` and `#else`) clauses, the compiler looks for the first *const-expr* that's true and compiles the text following it, ignoring the other text blocks. If none is true and there is an `#else` clause, the text following it is compiled. If there is no `#else` clause, no text is compiled.

When preprocessor directives appear in a conditional `#if` block that is skipped, **curl** ignores the directives. Preprocessor directives are processed when they appear in a conditional `#if` block that is compiled.

For each *const-expr*, before any token replacement occurs, a scan is made for the operator defined, which must have an identifier as an argument:

```
defined(identifier)
```

Each use of `defined` is replaced by a 1 if the identifier is defined (even as the empty string), and by 0 otherwise.

After all `defined` operators are processed, defined tokens are replaced by their substitution text, as discussed in the `#defined` directive. After all replacements, the result must be a constant expression.

The expression is then evaluated (according to the rules of C) using integer arithmetic, and the result is taken as false if it is zero and true otherwise.

See Also

`curl`

#ifdef and #ifndef

curl Resource Compiler Preprocessor Directives

Summary

`#ifdef` *`identifier`*

`#ifndef` *identifier*

Description

The first directive is exactly equivalent to

```
#if defined(identifier)
```

The second is equivalent to

```
#if !defined(identifier)
```

Both must be followed by an #endif directive. `#elif` and `#else` directives can intervene, along with text to be optionally compiled.

See Also

`#if, #elif, #else, and #endif` preprocessor directives
`#define` preprocessor directive
`curl`

#scan

curl Resource Compiler Preprocessor Directive

Summary

`#scan` *`"path-spec"`*

Description

The `#scan` directive is identical to the `#include` directive, except that all included text is ignored. Preprocessor directives are fully processed, however, and definitions for symbols and macros are retained. Thus, you should use `#scan`, instead of `#include`, for header files such as **xvt.h**, since the C code (e.g., `typedefs`) contained there is meaningless to **curl** and would generate a syntax error.

`#include` directives contained within a file that is `#scan`ned are treated as though they were `#scan` directives.

As an example, here are the first few lines of the standard **curl** header **url.h**:

```
#define NO_INCLUDES
#scan "xvt.h"
#include "url_plat.h" */
```

The file **xvt.h** is used only for its symbols and macros, so it is `#scan`ned. On the other hand, **url_plat.h** contains URL statements, so it is `#include`d. The file **url.h** itself *must* be `#include`d, because it contains URL directives. These, of course, should not be ignored. The symbol `NO_INCLUDE` is used internally by **xvt.h** to suppress the inclusion (or even scanning) of standard C headers (e.g., **stdio.h**).

See Also

`#include` preprocessor directive
`curl`

#transparent

curl Resource Compiler Preprocessor Directive

Summary

```
#transparent sentinel [literal] [no_include]
... arbitrary lines of text ...
sentinel
```

Description

This statement causes all lines following it to be copied to the output resource script until the *sentinel* is detected at the beginning of a line. The intervening lines are scanned for preprocessor directives and identifier substitutions are made unless the keyword *literal* is specified.

If `ino_include` is specified, all `#include` and `#scan` directives in the intervening lines are ignored; the referenced files are not included or scanned.

Note: `#transparent` is a statement in the URL language, on the same level as the menu and dialog statements. It can't be used for an individual control within a `dialog` or `window` statement, or for an item within a menu statement.

Not all platforms allow `#transparent`. Some produce binary resource data directly, without going through a native resource-script compiler. In these cases, the transparent text is ignored and doesn't appear in **curl**'s output.

As **curl** doesn't parse the text output by `#transparent` statements, it can't check it for correctness. If you make a mistake, you will probably get an error message from the window system's native resource compiler that processes the **curl** output. In most cases these messages are extremely difficult to understand, and the offending input statement is hard to locate. Therefore, it's best to code in URL to the extent possible, reserving `#transparent` statements for what is inexpressible in URL.

See Also

`#include` preprocessor directive
`#scan` preprocessor directive
`curl`

#undef

curl Resource Compiler Preprocessor Directive

Summary

`#undef` *`identifier`*

Description

The `#undef` directive removes the definition of an identifier. It's okay to `#undefine` an identifier that isn't defined.

See Also

`#define` preprocessor directive
`curl`

Help File Statements

Help File Source Statements
BODYSTANZA
BROWSE
Comments
FONT

HEADER, VERSION, APPNAME
HTOPIC and BTOPIC

helpc Help File Compiler Preprocessor Directives

#define
#if, #elif, #else, and #endif
#ifdef and #ifndef
#include
#scan

Formatting Commands

Bitmap (\P)
Font Change (\F)
Hanging Indentation (I)
Horizontal Line (\V)
Hot Button (\B)
Hyperlink (\L)
Margin (\M)
No Word Wrap (\N)
Paragraph (\A)
Reserved (\S)
Word Wrap (\W)

Predefined Help Topic Information

Predefined Help IDs
Predefined Help Topics

Comments

Help File Source Statement

Description

Comments can appear any where in the source file. A comment is denoted by a single quote (’) at the beginning of the line, like this:

```
’ This is a comment.
```

If you need to place a quote at the beginning of a line as part of the displayed help text, place a backslash before it, like this:<indented normal’these’ quote marks will appear in the help text.

See Also

helpc

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

BODYSTANZA

Help File Source Section

Summary

```
BODYSTANZA<
help topics><help topics>
```

The help text (the text that your application's user reads).

Description

The `BODYSTANZA` statement marks the end of the header section and the start of the help text.

The help text follows the header in your help source file. The help text is divided into help topics. Each topic describes one concept or process, and is displayed in its entirety in a topic window or pop-up window. You can freely insert comments and preprocessor statements within help topic text.

See Also

`HTOPIC and BTOPIC` help file source statements
`helpc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

BROWSE

Help File Source Section

Summary

```
BROWSE <sequence name><sequence name>
```

The name of the browse sequence.

Description

Browse sequence definitions declare the names of all browse sequences used within the help source file. The names must be composed only of upper- or lowercase alphanumeric characters (without spaces or other punctuation). The names are used later in

the file to connect related help topics into separate browse sequences.

Example

```
BROWSE Main
BROWSE Controls
```

See Also

`HTOPIC and BTOPIC` help file source statements
`helpc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

FONT

Help File Source Statement

Summary

`FONT <code> <font descriptor><code>`

A single character that refers to this font specification. This character is used in font change formatting commands (see `Font Change (\F)`).

`<font descriptor>`

A portable or native font descriptor. A font descriptor has the following form:

`"[plat-name:]<family>[,<size>[,style-flag]]"` where:

`plat-name` is one of the following (use only with platform-specific family):

`WIN01` Use with XVT/Win32 when linking with the native help viewer
`NT_01` Use for XVT/Win32 when linking with the XVT Portable Help Viewer
`X1101`

family is one of the following:
`FIXED`
`TIMES`
`HELVETICA`
`*COURIER`

```
SYSTEM
```
platform-specific (must specify a platform-name)

style-flag is one of the following:

```
BOLD
ITALIC
*UNDERLINE
*OUTLINE
*SHADOW
*INVERSE
*BLINK
*STRIKEOUT
```

Note: * Only the portable help compiler and the portable help viewer will attempt to map these font descriptors.

Description

Font specification statements associate single characters, called font codes, with a particular font, size, and style. In the body of your help text, these codes determine how text appears in the help viewer.

Example

Here are some sample font specifications:

```
FONT0   "TIMES,14"
FONTA   "HELVETICA,12"
FONTB   "TIMES,12,Italic"
```

See Also

helpc

The "Hypertext Online Help" and the "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*
The *XVT Platform-Specific Books*

HEADER, VERSION, APPNAME

Help File Source Statements

Summary

```
HEADER
VERSION <version number>
APPNAME "application name"
(font specifications)
(browse sequence definitions)
```

Description

Your help source file must contain a header section that precedes all other commands in the file (except comments and other preprocessor commands). `VERSION` denotes the version of the markup language used in this source file.

For XVT R4 help source files, always use a version number of 300. `APPNAME` contains the complete name of your application, enclosed in double quotation marks (""). If the name of your application has no blank characters, you can omit the quotation marks. The font specification and browse sequence definition sections enclosed in parentheses are optional, but the other lines are mandatory.

Example

```
APPNAME "My New Database"
APPNAME NewDatabase
```

At runtime, the application name from the `APPNAME` statement appears in the help viewer windows.

See Also

`BROWSE` help file source statement
`FONT` help file source statement
`helpc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

HTOPIC and BTOPIC

Help File Source Statements

Summary

Help topics and glossary topics share a common format. The first line (`HTOPIC` or `BTOPIC`) is mandatory; the second line (`BROWSE`) optional.

The help topic is formatted as follows:

```
HTOPIC <topic ID> <topic title> [keywords]
[BROWSE <sequence name>:<sequence number>]
```

The glossary topic is formatted as follows:

```
BTOPIC <topic ID> <topic title> [keywords]
[BROWSE <sequence name>:<sequence number>]
```

`<topic ID>`

An integer or defined constant that uniquely identifies this topic. For convenience, use `#define` statements in your application's header file to define symbolic names for the topic ID numbers.

`<topic title>`

The title of the help topic. This title appears in the topic window of the help viewer. The title must be enclosed in quotation marks, unless it contains no space characters.

`[keywords]`

Zero or more words that appear in the list of searchable keywords in the help viewer's Search dialog. The keywords are separated by spaces. Each must be enclosed in quotation marks, unless it contains no space characters. This field is optional.

The `BROWSE` line indicates the browse sequence to which the help topic belongs, along with its location within the sequence. It contains the following fields:

`<sequence name>`

The name of the browse sequence. This name must be one of the browse sequence names defined in the file header.

`<sequence number>`

An integer that indicates the placement of the help topic within the browse sequence. Higher-numbered topics appear later in the browse sequence. The sequence numbers are not required to be consecutive. Browse sequence numbers are sorted lexically as ASCII strings, not numerically (so 100 sorts before 20 or 99).

Description

These format statements create help topics (HTOPIC) or glossary topics (`BTOPIC`).

Tip: For your sequence numbers, use multiples of five or ten. This allows you to insert new topics (with sequence numbers that fall between multiples of five or ten) within a sequence without renumbering the old topics. XVT recommends using 5-digit numbers (e.g., 20000).

Note: Browse sequences are available in the Win32 native help viewer. They are also available on all other platforms through the portable help viewer. If you use the Win32 native viewer, you cannot use topics defined with `BTOPIC` in calls to `xvt_help_display_topic`.

See Also

`BODYSTANZA` help file source statement
`BROWSE` help file source statement
`helpc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

#define

helpc Help File Preprocessor Directive

Summary

```
#define identifier "string"
```

Description

This preprocessor directive assigns the string to the identifier. Use this command for substitutions within the help file. For example:

```
#define companyName "XVT Software Inc."
```

The help compiler expands identifiers when it creates the compiled help file.

Summary

```
#define identifier integer
```

Description

Assigns the integer to the identifier. Use this command to create mnemonic names for topic identifiers. For example:

```
#define searchDialog 200
#define searchHelp 200
#define moreSearchHelp 201
```

See Also

`helpc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

#if, #elif, #else, and #endif

helpc Help File Preprocessor Directives

Summary

```
#if expression
#else
#elif <expression>
#endif
```

Description

If `expression` is nonzero, include the following text, up to the next `#endif` or `#else statement`:

```
#else
```

If the preceding `#if`, `#ifdef`, or `#ifndef` statement is not true, include the following text, up to the next `endif` statement:

```
#elif <expression>
```

If the preceding `#if`, `#ifdef`, `#ifndef`, or `#elif` statement is not true and <`expression`> is nonzero, include the following text, up to the next `#elif` or `#endif` statement.

```
#endif
```

End a block of conditional text, started with an `#if`, `#ifdef`, `#ifndef`, `#else`, or `#elif` statement.

See Also

`#ifdef and #ifndef` preprocessor directive
`helpc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

#ifdef and #ifndef

helpc Help File Compiler Preprocessor Directives

Summary

```
#ifdef identifier#ifndef identifier
```

Description

If the `identifier` is defined, either in a `#define` statement or on the help compiler command line, include the following text, up to the next `#endif` or `#else` statement.

```
#ifdef identifier
```

If the identifier is not defined, include the following text, up to the next `#endif` or `#else` statement.

```
#ifndef identifier
```

See Also

`#if, #elif, #else, and #endif` preprocessor directives

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

#include

helpc Help File Compiler Preprocessor Directive

Summary

```
#include "file"
```

Description

This preprocessor directive includes another file in the compilation. Use this command to include header files with topic identifier definitions.

Caution: The **helpc** preprocessor is not intended to be ANSI or K&R compliant. Specifically, it can generate fatal errors when processing the standard C preprocessor keyword "`defined`," or file inclusions of the type `#include <filename>`. To avoid encountering these limitations, do not `#include` or `#scan` **xvt.h**, or any other header files that contain the above mentioned preprocessor constructs.

Since the predefined help topic IDs are in **xvt_help.h**, it is not necessary to `#include` or `#scan` **xvt.h**, only **xvt_help.h**.

You should limit the complexity of the header files you include in your help source. XVT recommends a header that contains only macro definitions of the topic identifiers. To conditionally compile your help source, use the `#ifdef` and `#ifndef` constructs in conjunction with the `-D` **helpc** option.

See Also

`#scan` preprocessor directive
`helpc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

#scan

helpc Help File Compiler Preprocessor Directive

Summary

```
#scan "file"
```

Description

This preprocessor directive scans a file and processes its preprocessor commands. It does not scan or process any other text from the file.

Caution: The **helpc** preprocessor is not intended to be ANSI or K&R compliant. Specifically, it can generate fatal errors when processing the standard C preprocessor keyword "`defined`," or file inclusions of the type `#include <filename>`. To avoid encountering these limitations, do not `#include` or `#scan` **xvt.h**, or any other header files that contain the above mentioned preprocessor constructs. Since the predefined help topic IDs are in **xvt_help.h**, it is not necessary to `#include` or `#scan` **xvt.h**, only **xvt_help.h**.

You should limit the complexity of the header files you include in your help source. XVT recommends a header that contains only macro definitions of the topic identifiers. To conditionally compile your help source, use the `#ifdef` and `#ifndef` constructs in conjunction with the `-D` **helpc** option.

See Also

`#include` preprocessor directive
`helpc`

The"Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

Bitmap (P)

Help File Format Code

Summary

```
\P<pathname>\P
```

Description

This format code inserts a bitmap into the compiled help file. The `<pathname>` is the name of the bitmap file and *must* include path information (if the bitmap is not in the same directory as the source file). The file must be in the MS-Windows BMP format and have a **.bmp** suffix. You must convert images from your native platform format to the BMP format. You can use the XVT Portable Bitmap function calls, or use image translators.

Note: When the help source file is compiled, the bitmap's filename must conform to the conventions of the operating system in use. If you are moving your help source file between platforms during development, it is most convenient if you keep all of your bitmap files in the same directory as the source file, since no path information will be needed in the bitmap file names. At runtime, the bitmaps are included in the compiled help file, so the path information is no longer relevant.

See Also

`HTOPIC and BTOPIC` help file source statements
`helpc`

The "Hypertext Online Help" and the "Portable Images" chapter in the *XVT Portability Toolkit Guide*

Font Change (F)

Help File Format Code

Summary

```
\F<font code>
```

Description

This format code changes the help text font. The <font code> is one of the font codes defined in the font specification statements in the help file header. It is a single character. The font code "0" is the default font if no font change is specified.

Example

The following is a sample font change specification. It specifies that the word "Glossary" is to be displayed using the font identified as code "c"; then, after "Glossary" is displayed, the font changes back to font code "0."

```
fcGlossaryf0
```

See Also

FONT help file source statement
HTOPIC and BTOPIC help file source statements
helpc

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

Hanging Indentation (I)

Help File Format Code

Summary

```
I<indentation>.<hanging_text><tab><paragraph
text><indentation>
```

The distance between the left margin and the indented text, in pixels.

<hanging text>

The text that is not indented (i.e., set to the left margin).

<tab>

A tab character separates the hanging text from the indented text.

<paragraph text>

The indented text.

Description

This format code creates a paragraph with hanging indentation. It is useful for bullet lists and other similar items.

See Also

`FONT` help file source statement
`HTOPIC and BTOPIC` help file source statements
`helpc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

Horizontal Line (V)

Help File Format Statement

Summary

`V<width>`

Description

This format code draws a horizontal line in the topic window. The line extends from the left margin to the right edge of the topic window. The <`width`> specifies the thickness of the line, in pixels.

See Also

`FONT` help file source statement
`HTOPIC and BTOPIC` help file source statements
`helpc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

Hot Button (B)

Help File Format Code

Summary

`B<topic ID>[:string]B<topic ID>`

Identifies which glossary topic to display.

`[:string]`

The text for the glossary topic.

Description

This format code creates a hot button `(hot link)`. When the user clicks the hot button, a glossary topic is displayed. The <`topic ID`> can only be a glossary topic (i.e., a `BTOPIC`); it cannot be an `HTOPIC`. The `string` is separated from the topic identifier by a colon (:). If string is omitted, the title of the help topic is used for the button text.

See Also

`FONT` help file source statement
`HTOPIC and BTOPIC` help file source statements
`Hyperlink (\L)` help file format code
`helpc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

Example

```
BB_RECTANGLE:rectangle B
```

Hyperlink (L)

Help File Format Code

Summary

`L<topic ID>[:string]L<`

`topic ID>`

Identifies which help topic to display in the topic window when the user clicks the hypertext link.

`string`

The text for the link.

Description

This format code creates a hypertext link. The `string` is separated from the topic identifier by a colon (:). If `string` is omitted, the title of the help topic is used for the link text.

See Also

`Hot Button (\B)` help file format code
`HTOPIC and BTOPIC` help file source statements
`helpc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

Example

```
<fcL H_FONT_DIALOG:font dialog L
```

Margin (M)

Help File Format Code

Summary

```
M<margin width>.
```

Description

This format code sets the left margin. The <`margin width>` is specified in pixels. You can omit the trailing period (.) if this statement appears on a line by itself.

See Also

`HTOPIC and BTOPIC` help file source statements
`helpc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

No Word Wrap (N)

Help File Format Code

Summary

```
N
```

Description

This format code turns off word-wrap mode. Subsequent lines of text break only where broken in the help source file. Lines longer than the width of the help topic window are truncated.

See Also

`HTOPIC and BTOPIC` help file source statements
`Word Wrap (\W)` help file format code
`helpc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

Paragraph (A)

Help File Format Code

Summary

`A`

Description

This format code creates a line break when it is within a line. When it is in a line by itself, it causes a paragraph break (inserts a vertical space one and a half lines high).

See Also

`HTOPIC and BTOPIC` help file source statements
`helpc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

Reserved (S) Format Code

Help File Format Code

Summary

`S`

Description

This command is reserved and is currently not supported. You should not use this command in your help source.

Word Wrap (W)

Help File Format Code

Summary

`W`

Description

This format code turns on word-wrap mode. Subsequent text fills the width of the help topic window, wrapping as needed, ignoring line breaks in the help source file.

See Also

`HTOPIC and BTOPIC` help file source statements
`No Word Wrap (\N)` help file format code
`helpc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

Predefined Help IDs

Reserved Topic Identifiers

Description

The following symbols are reserved topic identifiers, which correspond to the items on the predefined Help menu:

Topic ID	Corresponding Item
`XVT_TPC_HELPONHELP`	Information about the help system
`XVT_TPC_KEYBOARD`	Information about special keys
`XVT_TPC_INDEX`	Help index
`XVT_TPC_CONTENTS`	Help table of contents
`XVT_TPC_TUTORIAL`	Application tutorial information
`XVT_TPC_ONVERSION`	Application version information
`XVT_TPC_GLOSSARY`	Glossary of terms

The **xvt_help.h** file defines these reserved topic identifiers. It may also contain additional identifiers that are not listed here.

If you use the standard Help menu, you must provide help text for each of these topics. You can either write the text yourself, or use pre-written text for some of the topics.

Note: Since the topics `XVT_TPC_INDEX`, `XVT_TPC_CONTENTS`, and `XVT_TPC_TUTORIAL` are necessarily dependent on your application, XVT provides no pre-written text for them.

The following symbols are reserved topic identifiers, which correspond to predefined XVT dialogs:

Topic ID	Corresponding Dialog
XVT_TPC_FILE_OPEN	xvt_dm_post_file_open
XVT_TPC_FILE_SAVE	xvt_dm_post_file_save
XVT_TPC_ASK	xvt_dm_post_ask
XVT_TPC_NOTE	xvt_dm_post_note
XVT_TPC_ERROR	xvt_dm_post_error
XVT_TPC_WARNING	xvt_dm_post_warning
XVT_TPC_STRING_PROMPT	xvt_dm_post_string_prompt
XVT_TPC_FONT_SEL	xvt_dm_post_font_sel
XVT_TPC_PAGE_SETUP	xvt_dm_post_page_setup
XVT_TPC_MESSAGE	xvt_dm_post_message

When the user requests help while an XVT predefined dialog is active, XVT sends an `E_HELP` event with the corresponding topic ID to the task event handler. The `tid` member of the help event structure is set to one of the predefined IDs above. If the help file contains a topic pertaining to that ID, the help viewer displays it.

See Also

Predefined Help Topics
`helpc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

Predefined Help Topics

Help Topic Text

XVT provides help topic text for several of the predefined help IDs, including `XVT_TPC_HELPONHELP`, `XVT_TPC_KEYBOARD`, and others. The file **xvt_help.csh** contains these topics.

You can include this file in your help source file to provide default help information for the reserved topic symbols. However, the predefined help topic text in the **xvt_help.csh** header is incomplete, so you will most likely want to override these topics in your own help source file.

Tip: To include all of the XVT-provided topics in **xvt_help.csh**, add this line to the end of your help source file:

```
#include "xvt_help.csh"
```

Tip: To include some, but not all, of the topics in **xvt_help.csh**:

1. Provide your own help topic text for the topics you wish to customize, in your help source file.
2. At the bottom of your help source file, undefine all reserved topic identifiers whose XVT-provided topic text you want to omit, and redefine them as -1. The compiler skips any help topics that have a topic identifier of -1.
3. Add this line to the end of your help source file:

```
#include "xvt_help.csh"
```

Note: If your code includes the file **xvt_help.csh**, you must place the "`#scan "xvt_help.h""` statement in the order shown in the following example.

The **xvt_help.h** file defines the topic identifiers referenced by **xvt_help.csh**.

Example

Suppose you want to provide custom help text for the `XVT_TPC_KEYBOARD` topic, but use the XVT-provided text for all other reserved topics. Your help source file would contain the following text:

```
HTOPIC XVT_TPC_KEYBOARD "Special Keyboard Commands"
' your help text
…
' at the end of the file:
#scan "xvt_help.h"
#undef XVT_TPC_KEYBOARD
#define XVT_TPC_KEYBOARD -1
#include "xvt_help.csh"
```

See Also

Predefined Help IDs
`helpc`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

Tools

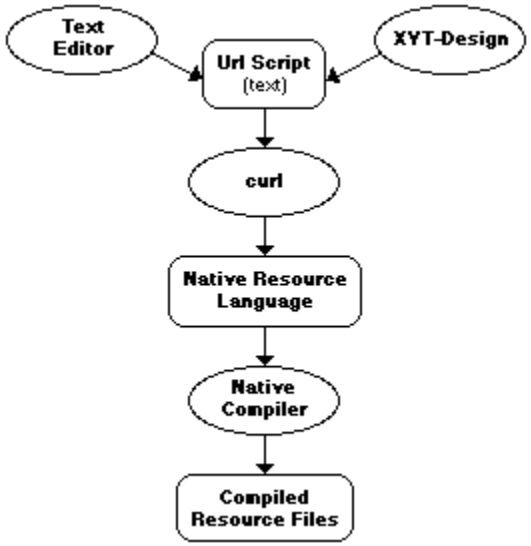
curl
errscan
helpc
maptabc

curl

XVT resource compiler

Summary

As this figure shows, you can manually code the resources into a text file in URL, or you can develop the resources with XVT-Design (a utility product that interactively produces URL).



The **curl** compiler produces a text file that must be compiled by the platform-specific resource compiler on most platforms. This is the **curl** command line format:

curl -r *resource_type [options] url_file*

Running **curl** with no arguments produces a usage line and a list of resource types and options.

Command Line Options

`-f filename`

Specifies the name of an **options** file. Command line options are read from this file as if they were given on the command line in place of the `-f` option. Options in the file can be given on separate lines, on one line separated by blanks, or on any combination of the two.

`-d symbol`

Define a symbol. This option has the same effect at the `#define` preprocessor statement. (See Preprocessor Symbol Definition below.)

`-d symbol=text`

Define a symbol with substitution text. This option has the same effect at the `#define` preprocessor statement. (See Preprocessor Symbol Definition below.)

`-h`

Help (displays extended option list).

`-i path`

Add directory to include search path.

`-k`

Keep partial resource file (after error has occurred).

`-m char_width x char_height`

Character to pixel conversion (default: 6 x 13). This option is useful only for converting from URL to a native resource language (e.g., Motif's UIL). (See Units Conversion below.)

`-nb`

Do not output binary-encoded resources.

`-np`

Allow non-portable constructs. (See Non-portable Contructs below.)

`-o filename`

Specify output name (and path), but not extension.

`-s[n]`

Controls the level of diagnostic output according to `n`, an optional digit of 0-4. If you don't use the `-s` option, output defaults to level 2. If you use the `-s` option without a digit, output defaults to level 1. The optional `n` digit specifies the diagnostic output like this:

0 - No output
1 - Errors only
2 - Normal (errors, warnings, and status indicators)
3 - Verbose (same as 2 plus additional information)
4 - Trace/debug (same as 3 plus debug information).

`input_filename`

The first argument that is not identified as an option will be assumed to be the URL file you want to compile. It must end with a **.url** extension. **curl** allows the filename to be in lower- or uppercase; however, the host operating system may be case-sensitive.

Resource types

`-r rcnt`

MS-Windows NT-compatible RC (XVT/Win32)

`-r mtf12`

Motif 1.2-compatible UIL (XVT/XM).

`-r rez`

Macintosh Rez (XVT/Mac).

`-r dpr`

XVT-Design project file.

Preprocessor Symbol Definition

The `-d` command line option lets you define a symbol, with or without substitution text, similar to most C/C++ compilers.

Tip: To define a symbol:

On the **curl** command line, type `-d` followed by the symbol name (the white space between the `-d` and the symbol name is optional). For example:

```
-d NO_STD_FILE_MENU
```

defines the symbol `NO_STD_FILE_MENU` (removing the standard XVT file menu).

Note: The symbol `NO_STD_ABOUT_BOX` can be used if no XVT About box is to be displayed.

Tip: To define a symbol with substitution text:

On the **curl** command line, type `-d` followed by the symbol name, then an equal sign (=), and finally the substitution text. There must be no white space before or after the equal sign or within the substitution text. For example:

```
-d LIBDIR=w32_ptkmy_lib
```

causes all subsequent appearances of `LIBDIR` in preprocessed lines to be replaced by w32_ptkmy_lib. `LIBDIR` is replaced only if it appears as a token; it is not replaced if it appears in a comment, a string, or as part of a larger token.

Note: The `LIBDIR` macro is used to specify the pathname for XVT-supplied platform-specific resources, such as bitmaps, icons, pointers, and cursors. By default, XVT supplies a pathname, which you can override if you have located these resources in another directory.

Units Conversion

For those platforms that do not naturally work in chars or semichars, the `-m` command line option lets you specify the character cell width and height. **curl** uses these numbers whenever it translates between chars or semichars and pixel units.

Use the `-m` option to change the character cell dimensions either to facilitate a better layout on a particular platform or to ensure portability among other platforms. The format of this command line option is as follows:

```
-m char_cell_widthXchar_cell_height
```

`char_cell_width`

The width of the character cell in pixels. It must be a positive integer.

`X`

Separates the width from the height. It must be typed literally and can be upper- or lowercase.

`char_cell_height`

The height of the character cell in pixels. It must be a positive integer.

Default Character Cell Dimensions

If you do not specify the `ëm` option, the **curl** compiler uses these character cell dimensions: 6 pixels wide and 13 pixels high.

The 6 x 13 character cell size is a compromise value to allow portability among the XVT/Mac and XVT/Win32 products.

Non-portable Constructs

If you use non-portable constructs in the URL file, `curl` can generate error messages about them. Currently, the only construct to which this applies is the use of menu items on a menubar. To suppress these error messages, use the `ënp CURL` command line option.

GUI Application

If you are a source customer, you can build **curl** as a GUI application (**curla.exe** on XVT/Win16, XVT/Win32 and XVT/PM, **curl.app** on XVT/Mac and **curl_app** on XVT/XM). (XVT provides built GUI **curl** executables with XVT/Win16 and XVT/Mac.) Each time the user initiates a scan of a new URL file from the menu, the GUI version of **curl** attempts to read the default options file, **curl.opt** (located in the startup directory).

The default options file can contain any of the options available in the command line version. After processing this file, **curl** prompts you for additional directories and files to scan. Cancelling the next directory selection generates output files.

See Also

`NO_STD_*_MENU Values`
`URL Statements`

For more information on executing **curl** on your platform and to learn how to build **curl**, see the *XVT Platform-Specific Books*.

Example

Here is a sample **curl** command line using the units conversion option:

```
curl -r rcnt -m 9x16 -i ....include dlg.url
```

In this example, **curl** generates resources for Win32 with a character cell width of 9 pixels and a character cell height of 16 pixels.

errscan

XVT Error Code Scan Tool

Summary

XVT supplies an **errscan** tool, which can examine your application code and perform the following operations:

- Find all instances of error signaling (`xvt_errmsg_sig` and `xvt_errmsg_sig_if` calls)
- Find all predefined error messages (defined with `xvt_errmsg_def_*` macros)
- Generate the error message text file **ERRCODES.TXT**
- Generate the error codes definition file **xvt_perr.h**

As a result, you don't have to manually collect and maintain a list of error codes and associated messages.

The **errscan** tool uses the message suffix and number supplied by each `xvt_errmsg_sig` or `xvt_errmsg_def_*` macro to build an error code `#define`. Consequently, the suffix and number must be unique within a given message category.

The **errscan** tool warns you of any duplication and/or syntax errors. However, its syntax checking is not as sophisticated as a compiler's. In particular, **errscan** does not use the **cpp** pre-processor, and it makes several assumptions about the error signaling call.

You can build **errscan** either as a command line utility or as an interactive application. XVT provides **errscan** both as a source file and an executable, in the directories **ptk/toolsrc/errscan/errscan.c** and **ptk/bin/errscan**.

Note: Source customers can also run **errscan** on XVT source code.

Command Line Options

errscan is executed as a command line utility. (This is the default for all XVT-supported platforms except XVT/Mac.) The command line has the following form:

```
    errscan [-f filename] [-o filename] [-h filename]
input_filename ...
-f filename
```

Specifies an option file containing filenames, one per line, or options. `filename` cannot contain multibyte characters.

`-o filename`

Specifies the output message text file (the default is **ERRCODES.TXT**). `filename` cannot contain multibyte characters.

`-h filename`

Specifies the output header file (the default is **xvt_perr.h**).

`input_filenames`

List of filenames to scan for error messages.

GUI Application

You can build **errscan** as a GUI application (**errscana.exe** on XVT/Win16, XVT/Win32 and XVT/PM, **errscan.app** on XVT/Mac and **errscan_app** on XVT/XM). (XVT provides a built GUI **errscan** executable with XVT/Mac.) Each time the user initiates a scan from the menu, the GUI version of **errscan** attempts to read the default options file, **errscan.opt** (located in the startup directory).

The default options file can contain any of the options available in the command line version. After processing this file, **errscan** prompts you for additional directories and files to scan. Cancelling the next directory selection generates output files.

See Also

`xvt_errmsg_sig`
`xvt_errmsg_sig_if`

For more information on executing errscan on your platform and to learn how to build errscan, see the XVT Platform-Specific Books.

---

helpc

Help Compiler

Summary

XVT's help compiler, **helpc**, compiles your help source files into a compact, binary format or into a native help source text file. The portable binary file format allows the help system to rapidly access your help text while your application executes. If you use compatible character code sets, the compiled help file is portable across all XVT platforms; you can use one file on each platform without separate recompilation.

The help compiler operates in essentially the same manner, and uses the same command-line options, on all XVT platforms.

Command Line Options

Tip: To run the help compiler, use the following command-line format:

```
helpc [-o filename] [-v level] [-P]
     [-f {xvt | win}] [-i include_directory]
    [-d symbol] input_filename ...
```

The options have the following effects:

`-o filename`

Use `filename` to name the binary output file. If this option is not used, the output file has the same name as the source file, with the extension **.csc**.

`-v level`

Set the verbosity level for the compiler's diagnostic messages. level is an integer from 0 (no diagnostic messages) to 5 (verbose diagnostic output).

`-P`

Preprocess the source file: perform macro expansion and conditional compilation only (see Using the Preprocessor Option).

`-f {xvt | win}`

Choose the output file format:

`xvt`XVT help system (portable) format
`win`Win32 native help viewer format

If this option is not specified, the XVT help system format is used.

`-i include_directory`

Specify an additional directory to search for `#include` files and bitmap files. `include_directory` is the pathname of the directory. This option can be used more than once, to specify multiple directories.

`-d symbol`

Define a symbol. This option has the same effect as the #define `preprocessor statement.`

`-d symbol=text`

Define a symbol with substitution text. This option has the same effect as the `#define` preprocessor statement.

`input_filename`

The name of the help source file. No restrictions are placed on this file name, except that the default (and suggested) extension is **.csh**.

Using the Preprocessor Option

The `-P` preprocessing option instructs the **helpc** compiler to process a help source file and generate a source file instead of a compiled help file. The source file has the macros replaced and the `#include` files included. It is still in text format, not in binary.

With this option, you can use symbolic names for the topic and resource identifiers in the topic association file, rather than using plain integers. You can include your header files that define symbolic names for help topic and resource ID numbers, and use these names to construct your association file.

Manifest Constants

The help compiler always predefines the following symbols before compiling the help source:

`__helpc__`

Defined when the compiler is running. When the compiler was built, this symbol was set to the value of `XVT_HELP_VERSION`. You can conditionally compile your header files based on the existence of this symbol.

`HELP_FMT_XVT`
`HELP_FMT_WIN`

One of these is defined by the compiler based on the value of the `-f` command line option. This allows you to conditionally compile your help source based on the format of the output that the help compiler is generating. These symbols are defined as follows:

Symbol Defined	-f Option	Description
`HELP_FMT_XVT`	`XVT`	XVT portable help file format
`HELP_FMT_WIN`	`WIN`	Win32 file format

GUI Application

If you are a source customer, you can build **helpc** as a GUI application (**helpca.exe** on XVT/Win32, **helpc.app** on XVT/Mac

and **helpc_app** on XVT/XM). (XVT provides a built GUI **helpc** executable with XVT/Mac.) Each time the user initiates a scan from the menu, the GUI version of **helpc** attempts to read the default options file, **helpc.opt** (located in the startup directory). The default options file can contain any of the options available in the command line version.

Implementation Note

Help source files are constrained by the following limitations:

- The text for each topic must be no more than 64K bytes
- Individual bitmaps must be no more than 32K bytes in size
- Bitmaps can be black and white, 16-color or 256-color
- Bitmaps must be in Win32 BMP format, and should have a resolution of 96 dots per inch
- The total size of the help source file must be no more than 99,999,999 bytes
- Each topic can have no more than 16 keywords
- Tokens in the help file must be no more than 256 bytes each (a token is delimited by white space or punctuation)

The Win32 native help viewer (**Winhelp**) cannot display 256-color bitmaps. 256-color bitmaps are restricted in size by the 32KB limit mentioned above. For example, a 200- by 175-pixel, 256-color bitmap exceeds the 32KB limit, and thus cannot be used in a help file.

See Also

Help File Statements

For more information on executing helpc on your platform and to learn how to build **helpc**, see the *XVT Platform-Specific Books*.

Example

This code conditionally compiles a topic for the native MS-Windows help viewer:

```
#ifdef HELP_FMT_WIN
HTOPIC NativeWinHelpTopic "Native Topic"
This topic will only be compiled for the native
     MS-Windows help viewer.
 #endif
```

maptabc

XVT Character Codeset Map Table Compiler

Summary

XVT supplies a map-table compiler **maptabc**, which reads a source file containing a description of a codeset mapping into Unicode, and generates a binary version of the mapping table. The binary codeset table (**.bct**) file is used by the function xvt_str_create_codeset_map, which in turn creates an XVT_CODESET_MAP object. The object defines a mapping of characters from one codeset to another—either across platforms or on the same platform.

The codeset map-table source file has the following format:

1. '#' begins a comment that continues to the end of the line.
2. A mapping line consists of three tab- or space-separated columns.

Column one is the character code of the local codeset specified in hex (NxNN or NxNNNN). Column two is the character code of the Unicode codeset specified in hex (NxNNNN or NxNNNN+NxNNNN). Column three is the Unicode name (follows a comment symbol, '#').

The character 0x00 is always forced to map to 0x0000 in Unicode. By default, the control characters 0x01–0x19 are mapped to the characters 0x0001–0x0019, respectively, in Unicode, unless overridden by the map source file.

The .**bct** files for the basic codesets (for languages that XVT provides translations to) are provided by XVT in the **...bin/codemaps** directory.

These files are provided:

8859-1.bct	ISO Latin-1 codeset mapping to Unicode
cp1250.bct	MS-Windows code page 1250 mapping to Unicode
cp1252.bct	MS-Windows code page 1252 mapping to Unicode
MJapan.bct	Mac-Japanese codeset mapping to Unicode
MRoman.bct	Mac-Roman codeset mapping to Unicode
sjis.bct	Japanese Shift-JIS codeset mapping to Unicode

Command Line Options

maptabc is executed as a command line utility. (This is the default or all XVT-supported platforms except XVT/Mac.) The command line has the following form:

maptabc [-o filename] input_filename

-o filename

Specifies the binary output mapping file (the default is to change the input_filename extension to **.bct**).

input_filename

Text file specifying a codeset mapping to Unicode.

GUI Application

You can build **maptabc** as a GUI application (**maptabca.exe** on XVT/Win32, **maptabc.app** on XVT/Mac and **maptabc_app** on XVT/XM).

(XVT provides a built GUI **maptabc.app** executable with XVT/Mac.)

In the GUI application, select the input text file to compile by selecting "Open..." from the "File" menu. Select the file to compile from the Open File dialog. The default output file name is automatically set when the input file is selected. However, you can change the output binary file name by selecting "Save As..." from the "File" menu. Enter the file name in the Save File dialog. To compile the file select "Compile" from the "Compile" menu.

The compiler will show a status window displaying the current file being processed and the number of lines currently processed. When the compile is complete, a dialog box appears displaying the number of characters found in the codeset mapping table.

You can reset the input file (and output file) and compile as many files as you need to.

See Also

XVT_CODESET_MAP
xvt_str_create_codeset_map
xvt_str_destroy_codeset_map
xvt_str_translate_codeset

For more information on executing maptabc on your platform and to learn how to build maptabc, see the XVT Platform-Specific Books.

Unicode is a trademark of Unicode Inc. (The Unicode Consortium) Further information on the Unicode Standard, The Unicode Consortium, and various codeset-to-Unicode mappings can be found on the Unicode public file server:

ftp://unicode.org/pub/

Conventions Used in This Reference

To accompany this release, the *XVT Portability Toolkit Reference* (*PTK Reference*) is now online. XVT has gone to this format to make reference information clearer, easier to find, and more usable. XVT takes pride in its documentation, and continually seeks to improve it. If you find a documentation error, please contact Customer Support. They will forward your suggestion to XVT's documentation team.

The following typographic and code conventions indicate different types of information.

General Conventions

`code`

This typestyle is used for code and code elements (names of functions, data types and values, attributes, options, flags, events, and so on). It also is used for environment variables and commands.

bold

Bold type is used for filenames, directory names, and program names (utilities, compilers, and other executables).

italics

Italics are used for emphasis and the names of documents.

Tip: This marks the beginning of a procedure having one or more steps. Tips can help you quickly locate "how-to" information.

Note: An italic heading like this marks a standard kind of information: a Note, Caution, Example, Tip, or See Also (cross-reference).

Code Conventions

`<non-literal element>`, **`non_literal_element`**, or *non_literal_element*

Angle brackets, bold code font, or italics indicate a non-literal element, for which you would type a substitute.

`[optional element]`

Square brackets indicate an optional element.

`...`

Ellipses in data values and data types indicate that these values and types are opaque. You should *not* depend upon the actual values and data types that may be defined.

XVT Portable Attributes

ATTR_APP_CTL_COLORS
ATTR_APP_CTL_FONT_RID
ATTR_APPL_NAME_RID
ATTR_BACK_COLOR
ATTR_COLLATE_HOOK
ATTR_CTL_BUTTON_HEIGHT
ATTR_CTL_CHECK_BOX_HEIGHT
ATTR_CTL_EDIT_TEXT_HEIGHT
ATTR_CTL_HORZ_SBAR_HEIGHT
ATTR_CTL_RADIOBUTTON_HEIGHT
ATTR_CTL_STATIC_TEXT_HEIGHT
ATTR_CTL_VERT_SBAR_WIDTH
ATTR_DBLFRAME_HEIGHT
ATTR_DEBUG_FILENAME
ATTR_DEFAULT_PALETTE_TYPE
ATTR_DISPLAY_TYPE
ATTR_DOC_STAGGER_HORZ
ATTR_DOC_STAGGER_VERT
ATTR_DOCFRAME_HEIGHT
ATTR_DOCFRAME_HEIGHT
ATTR_ERRMSG_FILENAME
ATTR_ERRMSG_HANDLER
ATTR_EVENT_HOOK
ATTR_FONT_CACHE_SIZE
ATTR_FONT_DIALOG
ATTR_FONT_MAPPER
ATTR_FRAME_HEIGHT
ATTR_FRAME_WIDTH
ATTR_HAVE_MOUSE
ATTR_HELP_CONTEXT
ATTR_HELP_HOOK
ATTR_ICON_HEIGHT
ATTR_ICON_WIDTH
ATTR_KEY_HOOK
ATTR_MEMORY_MANAGER
ATTR_MENU_HEIGHT
ATTR_MULTIBYTE_AWARE
ATTR_NATIVE_GRAPHIC_CONTEXT
ATTR_NATIVE_WINDOW
ATTR_NUM_TIMERS
ATTR_PRINTER_HEIGHT
ATTR_PRINTER_HRES
ATTR_PRINTER_VRES
ATTR_PRINTER_WIDTH
ATTR_PROPAGATE_NAV_CHAR
ATTR_RESOURCE_FILENAME
ATTR_R40_TXEDIT_BEHAVIOR
ATTR_SCREEN_HEIGHT
ATTR_SCREEN_HRES

```
ATTR_SCREEN_VRES
ATTR_SCREEN_WIDTH
ATTR_SCREEN_WINDOW
ATTR_SUPPRESS_UPDATE_CHECK
ATTR_TASK_WINDOW
ATTR_TASKWIN_TITLE_RID
ATTR_TITLE_HEIGHT
ATTR_XVT_CONFIG
```

ATTR_APP_CTL_COLORS

Description

This attribute specifies the address of an `XVT_COLOR_COMPONENT` array to use as the application default control colors. Unlike the analogous attribute for fonts which takes a resource ID, this attribute takes a memory address. The application can safely deallocate the `XVT_COLOR_COMPONENT` array during the process of an `E_DESTROY` event for the task window.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	The address of the application default control colors array (of type `XVT_COLOR_COMPONENT*`), or `NULL` if not set.
`xvt_vobj_set_attr` effect:	Register the address of `XVT_COLOR_COMPONENT` array to be used as an application default for rendering controls.
`xvt_app_create` use:	Must use before. If the application changes the contents of the array after `xvt_app_create` has been called, the effect is undefined.
Default value:	`NULL`. This default value indicates that no application default colors have been defined.

If the application uses this attribute, the specified `XVT_COLOR_COMPONENT` array will be used as the application default control colors. An application uses this attribute as follows:

In the application source code before calling `xvt_app_create`:

```
static XVT_COLOR_COMPONENT app_colors[] = {
    {XVT_COLOR_FOREGROUND, COLOR_BLACK},
    {XVT_COLOR_BLEND, COLOR_WHITE},
    {XVT_COLOR_BACKGROUND, COLOR_BLUE},
    {XVT_COLOR_NULL, 0}
 };
xvt_vobj_set_attr(NULL_WIN, ATTR_APP_CTL_COLORS,
     (long)app_colors);
```

The individual colors specified in this array will be used in all controls which do not already have corresponding colors specified for them and whose container also does not define the corresponding colors. No functionality is defined in this specification to allow the application to switch the application default control colors after `xvt_app_create` has been called.

See Also

`XVT_COLOR_COMPONENT`
`xvt_ctl_set_colors`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`
`xvt_win_set_ctl_colors`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

ATTR_APP_CTL_FONT_RID

Description

This attribute specifies a font resource ID to use as the application default control font.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	The value of the application default control font resource ID.
`xvt_vobj_set_attr` effect:	Establish the URL resource ID of the font resource used as an application default font for rendering controls. If the application has not defined this resource in URL, the application behaves as if this attribute has not been set.
`xvt_app_create` use:	Must use before
Default value:	Zero. This default value indicates that no application default font resource ID has been defined.

If the application uses this attribute, the specified font resource ID will be used as the application default control font. An application uses this attribute as follows:

In the application header file:

```
#define MY_APP_CTL_FONT 10
```

In the application source code before calling `xvt_app_create`:

```
xvt_vobj_set_attr(NULL_WIN,
ATTR_APP_CTL_FONT_RID,
(long)MY_APP_CTL_FONT);
```

In the application URL file:

```
font MY_APP_CTL_FONT helvetica 12
```

This font is used in all controls which do not already have fonts specified for them, and whose containers do not have default control fonts specified for them. No functionality is defined in this specification to allow the application to switch the application default control font after `xvt_app_create` has been called.

See Also

`xvt_ctl_set_font`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`
`xvt_win_set_ctl_font`

The "Controls" and the "Fonts and Text" chapters in the *XVT Portability Toolkit Guide*

ATTR_APPL_NAME_RID

Description

This attribute can be set to the resource ID of the multibyte string containing the value of `appl_name` for use in the `XVT_CONFIG` structure. In `xvt_app_create`, this attribute is tested for non-zero by XVT and the resource loaded into `XVT_CONFIG`. The existing pointer `appl_name` in `XVT_CONFIG` will be replaced. This attribute allows the application to externalize the application name in resources for localization.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Gets resource ID of `appl_name`
`xvt_vobj_set_attr` effect:	Sets resource ID of `appl_name`

`xvt_app_create` use:	Must use before
Default value:	Zero

See Also

`ATTR_TASKWIN_TITLE_RID`
`ATTR_XVT_CONFIG`
`XVT_CONFIG`
`xvt_app_create`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`

The "Multibyte Character Sets and Localization" chapter in the *XVT Portability Toolkit Guide*

ATTR_BACK_COLOR

Description

The system-wide window background color as set by the user. Applications wishing to honor the user's settings can retrieve this color and use it in their calls to `xvt_dwin_clear`. Be sure not to confuse this with the XVT drawing tools background color.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	The user's choice of window background color
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

`COLOR`
`xvt_dwin_clear`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`

Example

```
xvt_dwin_clear(win, (COLOR) xvt_vobj_get_attr(NULL_WIN,
ATTR_BACK_COLOR));
```

ATTR_COLLATE_HOOK

Description

This attribute holds the pointer to the application-supplied string collation function. It controls whether or not an application-supplied string collation function is used when the application calls `xvt_str_collate`, `xvt_str_collate_ignoring_case`, or `xvt_slist_add_sorted`.

The value of the attribute is a pointer to the collation function. This function receives two multibyte string pointers and then determines their collation order for a particular locale. To register an application-supplied string collation function with XVT, use `xvt_vobj_set_attr`. To retrieve the current application-supplied string collation function pointer, use `xvt_vobj_get_attr`.

If you set this attribute to `NULL`, or if you do *not* set it, XVT uses the default string collation function when the above functions are called. Any application-supplied string collation function must use the `XVT_COLLATE_FUNCTION` signature. Your collation function must return -1 if the first string comes before the second string, zero if they are equal, and 1 if the first string comes after the second string.

Prototype:

```
typedef long (* XVT_COLLATE_FUNCTION) (const
    char *mbs1, const char *mbs2)
```

Uses `win` argument:	No
`xvt_vobj_get_attr` returns;	Pointer to application-supplied string collation function or `NULL`
`xvt_vobj_set_attr` effect:	Sets the string collation function pointer
`xvt_app_create` use:	Can use either before or after
Default value:	`NULL`

See Also

`XVT_COLLATE_FUNCTION`
`xvt_slist_add_sorted`
`xvt_str_collate\`
`xvt_str_collate_ignoring_case`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`

The "Multibyte Character Sets and Localization" chapter in the *XVT Portability Toolkit Guide*

ATTR_CTL_BUTTON_HEIGHT

Description

The button height (in pixels) most appropriate for a platform, based on the system default control font. The optimal button width depends on the width of its label.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Button height
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

`WIN_DEF`
`xvt_ctl_create`
`xvt_ctl_create_def`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`
`xvt_win_create_def`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

ATTR_CTL_CHECK_BOX_HEIGHT

Description

The button height (in pixels) most appropriate for a platform, based on the system default control font. The optimal button width depends on the width of its label.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Button height
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

`WIN_DEF`
`xvt_ctl_create`
`xvt_ctl_create_def`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`
`xvt_win_create_def`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

ATTR_CTL_EDIT_TEXT_HEIGHT

Description

The edit control height most appropriate for a platform, in pixels.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Edit control height
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

`WIN_DEF`
`xvt_ctl_create`
`xvt_ctl_create_def`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`
`xvt_win_create_def`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

ATTR_CTL_HORZ_SBAR_HEIGHT

Description

The horizontal scrollbar thickness most appropriate for a platform, in pixels. This value is the same as the thickness of the horizontal scrollbars that are created by specifying `WSF_HSCROLL` when creating a window.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Scrollbar thickness
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

```
ATTR_CTL_VERT_SBAR_WIDTH
WIN_DEF
WSF_* Options Flags
xvt_ctl_create
xvt_ctl_create_def
xvt_vobj_get_attr
xvt_vobj_set_attr
xvt_win_create_def
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

ATTR_CTL_RADIOBUTTON_HEIGHT

Description

The radio button height most appropriate for a platform, in pixels. The optimal radio button width depends on the width of its label.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Radio button height
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

```
WIN_DEF
xvt_ctl_create
xvt_ctl_create_def
xvt_vobj_get_attr
xvt_vobj_set_attr
xvt_win_create_def
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

ATTR_CTL_STATIC_TEXT_HEIGHT

Description

The default static text control height most appropriate for a platform, in pixels.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Static text
`xvt_vobj_set_attr` effect:	Illegal

`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

`WIN_DEF`
`xvt_ctl_create`
`xvt_ctl_create_def`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`
`xvt_win_create_def`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

ATTR_CTL_VERT_SBAR_WIDTH

Description

The vertical scrollbar thickness most appropriate for a platform, in pixels. This value is the same as the thickness of vertical scrollbars that are created by specifying `WSF_VSCROLL` when creating a window.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Scrollbar thickness
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

```
ATTR_CTL_HORZ_SBAR_HEIGHT
WIN_DEF
WSF_* Options Flags
xvt_ctl_create
xvt_ctl_create_def
xvt_vobj_get_attr
xvt_vobj_set_attr
xvt_win_create_def
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

ATTR_DBLFRAME_HEIGHT

Description

The thickness in pixels of a horizontal border of a double-border window. You can use this value to calculate the outer size of a window, given its client area.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Border thickness in pixels
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

```
ATTR_DBLFRAME_WIDTH
WIN_DEF
xvt_ctl_create
xvt_vobj_get_attr
xvt_vobj_set_attr
xvt_win_create_def
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

ATTR_DBLFRAME_WIDTH

Description

The thickness in pixels of a vertical border of a double-border window. You can use this value to calculate the outer size of a window, given its client area.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Border thickness in pixels
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

`ATTR_DBLFRAME_HEIGHT`
`WIN_DEF`
`xvt_ctl_create`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`
`xvt_win_create_def`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

ATTR_DEBUG_FILENAME

Description

The name of the debugging output file used by XVT.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	A pointer to a static buffer containing the current debug filename, which is "`debug`" by default
`xvt_vobj_set_attr` effect:	Passing a pointer to a string containing the new debugging filename causes calls to `xvt_debug_printf` or `xvt_debug` to output to this file, if the file exists in what the XVT application considers to be the "current" directory
`xvt_app_create` use:	Can use either before or after
Default value:	"debug"

See Also

```
xvt_debug
xvt_debug_printf
xvt_vobj_get_attr
xvt_vobj_set_attr
```

The "Diagnostics and Debugging" chapter in the *XVT Portability Toolkit Guide*

ATTR_DEFAULT_PALETTE_TYPE

Description

The type of the default `XVT_PALETTE` object created during XVT initialization (when `xvt_app_create` is called). This "default palette" is the palette object used when no other palette has been specified for the target window of a display operation (see `xvt_vobj_set_palet`).

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	`XVT_PALETTE_TYPE`
`xvt_vobj_set_attr` effect:	Changes the type of the default palette created during XVT initialization
`xvt_app_create` use:	Must use before
Default value:	`XVT_PALETTE_STOCK`; or it can be set to any other `XVT_PALETTE_TYPE` before calling `xvt_app_create`

See Also

```
XVT_PALLETE_* Values
xvt_vobj_get_attr
xvt_vobj_set_attr
xvt_vobj_set_palet
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

ATTR_DISPLAY_TYPE

Description

The system's hardware display color capabilities. Check this attribute to determine whether your application can display color or grayscale graphics.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	`XVT_DISPLAY_TYPE` value
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

`XVT_DISPLAY_* Values`
`xvt_palet_create`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

ATTR_DOC_STAGGER_HORZ

Description

Recommended horizontal document window cascading offset.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Offset in pixels
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

`ATTR_DOC_STAGGER_VERT`
`xvt_ctl_create_def`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`
`xvt_win_create`
`xvt_win_create_def`

The "Windows" chapter in the *XVT Portability Toolkit Guide*

ATTR_DOC_STAGGER_VERT

Description

Recommended vertical document window cascading offset.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Offset in pixels
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

`ATTR_DOC_STAGGER_HORZ`
`xvt_ctl_create_def`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`
`xvt_win_create`
`xvt_win_create_def`

The "Windows" chapter in the *XVT Portability Toolkit Guide*

ATTR_DOCFRAME_HEIGHT

Description

The thickness in pixels of a horizontal border of a resizable window. You can use this value to calculate the outer size of a window, given its client area.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Border thickness in pixels
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

`ATTR_DOCFRAME_WIDTH`
`xvt_ctl_create`
`xvt_ctl_create_def`
`xvt_vobj_get_attr`
`xvt_vobj_get_client_rect`
`xvt_vobj_get_outer_rect`
`xvt_vobj_set_attr`
`xvt_vobj_translate_points`

The "Windows" chapter in the *XVT Portability Toolkit Guide*

ATTR_DOCFRAME_WIDTH

Description

The thickness in pixels of a vertical border of a resizable window. You can use this value to calculate the outer size of a window, given its client area.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Border thickness in pixels
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

`ATTR_DOCFRAME_HEIGHT`
`xvt_ctl_create`
`xvt_ctl_create_def`
`xvt_vobj_get_attr`
`xvt_vobj_get_client_rect`
`xvt_vobj_get_outer_rect`
`xvt_vobj_set_attr`
`xvt_vobj_translate_points`

The "Windows" chapter in the *XVT Portability Toolkit Guide*

ATTR_ERRMSG_FILENAME

Description

This attribute holds the filename of the (customized) error message file. The **errscan** utility creates the error message file, but you can modify (e.g., translate) the contained text to customize messages for a given application.

Error handlers perform message retrieval, using the `XVT_ERRMSG` object interface. If no error message file is found, XVT provides hardcoded English error messages for the basic, standard messages. Other messages are then represented by message number.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Current message filename
`xvt_vobj_set_attr` effect:	Replaces the message filename used by subsequent error messaging
`xvt_app_create` use:	Can use either before or after
Default value:	`ERRCODES.TXT`

See Also

`ATTR_ERRMSG_HANDLER`
`XVT_ERRMSG`
`xvt_errmsg_*`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`
`errscan`

The "Diagnostics and Debugging" chapter in the *XVT Portability Toolkit Guide*

ATTR_ERRMSG_HANDLER

Description

This attribute holds the pointer to an application-supplied, permanent error handler. This permanent error handler is called for any error message signal not caught by the temporary handler pushed using a `xvt_errmsg_push_handler` call, prior to the XVT-provided "last chance" error handler.

The handler must use the `XVT_ERRMSG_HANDLER` signature, and return `TRUE` if it handled the error, `FALSE` to pass this error to the "last chance" error handler. The default value of this attribute is `NULL`.

Note: The XVT-provided "last chance" event handler cannot be queried or called directly. However, you can prevent it from being invoked by establishing an application error handler that always returns `TRUE`.

Prototype:

```
typedef BOOLEAN (* XVT_ERRMSG_HANDLER)
    (XVT_ERRMSG err, DATA_PTR context)
```

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	The current permanent error handler pointer
`xvt_vobj_set_attr` effect:	Replaces the previous permanent error handler pointer with a new one
`xvt_app_create` use:	Can use either before or after
Default value:	`NULL`

See Also

```
ATTR_ERRMSG_FILENAME
XVT_ERRMSG
XVT_ERRMSG_HANDLER
xvt_errmsg_push_handler
xvt_vobj_get_attr
xvt_vobj_set_attr
```

The "Diagnostics and Debugging" chapter of the *XVT Portability Toolkit Guide*

ATTR_EVENT_HOOK

Description

A pointer to an event-handling function for native events. The prototype of this function varies between platforms, as do the nature of events sent to it. Refer to your platform-specific book for proper function prototypes and return value meaning.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	The currently installed event hook function.
`xvt_vobj_set_attr` effect:	Sets the event hook function. Setting this to `NULL` is valid, and means that no event hook is installed.
`xvt_app_create` use:	Can use either before or after
Default value:	`NULL`

See Also

```
ATTR_HELP_HOOK
ATTR_KEY_HOOK
EVENT
xvt_vobj_get_attr
xvt_vobj_set_attr
```

The "Events" chapter in the *XVT Portability Toolkit Guide*
The *XVT Platform-Specific Books*

ATTR_FONT_CACHE_SIZE

Description

This attribute controls the size of the font cache, which has a significant impact upon text drawing performance when multiple

physical fonts are used. It takes a single `long` number, which is the desired font cache size.

For each platform, the XVT Portability Toolkit establishes a reasonable font cache size, which is in effect at system startup time. If your application is font-intensive, you might want to increase the cache size to increase performance.

Your application can change the default font cache size by setting this attribute *before* you call `xvt_app_create`. Setting this attribute after calling `xvt_app_create` has no effect on the cache size because it has already been allocated.

If you don't set this attribute, the XVT default cache size is used. Setting it to zero resets the cache size to the default.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Current font cache size
`xvt_vobj_set_attr` effect:	Sets size of font cache
`xvt_app_create` use:	Must use before
Default value:	Varies for each platform

See Also

`xvt_vobj_get_attr`
`xvt_vobj_set_attr`

The "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

ATTR_FONT_DIALOG

Description

This attribute holds the pointer to the application-supplied Font Selection dialog. It controls whether or not an application-supplied Font Selection dialog is used in the following situations:

- When a user invokes the Font Selection dialog from the menu
- When the application calls xvt_dm_post_font_sel

The attribute value is a pointer to the dialog-invoking function. To register an application-customized font dialog with XVT, use `xvt_vobj_set_attr` with this attribute. To retrieve the current application font dialog function pointer, use `xvt_vobj_get_attr`.

If you don't set this attribute, the XVT default Font Selection dialog is used for both the menu-activated Font Selection dialog and `xvt_dm_post_font_sel`. Setting it to `NULL` causes the default Font

Selection dialog to be used. Any application-supplied Font Selection dialog must use the XVT_FONT_DIALOG signature, and return TRUE if the font is selected, or FALSE if one is not selected.

Prototype:

```
typedef BOOLEAN (* XVT_FONT_DIALOG) (WINDOW
    win, XVT_FNTID default_font_id, PRINT_RCD
    *precp,unsigned long reserved)
```

Uses win argument:	No
xvt_vobj_get_attr returns:	Pointer to application-written Font Selection dialog function or NULL
xvt_vobj_set_attr effect:	Sets the Font Selection dialog function pointer
xvt_app_create use:	Can use either before or after
Default value:	NULL

See Also

PRINT_RCD
XVT_FNTID
XVT_FONT_DIALOG
xvt_dm_post_font_sel
xvt_vobj_get_attr
xvt_vobj_set_attr

The "Customized Font Selection Dialogs" section of the "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

ATTR_FONT_MAPPER

Description

This attribute holds the pointer to the application-supplied Logical Font mapping Function. It controls whether or not an application-supplied font mapper is used. To register an application-supplied font mapper with XVT, use xvt_vobj_set_attr with this attribute.

To retrieve the current application font mapper function pointer, use xvt_vobj_get_attr. To remove a registered application-supplied font mapper, use xvt_vobj_set_attr with this attribute, passing NULL as the value. Any application-supplied font mapper must use the XVT_FONT_MAPPER signature, and return TRUE if the function maps the logical font, or FALSE if the function does not map the logical font.

Prototype:

```
typedef BOOLEAN (* XVT_FONT_MAPPER)
(XVT_FNTID font_id)
```

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Pointer to application-supplied font mapper function or `NULL`
`xvt_vobj_set_attr` effect:	Sets the font mapper function pointer
`xvt_app_create` use:	`Can use either before or after`
`Default value:`	`NULL`

See Also

`ATTR_FRAME_WIDTH`
`XVT_FNTID`
`XVT_FONT_MAPPER`
`xvt_win_create`
`xvt_win_create_def`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`

The "Font Mapping Example" section of the "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

ATTR_FRAME_HEIGHT

Description

The thickness in pixels of a horizontal border of a non-resizable window. You can use this value to calculate the outer size of a window, given its client area.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Border thickness in pixels
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

`ATTR_FRAME_WIDTH`
`xvt_vobj_get_attr`
`xvt_vobj_get_client_rect`
`xvt_vobj_get_outer_rect`
`xvt_vobj_set_attr`
`xvt_vobj_translate_points`
`xvt_win_create`
`xvt_win_create_def`

The "Windows" chapter in the *XVT Portability Toolkit Guide*

ATTR_FRAME_WIDTH

Description

The thickness in pixels of a vertical border of a non-resizable window. You can use this value to calculate the outer size of a window, given its client area.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Border thickness in pixels
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

`ATTR_FRAME_HEIGHT`
`xvt_vobj_get_attr`
`xvt_vobj_get_client_rect`
`xvt_vobj_get_outer_rect`
`xvt_vobj_set_attr`
`xvt_vobj_translate_points`
`xvt_win_create`
`xvt_win_create_def`

The "Windows" chapter in the *XVT Portability Toolkit Guide*

ATTR_HAVE_MOUSE

Description

A `BOOLEAN` value indicating if the program is running on a system with a mouse or other pointing device present.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	`TRUE` if the system has a pointing device
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

The *XVT Platform-Specific Books*

ATTR_HELP_CONTEXT

Description

Registers a context that is passed to the help hook function. This attribute is used internally to pass an `XVT_HELP_INFO` handle to the default help hook. You can use this attribute with the `ATTR_HELP_HOOK` attribute to customize the behavior of the help system.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	The currently registered help hook context
`xvt_vobj_set_attr` effect:	Sets the help hook context. Setting this to `NULL` is valid, and means that `NULL` will be passed to the registered help hook.
`xvt_app_create` use:	Can use either before or after
Default value:	`NULL`

See Also

`ATTR_HELP_HOOK`
`XVT_HELP_INFO`
`xvt_help_process_event`
`xvt_help_open_helpfile`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

ATTR_HELP_HOOK

Description

A pointer to an event-handling function. The purpose of this function is to pass XVT events to the help system. All XVT events are passed through it, including those delivered via a call to `xvt_win_dispatch_event`. The function may do special processing of the events before passing them to the help system. Returning `TRUE` means that the XVT Portability Toolkit should continue to process the event. Returning `FALSE` means the Toolkit should not process it further.

The `context` argument to the help hook function contains programmer-specified context data. The Toolkit normally sets this attribute when `xvt_help_open_helpfile` is called. The Toolkit-provided help hook simply calls `xvt_help_process_event` and returns `FALSE`. To customize the event delivery to the help system, you should set this attribute to your own help hook prior to calling `xvt_help_open_helpfile`.

Prototype:

```
BOOLEAN HelpHook (void* context, WINDOW win,
    EVENT* ev);
```

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	The currently installed help hook function.
`xvt_vobj_set_attr` effect:	Sets the help hook function. Setting this to `NULL` is valid, and means that there is no help hook installed.
`xvt_app_create` use:	Can use either before or after
Default value:	`NULL`

See Also

`ATTR_EVENT_HOOK`
`ATTR_HELP_CONTEXT`
`ATTR_KEY_HOOK`
`xvt_help_open_helpfile`
`xvt_help_process_event`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`
`xvt_win_dispatch_event`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

ATTR_ICON_HEIGHT

Description

The default icon height, which can be used to determine how much vertical space is used by `xvt_dwin_draw_icon`. This value's usefulness is limited by the fact that it is possible to create variable-size icons on some platforms.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Icon height
`xvt_vobj_set_attr` effect:	Illegal

xvt_app_create use:	Can use either before or after
Default value:	Varies for each platform

See Also

ATTR_ICON_WIDTH
xvt_dwin_draw_icon
xvt_vobj_get_attr
xvt_vobj_set_attr

ATTR_ICON_WIDTH

Description

The default icon width, which can be used to determine how much horizontal space is used by xvt_dwin_draw_icon. This value's usefulness is limited by the fact that it is possible to create variable-size icons on some platforms.

Uses win argument:	No
xvt_vobj_get_attr returns:	Icon width
cxvt_vobj_set_attr effect:	Illegal
xvt_app_create use:	Can use either before or after
Default value:	Varies for each platform

See Also

ATTR_ICON_HEIGHT
xvt_dwin_draw_icon
xvt_vobj_get_attr
xvt_vobj_set_attr

ATTR_KEY_HOOK

Description

A pointer to an event-handling function for native keystroke events. The prototype of this key hook function varies between platforms, as do the nature of events sent to it. The protoype also depends on whether the application is executing in single-byte mode or in multibyte-aware mode (ATTR_MULTIBYTE_AWARE is TRUE). Refer to your platform-specific book for proper function prototypes and return value meaning. This pointer allows an application to supplement XVT's internal key translation algorithm.

Note: If ATTR_MULTIBYTE_AWARE is set to TRUE, the application key hook function is responsible for properly setting both the virtual_key field and the modifiers field in the EVENT structure.

Uses win argument:	No
xvt_vobj_get_attr returns:	The currently installed key hook function.
xvt_vobj_set_attr effect:	Sets the key hook function. Setting this to NULL is valid and means no key hook is installed.
xvt_app_create use:	Can use either before or after
Default value:	NULL

See Also

ATTR_EVENT_HOOK
ATTR_HELP_HOOK
ATTR_MULTIBYTE_AWARE
E_CHAR
XVT_MOD_KEY
xvt_vobj_get_attr
xvt_vobj_set_attr

The "Events" and the "Multibyte Charater Sets and Localization" chapters in the *XVT Portability Toolkit Guide*
The *XVT Platform-Specific Books*

ATTR_MEMORY_MANAGER

Description

This attribute is the address of a structure of type XVT_MEM (defined in **xvt_type.h**). This structure contains the addresses of the system-wide memory management functions that are called when the application invokes xvt_mem_alloc, xvt_mem_free, xvt_mem_realloc, and xvt_mem_zalloc.

Applications wishing to set or retrieve the addresses of the underlying memory management functions used by the system should use this attribute.

This attribute must be set by the *first* call to the xvt_vobj_set_attr. Setting any other attribute first, or calling xvt_app_create, forces the system to use the default memory management functions, which cannot be replaced.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Current memory management functions in an `XVT_MEM` structure
`xvt_vobj_set_attr` effect:	Sets the memory management functions that will be used for memory allocation, memory freeing, memory reallocation, and allocating and zeroing memory
`xvt_app_create` use:	Must use before
Default value:	`NULL` before a call to `xvt_app_create` (afterwards it is the address of an `XVT_MEM` structure containing addresses of the system default memory management functions)

Note: Remember to use the `XVT_CALLCONV1` macro in the prototypes and headers for your memory management functions.

See Also

`XVT_CALLCONV*`
`xvt_mem_*`

The "Memory Allocation" chapter in the *XVT Portability Toolkit Guide*

Example

The following code sets the memory management functions, which must be done before `xvt_app_create`:

```
    XVT_MEM my_functions = {my_alloc, my_free,
my_realloc,
        my_zmalloc};
     xvt_vobj_set_attr(NULL_WIN, ATTR_MEMORY_MANAGER,
         (long)&my_functions);
```

ATTR_MENU_HEIGHT

Description

The height of a menubar. You can use this value to calculate the outer size of a window, given its client area. However, it is up to the application to determine whether a particular window has a menu attached to it.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Menu height in pixels
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

`xvt_menu_get_tree`
`xvt_menu_set_tree`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`
`xvt_win_create`
`xvt_win_create_def`
`xvt_win_has_menu`

The "Menus" chapter in the *XVT Portability Toolkit Guide*

ATTR_MULTIBYTE_AWARE

Description

This attribute of type `BOOLEAN` must be set to `TRUE` before `xvt_app_create` is called if and only if the application has been internationalized to be multibyte-capable. This allows XVT to determine which version of the application key hook interface to invoke, whether to set the new `E_CHAR` event fields, as well as do some internal performance enhancements for non-multibyte applications.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	`TRUE` if set
`xvt_vobj_set_attr` effect:	Sets multibyte functionality if set to `TRUE`
`xvt_app_create` use:	Must use before
Default value:	`FALSE`

See Also

`ATTR_KEY_HOOK`
`E_CHAR`
`XVT_MOD_KEY`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`

The "Multibyte Character Set and Localization" chapter in the *XVT Portability Toolkit Guide*

ATTR_NATIVE_GRAPHIC_CONTEXT

Description

This value represents the underlying graphical context for a particular window in the native window system. While this attribute is portable, it has a non-portable return value.

Note: On some platforms, the graphic context returned for this attribute might not be persistent. The context should be acquired every time it is needed. For more information, see the *XVT Platform-Specific Books*.

Platform	Return Value
XVT/Win32	`HDC`
XVT/Mac	`Grafport`
XVT/XM	`GC`. However, XVT does not recommend using this `GC` because it has some undocumented side-effects, and because `GC`s are easy to create yourself.

Uses `win` argument:	Yes
`xvt_vobj_get_attr` returns:	Native context (requires casting)
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Must use after
Default value:	Undefined

See Also

`ATTR_NATIVE_WINDOW`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`

The *XVT Platform-Specific Books*

ATTR_NATIVE_WINDOW

Description

This value represents the underlying window object for a particular window in the native window system. While this attribute is portable, it has a non-portable return value.

Platform	Return Value
XVT/Win32	`HWND`

XVT/Mac	`Windowptr`
XVT/XM	`Window`

Uses `win` argument:	Yes
`xvt_vobj_get_attr` returns:	Native graphical window (requires casting)
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Must use after
Default value:	Undefined

See Also

`ATTR_NATIVE_GRAPHIC_CONTEXT`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`

The *XVT Platform-Specific Books*

ATTR_NUM_TIMERS

Description

The number of timers in the system available to the application via `xvt_timer_create`.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Number of available timers
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

`E_TIMER`
`xvt_timer_create`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`

The "`E_TIMER` Events" section of the "Events" chapter in the *XVT Portability Toolkit Guide*

ATTR_PRINTER_HEIGHT

Description

The height of the default printer, in pixels.

Note: The ATTR_PRINTER_* attributes return only those values appropriate for the default printer settings. To retrieve printer metrics for a printer setting stored in a PRINT_RCD, see XVT_ESC_*.

Uses win argument:	No
xvt_vobj_get_attr returns:	Printer height
xvt_vobj_set_attr effect:	Illegal
xvt_app_create use:	Can use either before or after
Default value:	Varies for each platform

See Also

ATTR_PRINTER_HRES
ATTR_PRINTER_VRES
ATTR_PRINTER_WIDTH
XVT_ESC_*
xvt_vobj_get_attr
xvt_vobj_set_attr

The *XVT Platform-Specific Books*

ATTR_PRINTER_HRES

Description

The horizontal resolution of the default printer, in pixels per inch.

Note: The ATTR_PRINTER_* attributes return only those values appropriate for the default printer settings. To retrieve printer metrics for a printer setting stored in a PRINT_RCD, see XVT_ESC_*.

Uses win argument:	No
xvt_vobj_get_attr returns:	Printer horizontal resolution
xvt_vobj_set_attr effect:	Illegal
xvt_app_create use:	Can use either before or after
Default value:	Varies for each platform

See Also

ATTR_PRINTER_HEIGHT
ATTR_PRINTER_VRES
ATTR_PRINTER_WIDTH
XVT_ESC_*
xvt_vobj_get_attr
xvt_vobj_set_attr

ATTR_PRINTER_VRES

Description

The vertical resolution of the default printer, in pixels per inch.

Note: The `ATTR_PRINTER_*` attributes return only those values appropriate for the default printer settings. To retrieve printer metrics for a printer setting stored in a `PRINT_RCD`, see `XVT_ESC_*`.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Printer vertical resolution
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

`ATTR_PRINTER_HEIGHT`
`ATTR_PRINTER_HRES`
`ATTR_PRINTER_WIDTH`
`XVT_ESC_*`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`

ATTR_PRINTER_WIDTH

Description

The width of the default printer, in pixels.

Note: The `ATTR_PRINTER_*` attributes return only those values appropriate for the default printer settings. To retrieve printer metrics for a printer setting stored in a `PRINT_RCD`, see `XVT_ESC_*`.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Printer width
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after
Default value:	Varies for each platform

See Also

ATTR_PRINTER_HEIGHT
ATTR_PRINTER_HRES
ATTR_PRINTER_VRES
XVT_ESC_*
xvt_vobj_get_attr
xvt_vobj_set_attr

ATTR_PROPAGATE_NAV_CHARS

Description

This attribute affects the propagation of navigation character E_CHAR events from controls to their container window's event handler. This attribute has no effect on dialogs. A navigation character is any typed character capable of causing a shift in keyboard focus as defined by platform look-and-feel. The default behavior of navigation character propagation from controls to container window event handlers is platform-specific. It varies with look-and-feel issues and the type of object having focus.

This attribute (set to a value of TRUE) insures that character events relevant to keyboard navigation are generated and sent to your window event handler overriding any default platform-specific behavior. This feature allows you to implement portable routines for keyboard navigation and field validation in windows of your XVT application.

Uses win argument:	Yes
xvt_vobj_get_attr returns:	TRUE or FALSE
xvt_vobj_set_attr effect:	TRUE to propagate navigation characters. FALSE for default platform-specific navigation character event propagation (R4.0 compatible).
xvt_app_create use:	Any time after
Default value:	FALSE

Navigation keys which may be affected by this attribute include the following:

- Tab key
- Back-Tab (Shift-tab) key
- Enter or Return key (except XVT/Mac)
- Escape key (except XVT/Mac)

- Arrow keys (except XVT/Mac)
- Text character keys used for mnemonics (except XVT/Mac and XVT/XM). Unmodified mnemonics are not affected for list or edit controls.

Other keys that also may be affected by this attribute include the following:

- Alt key combinations
- Control key combinations
- Virtual keys
- Function keys (`K_F*` function keys are distinguished separately here from other virtual keys)

Each platform propagates `E_CHAR` events in a different manner depending upon look-and-feel issues and keyboard focus.

Propagation of navigation and other key types (listed above) for value of `ATTR_PROPAGATE_NAV_CHARS`:

Focus in window with no focusable control:	
XVT/Mac	
Attribute`FALSE`:	All keys
Attribute`TRUE`:	All keys
XVT/Win32	
Attribute`FALSE`:	All keys *except* Alt key combinations and virtual keys
Attribute`TRUE`:	All keys
XVT/XM	
Attribute`FALSE`:	All keys *except* Alt key combinations
Attribute`TRUE`:	All keys *except* Alt key combinations

Focus in window with focusable control:

XVT/Mac

Attribute`FALSE`:	*Only* virtual keys and function keys *not* consumed by a control
Attribute`TRUE`:	All keys *except* virtual keys and function keys consumed by a control

XVT/Win32

Attribute`FALSE`:	All keys *except* Alt key combinations
Attribute`TRUE`:	All keys

XVT/XM

Attribute`FALSE`:	All keys *except* Alt key combinations
Attribute`TRUE`:	All keys *except* Alt key combinations

Focus in non-text entry type control:

XVT/Mac (includes list button)

Attribute`FALSE`:	Not applicable - these controls are non-focusable
Attribute`TRUE`:	Not applicable - these controls are non-focusable

XVT/Win32

Attribute`FALSE`:	All keys *except* text mnemonics
Attribute`TRUE`:	All keys

XVT/XM

Attribute`FALSE`:	All keys *except* Alt key combinations
Attribute`TRUE`:	All keys *except* Alt key combinations

Focus in text entry control (includes list button, list box):	
XVT/Mac (excludes list button)	
Attribute`FALSE`:	*Only* virtual keys and function keys *not* consumed by the control
Attribute`TRUE`:	All keys *except* virtual keys and function keys consumed by the control
XVT/Win32	
Attribute`FALSE`:	All keys *except* text mnemonics
Attribute`TRUE`:	All keys
XVT/XM	
Attribute`FALSE`:	All keys *except* Alt key combinations
Attribute`TRUE`:	All keys *except* Alt key combinations

SeeAlso

```
ATTR_KEY_HOOK
E_CHSee AlsoAR
xvt_vobj_get_attr
xvt_vobj_set_attr
```

The "Events" chapter in the *XVT Portability Toolkit Guide*

ATTR_RESOURCE_FILENAME

Description

This attribute can be set to a multibyte string containing a pathname to an external resource file. On the XVT/Win32 platform, the file is a DLL containing resources. On XVT/Mac and XVT/XM, the file is a resource file that is external to the application. If the attribute is `NULL`, XVT uses the resources bound to the application.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Name of resource file
`xvt_vobj_set_attr` effect:	Sets application resource DLL or resource file
`xvt_app_create` use:	Must use before
Default value:	`NULL`

See Also

```
xvt_vobj_get_attr
xvt_vobj_set_attr
```

The "Resources and URL" and the "Multibyte Character Sets and Localization" chapters in the *XVT Portability Toolkit Guide*
The *XVT Platform-Specific Books*

ATTR_R40_TXEDIT_BEHAVIOR

Description

By default, XVT encapsulates its text edit object (multiline edit control) in a child window. This allows text edit objects to participate fully and properly in keyboard focus and navigation as with any other type of control or child window.

In Release 4.0x, text edit objects were merely drawn in their container window with XVT providing functions for proper interaction with other GUI objects. The `ATTR_R40_TXEDIT_BEHAVIOR` attribute allows you to continue using this behavior. However, XVT strongly recommends that you use the default behavior whenever possible.

The Release 4.0x functions `xvt_tx_is_scroll_update` and `xvt_tx_process_event` are supported only when `ATTR_R40_TXEDIT_BEHAVIOR` is set to `TRUE`. Calling these functions when the attribute is set to `FALSE` result in no operation.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	`Previously set value`
`xvt_vobj_set_attr` effect:	If `TRUE`, use text edit objects as in Release 4.0x. If `FALSE`, use text edit objects encapsulated in a child window.
`xvt_app_create` use:	Must use before
Default value:	FALSE

See Also

The "Text Edit Object" section of the "Controls" chapter in the *XVT Portability Toolkit Guide*

ATTR_SCREEN_HEIGHT

Description

The height of the screen, in pixels.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Screen height
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after, except on XVT/XM, only use after
Default value:	Varies for each platform

See Also

`ATTR_SCREEN_HRES`
`ATTR_SCREEN_VRES`
`ATTR_SCREEN_WIDTH`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`
`xvt_win_create`
`xvt_win_create_def`
`xvt_win_create_res`

The "Coordinate Systems" chapter in the *XVT Portability Toolkit Guide*

ATTR_SCREEN_HRES

Description

The horizontal resolution of the screen, in pixels per inch.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Screen horizontal resolution
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after, except on XVT/XM, only use after
Default value:	Varies for each platform

See Also

`ATTR_SCREEN_HEIGHT`
`ATTR_SCREEN_VRES`
`ATTR_SCREEN_WIDTH`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`
`xvt_win_create`
`xvt_win_create_def`
`xvt_win_create_res`

The "Coordinate Systems" chapter in the *XVT Portability Toolkit Guide*

ATTR_SCREEN_VRES

Description

The vertical resolution of the screen, in pixels per inch.

Uses `win` argument:	No
xvt_vobj_get_attr returns:	Screen vertical resolution
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after, except on XVT/XM, only use after
Default value:	Varies for each platform

See Also

```
ATTR_SCREEN_HEIGHT
ATTR_SCREEN_HRES
ATTR_SCREEN_WIDTH
xvt_vobj_get_attr
xvt_vobj_set_attr
xvt_win_create
xvt_win_create_def
xvt_win_create_res
```

The "Coordinate Systems" chapter in the *XVT Portability Toolkit Guide*

ATTR_SCREEN_WIDTH

Description

The width of the screen, in pixels.

Uses `win` argument:	No
`xvt_vobj_get_attr` returns:	Screen width
`xvt_vobj_set_attr` effect:	Illegal
`xvt_app_create` use:	Can use either before or after, except on XVT/XM, only use after
Default value:	Varies for each platform

See Also

```
ATTR_SCREEN_HEIGHT
ATTR_SCREEN_HRES
ATTR_SCREEN_VRES
xvt_vobj_get_attr
xvt_vobj_set_attr
xvt_win_create
xvt_win_create_def
xvt_win_create_res
```

The "Coordinate Systems" chapter in the *XVT Portability Toolkit Guide*

ATTR_SCREEN_WINDOW

Description

The WINDOW value for the screen window. The screen window corresponds to the monitor screen and might serve as a container (or parent) for top-level windows and dialogs.

Uses win argument:	No
xvt_vobj_get_attr returns:	The WINDOW for the screen window
xvt_vobj_set_attr effect:	Illegal
xvt_app_create use:	Can use either before or after, except on XVT/XM, only use after
Default value:	Varies for each platform

See Also

ATTR_TASK_WINDOW
SCREEN_WIN
TASK_WIN
xvt_vobj_get_attr
xvt_vobj_set_attr

The "Screen and Task Windows" section of the "Windows" chapter in the *XVT Portability Toolkit Guide*

ATTR_SUPPRESS_UPDATE_CHECK

Description

A BOOLEAN value that controls XVT's policing of invalid function calls during E_UPDATE events. Normally, XVT does not allow many function calls during an E_UPDATE, because they confuse the native window systems and are poor programming practice. However, if your application runs into an obscure case requiring this check to be disabled, then you can set this attribute to TRUE.

Uses win argument:	No
xvt_vobj_get_attr returns:	TRUE if update checking is disabled
xvt_vobj_set_attr effect:	Disables update checking if TRUE; enables update checking if FALSE
xvt_app_create use:	Can use either before or after, except on XVT/XM, only use after
Default value:	FALSE

See Also

E_UPDATE
xvt_vobj_get_attr
xvt_vobj_set_attr

The "E_UPDATE Events" section of the "Events" chapter in the *XVT Portability Toolkit Guide*

ATTR_TASK_WINDOW

Description

The WINDOW value for the task window. The task window can serve as a container (or parent) for all windows on all platforms. On XVT/XVT/Win32, the task window corresponds to a physical container window. On XVT/XM, the task window is represented by an independent top-level window containing a menubar only. On other platforms, the task window corresponds to the screen.

Uses win argument:	No
xvt_vobj_get_attr returns:	The WINDOW for the task window
xvt_vobj_set_attr effect:	Illegal
xvt_app_create use:	Can use either before or after
Default value:	NULL_WIN

See Also

ATTR_SCREEN_WINDOW
SCREEN_WIN
TASK_WIN
xvt_vobj_get_attr
xvt_vobj_set_attr

The "Screen and Task Windows" section of the "Windows" in *XVT Portability Toolkit Guide*

ATTR_TASKWIN_TITLE_RID

Description

This attribute can be set to the resource ID of the multibyte string containing the value of taskwin_title for use in the XVT_CONFIG

structure. In xvt_app_create, this attribute is tested for non-zero by XVT and the resource loaded into XVT_CONFIG.

Uses win argument:	No
xvt_vobj_get_attr returns:	Gets title RID
xvt_vobj_set_attr effect:	Sets the title resource ID for taskwin
xvt_app_create use:	Must use before
Default value:	NULL

See Also

ATTR_APPL_NAME_RID
ATTR_XVT_CONFIG
XVT_CONFIG
xvt_app_create
xvt_vobj_get_attr
xvt_vobj_set_attr

ATTR_TITLE_HEIGHT

Description

The height of a window's title area. You can use this value to calculate the outer size of a window, given its client area. However, it is up to the application to determine whether a particular window has a title attached to it.

Uses win argument:	No
xvt_vobj_get_attr returns:	Menu height in pixels
xvt_vobj_set_attr effect:	Illegal
xvt_app_create use:	Can use either before or after
Default value:	Varies for each platform

See Also

xvt_vobj_get_attr
xvt_vobj_get_client_rect
xvt_vobj_get_outer_rect
xvt_vobj_set_attr
xvt_vobj_translate_points
xvt_win_create
xvt_win_create_def

The "Windows" chapter in the *XVT Portability Toolkit Guide*

ATTR_XVT_CONFIG

Description

The XVT_CONFIG pointer passed to xvt_app_create.

Uses win argument:	No
xvt_vobj_get_attr returns:	Pointer to XVT_CONFIG
xvt_vobj_set_attr effect:	Illegal
xvt_app_create use:	Can use either before or after
Default value:	NULL

See Also

XVT_CONFIG
xvt_app_create
xvt_vobj_get_attr
xvt_vobj_set_attr

XVT Events

```
EVENT_TYPE
E_CHAR
E_CLOSE
E_COMMAND
E_CONTROL
E_CREATE
E_CXO
E_DESTROY
E_FOCUS
E_FONT
E_HELP
E_HSCROLL
E_MOUSE_DBL
E_MOUSE_DOWN
E_MOUSE_MOVE
E_MOUSE_SCROLL
E_MOUSE_UP
E_QUIT
E_SIZE
E_TIMER
E_UPDATE
E_USER
E_VSCROLL
```

EVENT_TYPE

Event-Type

Summary

```
typedef enum _event_type {
    E_CREATE,       /* creation */
    E_DESTROY,      /* destruction */
    E_FOCUS,        /* window focus gain/loss */
    E_SIZE,         /* resize */
    E_UPDATE,       /* update */
    E_CLOSE,        /* close window request */
    E_MOUSE_DOWN,   /* mouse down */
    E_MOUSE_UP,     /* mouse up */
    E_MOUSE_MOVE,   /* mouse move */
    E_MOUSE_DBL,    /* mouse double click */
    E_CHAR,         /* character typed */
    E_VSCROLL,      /* vert. window scrollbar activity */
    E_HSCROLL,      /* horz. window scrollbar activity */
    E_COMMAND,      /* menu command */
    E_FONT,         /* font menu selection */
    E_CONTROL,      /* control activity */
    E_TIMER,        /* timer */
    E_QUIT,         /* application shutdown request */
    E_HELP,         /* help invoked */
    E_USER,         /* user defined */
} EVENT_TYPE;
```

Description

Values of this data type identify one of the XVT events.

Implementation Note

You can’t depend on the actual value of the enumeration constants, or on the number of event types, because these are likely to be changed in future versions of XVT.

See Also

XVT Events

`EVENT`

`EVENT_HANDLER`

`EVENT_MASK`

`WINDOW`

`XVT_CALLCONV*`

The "Events" chapter in the *XVT Portability Toolkit Guide*

Example

This code fragment shows a typical use of EVENT_TYPEs in XVT:

```
long XVT_CALLCONV1 win_eh(WINDOW xdWindow,
    EVENT *xdEvent)
{
    switch(xdEvent->type){
    case E_CREATE:
        /* creation handling code */
        break;
    case E_DESTROY:
        /* destruction handling code */
        break;
        ...
    }
}
```

E_CHAR

Keyboard-Character Event

Summary

```
typedef struct {            /* event description */
    EVENT_TYPE type;        /* E_CHAR */
    union {
    ...
    struct s_char {
    XVT_WCHAR ch;           /* wide character */
    BOOLEAN shift;          /* shift-key? */
    BOOLEAN control;        /* ctrl or option key? */
    BOOLEAN virtual_key;    /* virtual key? */
    unsigned long modifiers; /* key bit field modifiers */
    } chr;
    ...
    } v;

} EVENT;
```

Description

XVT sends an E_CHAR event to the event handler for a WINDOW when the user types a character or virtual key code into a window. The E_CHAR event is delivered only to the event handler of the window that has keyboard focus, and for a control that has not absorbed the character event for its own use. When the WINDOW event handler receives an event, the WINDOW argument specifies the window in which the event occurred, and the EVENT pointer defines an EVENT

structure with fields in the chr union expressing event-specific information.

If the key is held down and auto-repeat occurs, a separate event is generated for each repetition, so repeated characters don't require special handling.

Processing Characters

The EVENT substructure chr contains the character code field (ch) which is an XVT_WCHAR. Multibyte-capable applications must use the XVT function xvt_str_convert_wc_to_mb before assigning an XVT_WCHAR wide character to a multibyte string array or processing the character with other XVT functions. It is recommended, but not required, that single-byte applications also use this function. Single-byte applications can always cast XVT_WCHAR characters to char as long as they do not rely on the virtual key portion (high byte) of the XVT_WCHAR (this will not port to multibyte applications).

Modifier Keys

The BOOLEAN members of the chr substructure, shift and control, indicate whether those keys were held down while a character was typed. However, if the user types an uppercase character or a control character (such as '<fct' or '<fcb'), the true character code value is in ch, so it's not necessary to look at shift or control to see what was actually typed. In fact, your application should use the shift and control members sparingly, because doing so may make it less portable. (Some platforms cannot provide accurate information to be placed in the shift and control fields.)

In addition to the shift and control fields, the modifiers field is a general way for detecting a pressed modifier key (Control key, Option key, Alt key, etc.). This field holds an OR'd combination of bit-wise flags to indicate one or more modifier keys selected. All available modifier keys are passed in the E_CHAR event for use by the application. The following constants are defined for bit positions in the modifiers field and indicate which corresponding key or keys are held down:

XVT_MOD_KEY_NONE

No modifier keys are pressed.

XVT_MOD_KEY_SHIFT

Shift key is pressed (either XVT_MOD_KEY_LSHIFT bit or XVT_MOD_KEY_RSHIFT bit also set on platforms that can detect individual Left or Right Shift keys).

XVT_MOD_KEY_CTL

Control key is pressed.

XVT_MOD_KEY_ALT

Alt key is pressed.

XVT_MOD_KEY_LSHIFT

Left Shift key is pressed on platforms that can detect Left Shift key (XVT_MOD_KEY_SHIFT bit also set).

XVT_MOD_KEY_RSHIFT

Right Shift key is pressed on platforms that can detect Right Shift key (XVT_MOD_KEY_SHIFT bit also set).

XVT_MOD_KEY_CMD

Command key is pressed (available on XVT/Mac only).

XVT_MOD_KEY_OPTION

Option key is pressed (available on XVT/Mac only).

XVT_MOD_KEY_COMPOSE

Compose key is pressed (available on XVT/XM only).

Virtual Keys

XVT virtual key values are the K_* values (F1, Home key, etc.) defined in the **xvt_defs.h** header file. Virtual keys in character events may be detected by several means.

For the ASCII character code set only, values of the ch field greater than UCHAR_MAX indicate a virtual key (except for K_DEL which is less than UCHAR_MAX).

The virtual_key member of the chr substructure is also set to TRUE to distinguish virtual key characters. In multibyte applications, virtual key codes may conflict with some multibyte character encodings. Therefore, the virtual_key field must be validated for multibyte applications.

Alternately, the most general means for testing for a virtual key (regardless of character code set) is to pass the EVENT structure to the xvt_event_is_virtual_key utility function which determines if the character in a E_CHAR event is a virtual key.

Key Hook Attribute

You can change the mapping of raw key codes (as generated by the keyboard) to XVT virtual key codes, or add new codes, by changing the default key hook function. This is done with the function `xvt_vobj_set_attr` and the attribute `ATTR_KEY_HOOK`.

The parameters passed to a key hook function vary depending upon whether your XVT application is capable of processing multibyte characters (`ATTR_MULTIBYTE_AWARE` is set to `TRUE`). Parameters also vary between platforms. In single-byte mode, hook functions receive only platform-specific data. In multibyte-aware mode, though, key hook functions on all platforms receive a pointer to the `EVENT` structure (`E_CHAR` event) in addition to platform-specific information. This is necessary because only the hook function knows if it is mapping a passed character to a virtual key in a multibyte-aware environment. Note that the interface for multibyte hook functions is called only if `ATTR_MULTIBYTE_AWARE` is set to `TRUE`, otherwise the single-byte (default) interface is used.

Implementation Note

In XVT/Win32, the task window's event handler may receive `E_CHAR` events only if the task window is drawable (platform-specific attribute `ATTR_WIN_PM_DRAWABLE_TWIN` is set to `TRUE`).

On XVT/Mac, `E_CHAR` events also are delivered to XVT dialogs. However, to maintain portability, you should not process character events sent to dialogs.

Do not use the Control key for keyboard shortcuts (mnemonics), because the native platforms for XVT/Win32 use the Control key with menu accelerators. Also, on XVT/Mac, the Option key generates non-ASCII characters. XVT/Mac stores these into the `ch` member, and they can be handled normally but their use may not be portable.

See Also

`ATTR_KEY_HOOK`

`ATTR_MULTIBYTE_AWARE`

`ATTR_PROPAGATE_NAV_CHARS`

`E_FOCUS`

`EVENT`

`EVENT_TYPE`

`K_* Key Codes`

`UCHAR_MAX`

`XVT_WCHAR`

`WINDOW`

`XVT_CALLCONV*`

`XVT_MAX_MB_SIZE`

`XVT_MOD_KEY`

`xvt_event_is_virtual_key`

`xvt_str_convert_wc_to_mb`

The "Events" chapter in the *XVT Portability Toolkit Guide*

The *XVT Platform-Specific Books*

Example

The following code processes a single character delivered in an `E_CHAR` event:

```
long XVT_CALLCONV1 win_eh(WINDOW win, EVENT *ep)
{
    static int x = LEFT_MGN, y = 0;
    char mbc[XVT_MAX_MB_SIZE + 1];
    int len, width;
    ...
    switch (ep->type){
        ...
        case E_CHAR:
          if (y == 0) {
            y = doc.height;
            xvt_dwin_set_caret_pos(win, x, y);
            xvt_dwin_set_caret_visible(win, TRUE);
          }
          if (xvt_event_is_virtual_key(ep)) {
            /* donÂt process virtual characters */
            return;len =
              xvt_str_convert_wc_to_mb(mbc,
            ep->v.chr.ch);
          width = xvt_dwin_get_text_width(win, mbc,
            1);
          if ((len == 0) || ((len == 1) &&
              !xvt_str_is_alnum(mbc)))
            /* only process printable characters */
            return;
          if (x + width > doc.rct.right) {
            if (++doc.curline >= doc.maxlines) {
              xvt_dm_post_note(
                "Characters donÂt fit!");
              return;
            }
            x = LEFT_MGN;
            y += doc.height;
          }
          xvt_dwin_draw_text(win, x, y, mbc, 1);
          x += width;
          xvt_dwin_set_caret_pos(win, x, y);
          save_char(ep->v.chr.ch);
          ...
          break;
        ...
    }
}
```

E_CLOSE

Close-Window Event

Summary

```
typedef struct {       /* event description */
    EVENT_TYPE type;   /* event type (E_CLOSE) */
    ...

} EVENT;
```

Description

XVT sends an E_CLOSE event to the event handler for a window or dialog in response to the user clicking its "close box." Windows that aren't created with either the WSF_CLOSE or WSF_DECORATED flags won't have a "close box." The WINDOW argument in the application event handler indicates which dialog or window the user has tried to close. No additional information is needed to process this event, so the EVENT structure contains none.

When this event is received, the window or dialog hasn't actually been closed, your application must call xvt_vobj_destroy to destroy the window. If your application calls xvt_vobj_destroy, then an E_DESTROY event is sent to the window's event handler. The purpose of the E_DESTROY event is to notify your application that it needs to free the memory allocated for application data associated with the WINDOW being destroyed. The E_DESTROY is the last event the WINDOW will receive. Keep in mind that if your application does *not* call xvt_vobj_destroy, XVT will *not* close the window for it.

After calling xvt_vobj_destroy for the window, you should not attempt to call any XVT functions with the window as an argument.

If E_CLOSE is ignored, then no window is closed, and nothing in the application changes. This distinction is important. Typically, applications check the state of the window upon receiving an E_CLOSE event. If the state indicates that the contents of the window or dialog have been saved (for example), then the application can simply call xvt_vobj_destroy. If, however, the contents have not been saved, the application can display a dialog asking if the user wishes to save or discard changes, so that the changes can be preserved before calling xvt_vobj_destroy.

E_CLOSE events are generated for the task window, regular windows, and dialogs. They are not generated for print windows, pixmaps, controls, or screen windows.

On all platforms, if you call xvt_vobj_destroy with the task window, an E_DESTROY event will be sent to the task event handler, but it is not guaranteed that the children of the task window will receive E_DESTROY events.

Note: An E_CLOSE event is not sent when the user chooses "Close" from the file menu. Instead, an E_COMMAND event is sent with the v.cmd.tag field of the event structure set to M_FILE_CLOSE. It is common for applications to perform similar window closing activities both upon the receipt of an E_COMMAND event with a tag of M_FILE_CLOSE and upon the receipt of an E_CLOSE event.

Implementation Note

On the platforms that have a physical representation of a task window (XVT/Win16, XVT/Win32, XVT/PM, and XVT/XM), an E_CLOSE event is sent to the task event handler if the user clicked on the "close box" for the task window to shut down the application.

See Also

E_COMMAND

E_DESTROY

M_EDIT_*, M_FILE_*, M_HELP_* Menu Tags

WINDOW

WSF_* Options Flags

WSF_* Options Flags

xvt_vobj_destroy

The "Events" chapter in the *XVT Portability Toolkit Guide*

Example

In the following code, the application provides the function OK_to_close elsewhere:

```
long XVT_CALLCONV1 a_window_eh(WINDOW win, EVENT *ep)
{
    switch (ep->type) {
    case E_COMMAND:
        switch (ep->v.cmd.tag) {
        case M_FILE_CLOSE:
            if (OK_to_close(win))
                xvt_vobj_destroy(win);
            break;
        }
        break;
    case E_CLOSE:
        if (OK_to_close(win))
            xvt_vobj_destroy(win);
        break;
    }
}
```

E_COMMAND

Menu-Command Event

Summary

```
typedef struct {       /* event description */
    EVENT_TYPE type;   /* event type */
    union {
    ...
    struct s_cmd {
    MENU_TAG tag;      /* menu item tag */
    BOOLEAN shift;     /* shift key down? */
    BOOLEAN control;   /* control/option key? */
    } cmd;
    ...
    } v;

} EVENT;
```

Description

XVT generates an `E_COMMAND` event when the user makes a menu selection (or causes a menu selection by typing a menu-accelerator key). However, selections from Font/Style menus generate `E_FONT` events, not `E_COMMAND` events.

XVT sets the `v.cmd.tag` field of the event structure to the tag of the item that was chosen (as specified in `MENU_ITEM` or your URL file). The `v.cmd.shift` and `v.cmd.control` fields are set to `TRUE` if the user pressed Shift or Control while selecting the menu item.

Unless a window has a menubar, its event handler will *not* receive E_COMMAND events. Therefore, E_COMMAND events will never be sent to child windows or windows created with the WSF_NO_MENUBAR attribute set.

It is possible for a window that does not have the focus to receive an E_COMMAND event. It might be that the object with focus is not in the ancestor chain for the window that received the E_COMMAND event. In that case, calling xvt_scr_get_focus_vobj might result in a misdirected "dispatch" of the event.

The task event handler gets E_COMMAND events under various conditions. When no other menubar-possessing top-level windows are visible, XVT *guarantees* that the user can operate a menubar and will send E_COMMAND events to the task event handler. In addition, an E_COMMAND event can be sent to the task handler in other circumstances:

- On XVT/Mac, if a dialog has the focus, then the task menubar is displayed
- On XVT/Win32, if there aren't any visible document windows whose parent is TASK_WIN, then the task menubar is available

Implementation Note

On the XVT/Mac, the control member indicates whether the *Option* key is held down, not the Control key, which some Macs don't have.

See Also

EVENT_TYPE

MENU_ITEM

MENU_TAG

TASK_WIN

WSF_* Options Flags

xvt_menu_get_tree

xvt_menu_set_tree

The "Events" chapter in the *XVT Portability Toolkit Guide*

Example

The following code handles command events in a window event handler. The macros M_FILE_CLOSE and M_FILE_QUIT are defined in **xvtmenu.h**, which is included by **xvt.h**.

```
#include <xvt.h>static void
do_menu(WINDOW win, MENU_TAG cmd, BOOLEAN shift,
    BOOLEAN control)
{
    switch (cmd) {
    case M_FILE_CLOSE:
        do_close(win);
        break;
    case M_FILE_QUIT:
        xvt_app_destroy();
        break;
    }
}
long XVT_CALLCONV1 a_window_eh(WINDOW win, EVENT *ep)
{
    switch (ep->type) {
    case E_COMMAND:
        do_menu(win, ep->v.cmd.tag, ep->v.cmd.shift,
            ep->v.cmd.control);
        break;
    }
}
```

E_CONTROL

Control Activation Event

Summary

```
typedef struct {       /* event description */
EVENT_TYPE type;       /* type (E_CONTROL) */
    union {
    ...
    struct s_ctl {
    short id;          /* ID of control */
    CONTROL_INFO ci;   /* control info */
    ctl;
    ...
    } v;

} EVENT;
```

Description

XVT sends an `E_CONTROL` event to the event handler for a window or dialog in response to the user operating a control in that window or dialog. XVT also generates an `E_CONTROL` event when the underlying windows system indirectly modifies the state of a control.

The v.ctl.id field of the event contains the ID of the control that was operated. For details on specifying control IDs at creation time, see xvt_ctl_create, xvt_win_create_def, and xvt_dlg_create_def. The WINDOW argument passed to the event handler refers to the window or dialog containing the control, not the WINDOW that represents the control itself. The CONTROL_INFO sub-structure of the event contains information about the control. In particular, it contains the WINDOW for the control that was operated, and the WIN_TYPE of the control.

Implementation Note

This event is not sent to the event handler for the task window unless the application is running on XVT/Win32, and the attribute ATTR_WIN_PM_DRAWABLE_TWIN has been set.

See Also

CONTROL_INFO

EVENT_TYPE

WINDOW

WIN_TYPE

xvt_ctl_create

xvt_ctl_create_def

xvt_dlg_create_def

xvt_dlg_create_res

xvt_win_create_def

xvt_win_create_res

xvt_win_get_ctl

The "Controls" Chapter in the *XVT Portability Toolkit Guide*

E_CREATE

Window Creation Event

Summary

```
typedef struct {
    EVENT_TYPE type;   /* type (E_CREATE) */
    ...

}EVENT;
```

Description

XVT sends an E_CREATE event to the event handler for a WINDOW immediately after the window or dialog has been created. This event is guaranteed to be the first event received by the event handler. E_CREATE is also guaranteed to be the first event sent to the task event handler. The task event handler receives the E_CREATE event after the application calls xvt_app_create.

The WINDOW argument tells the event handler which dialog or window has been created. No additional information is needed to process this event, so the EVENT structure contains none.

Implementation Note

On some platforms, performing certain operations during a window's E_CREATE event (such as creating a dialog) might cause an E_SIZE event to be delivered to the window before the completion of the E_CREATE callback.

See Also

EVENT_TYPE

WINDOW

xvt_app_create

xvt_dlg_create_def

xvt_dlg_create_res

xvt_win_create_def

xvt_win_create_res

The "Event Ordering Rules" section of the "Events" chapter in the *XVT Portability Toolkit Guide*

E_CXO

Container Extension Object Event

Summary

```
typedef struct
{
     EVENT_TYPE type;      /*E_CXO*/
    union
    {
    ...
         struct s_cxo
        {
            long msg_id;   /* CXO message */
            void * ptr;    /* message data */
         }cxo;
     ...
    }v:
 }EVENT;
```

Description

XVT sends an `E_CXO` event to the event handler for a CXO when a CXO is created using `xvt_cxo_create` or destroyed using `xvt_cxo_destroy`. Additionally, an application program can send an `E_CXO` event to a CXO's event handler by calling `xvt_cxo_dispatch_msg`. The `E_CXO` message contains a CXO-specific message and data associated with that message. XVT defines two message types: `XVT_CXO_CREATE_MSG` and `XVT_CXO_DESTROY_MSG`, which are sent to a CXO's event handler when it is created or destroyed.

Implementation Note

The function `xvt_win_dispatch_event` cannot be used to send an `E_CXO` event. Additionally, `E_CXO` messages cannot be masked, and XVT will never dispatch one to a window's event handler {??if it's masked??].

See Also

`EVENT_TYPE`

`XVT_CXO`

`XVT_CXO_*_MSG`

`xvt_cxo_create`

`xvt_cxo_destroy`

E_DESTROY

Window Destruction Event

Summary

```
typedef struct {
     EVENT_TYPE type   /*type (E_DESTROY) */
    ...
}EVENT;
```

Description

XVT sends an E_DESTROY event to the event handler of a window or dialog to notify your application that the WINDOW is about to be destroyed. Typically, an event handler receives an E_DESTROY event soon after your application has called xvt_vobj_destroy. The purpose of the event is to give your application a chance to free memory it has allocated for application data associated with the WINDOW being destroyed.

An E_DESTROY event is guaranteed to be the last event an event handler receives. An E_DESTROY event sent to the task event handler is guaranteed to be the last event received by the application.

The WINDOW argument tells the event handler which window or dialog has been destroyed. No additional information is needed to process this event, so the EVENT structure contains none.

If your application is closing a top-level window, then all of the event handlers for its child windows will receive E_DESTROY events *before* the event handler for the top-level window receives its E_DESTROY. If your application deletes a window at the top of a hierarchy of nested windows, the E_DESTROY events for each window in the hierarchy will be delivered in bottom-up order. This guarantees that when a window receives its E_DESTROY, all lower-level descendent windows will have already received E_DESTROY events.

The only exception to this rule is the task window. If you call xvt_vobj_destroy(TASK_WIN) or xvt_app_destroy, you cannot be sure that the event handlers for the other WINDOWs in the application will get E_DESTROY events before the task event handler does.

Note: At the time that the window or dialog receives an E_DESTROY, you cannot call any functions that operate on the window or dialog *except* xvt_vobj_get_data.

An E_DESTROY might be received before a call to xvt_vobj_destroy returns. Therefore, after calling xvt_vobj_destroy, you should not attempt to call any XVT functions with the window as an argument *except* for xvt_vobj_get_data.

See Also

E_CLOSE

E_CREATE

EVENT_TYPE

E_SIZE

TASK_WIN

WINDOW

xvt_app_create

xvt_vobj_destroy

xvt_vobj_get_data

The "Event Ordering Rules" section of the "Events" chapter in the *XVT Portability Toolkit Guide*

E_FOCUS

Window Focus Gain or Loss Event

Summary

```
typedef struct {       /* event description */
EVENT_TYPE type;       /* type (E_FOCUS) */
    union {
    ...
    BOOLEAN active;    /* focus gain or loss */
    ...
    } v;

} EVENT;
```

Description

An E_FOCUS event is generated when a window or dialog gains or loses the focus. A WINDOW gains or loses the focus as a result of the following actions:

- The user clicks on a focusable area of a window
- Any active keyboard navigation occurs
- Your application calls `xvt_scr_set_focus_vobj`

The `v.active` field of the event structure is set to `TRUE` if the window is gaining the focus, whereas the `v.active` field is set to `FALSE` if the window is losing the focus. If a window does *not* have the focus, it cannot receive an `E_FOCUS` event with `v.active` set to `FALSE`; and if a window already has the focus, it cannot receive an `E_FOCUS` event with `v.active` set to `TRUE`. After a window has gained the focus, then it will either lose it at some point, or it will be destroyed. Also, the window or dialog is guaranteed to get `E_CHAR` events only in between a pair of `E_FOCUS` events indicating gain and loss.

Note: When `WC_EDIT` and `WC_LISTEDIT` controls gain and lose the focus, they communicate that information through the `CONTROL_INFO` structure accompanying the `E_CONTROL` event, instead of pairs of `E_FOCUS` events.

Implementation Note

This event is not sent to the event handler for the task window unless the application is running on XVT/Win32, and the attribute `ATTR_WIN_PM_DRAWABLE_TWIN` has been set. Calling `xvt_scr_set_focus_vobj` during an `E_FOCUS` event results in undefined behavior on some platforms.

See Also

`CONTROL_INFO`

`E_CHAR`

`EVENT_TYPE`

`WINDOW`

`W_*, WC_*, WD_*, Values for WIN_TYPE`

`W_*, WC_*, WD_*, Values for WIN_TYPE`

`xvt_scr_get_focus_vobj`

The "Events" chapter in the *XVT Portability Toolkit Guide*

Example

The following code fragments illustrate how to enable and disable items on the Edit menu when a window gains the focus:

```
static CB_FORMAT paste_fmt;static void
update_menus(WINDOW win)
{
        BOOLEAN paste_enable = TRUE;if
       (xvt_cb_has_format(CB_APPL, APPL_FORMAT))
            paste_fmt = CB_APPL;
        else if (xvt_cb_has_format(CB_PICT, NULL))
            paste_fmt = CB_PICT;
        else if (xvt_cb_has_format(CB_TEXT, NULL))
            paste_fmt = CB_TEXT;
        else
            paste_enable = FALSE;
       xvt_menu_set_item_enabled(win, M_EDIT_PASTE,
            paste_enable);
       xvt_menu_set_item_enabled(win,
           M_EDIT_CLIPBOARD, paste_enable);
 }
```

E_FONT

Font/StyleMenu or Font-Selection-Dialog Event

Summary

```
typedef struct {       /* event description */
    EVENT_TYPE type;   /* event type (E_FONT) */
    union {
    ...
    struct s_efont {
    XVT_FNTID font_id; /* Font id representing selected
                          font */
    } font;
    ...
    } v;

} EVENT;
```

Description

A window's event handler can receive an `E_FONT` event in these two situations:

- If the user selects an item from the standard XVT Font/Style menu *or* invokes the Font Selection dialog from the menu. The receiving window is always a top-level window. If the window does *not* have a menubar, its event handler never receives `E_FONT` events in this manner. Therefore, user interaction with the Font/Style menu or with the Font Selection dialog never results in `E_FONT` events being sent to child windows or windows created with the `WSF_NO_MENUBAR`

attribute set. XVT sets the `v.font.font_id` field of the event structure to the user's chosen logical font.

- If the application calls `xvt_dm_post_font_sel` programmatically (instead of the user invoking it from the menu) *and* the user selects a font. The window receiving the event is the one passed as a parameter to this function. XVT returns in the `v.font.font_id` field the same logical font that was passed in as a parameter to this function.
- It is possible for a window that does not have the focus to receive an `E_FONT` event. It might be that the object with focus is not in the ancestor chain for the window that received the `E_COMMAND` event. In that case, calling `xvt_scr_get_focus_vobj` might result in a misdirected "dispatch" of the event.
- The task event handler gets `E_FONT` events under various conditions. When no other menubar-possessing top-level windows are visible, XVT *guarantees* that the user can operate a menubar and will send `E_FONT` events to the task event handler (if the task window's menubar has a Font/Style menu). In addition, an `E_FONT` event can be sent to the task handler in these circumstances:
- On XVT/Mac, if a dialog has the focus, then the task menubar is displayed
- On XVT/Win32, if there aren't any visible document windows whose parent is `TASK_WIN`, then the task menubar is available

See Also

`EVENT_TYPE`

`TASK_WIN`

`WSF_* Options Flags`

`XVT_FNTID`

`xvt_dm_post_font_sel`

`xvt_dwin_draw_text`

`xvt_menu_get_font_sel`

`xvt_menu_set_font_sel`

The "Events" chapter in the*XVT Portability Toolkit Guide*

Example

The following code displays four text objects, represented by this array of structures:

```
#define NUM_OBJS 4static struct {
    /* information about each object */
    char *text;        /* text */
    PNT pos;           /* starting position */
    RCT bounds;        /* bounding rectangle */
    XVT_FNTID font_id; /* font */
} obj[NUM_OBJS] = {
    {
        "This is the first sentence.",
        {50, 10}
    },
    {
        "This is the second sentence.",
        {125, 150}
     },
    {
        "This is the third sentence.",
        {200, 100}
    },
    {
        "This is the fourth sentence.",
        {275, 200}
    }
};
```

During application initialization, the `font_id` member of each object is set and the bounding rectangle is calculated:

```
static void startup(void)
    {
        int i;
        WINDOW win;win = xvt_win_create(W_DOC,
        XVT_MAX_WINDOW_RECT,
            "FONT", WIN_MENUBAR, TASK_WIN,
            WSF_SIZE|WSF_CLOSE, EM_ALL, win_eh, 0L);
        for (i = 0; i < NUM_OBJS; i++) {
            obj[i].font_id = xvt_font_create();
            xvt_font_set_family(obj[i].font_id,
            XVT_FFN_HELVETICA);
            xvt_font_set_style(obj[i].font_id,
            XVT_FS_NONE);
            switch (i % 3) {
            case 0:
                xvt_font_set_size(obj[i].font_id,
                20);
                break;
            case 1:
                xvt_font_set_size(obj[i].font_id,
                10);
                break;
            case 2:
                xvt_font_set_size(obj[i].font_id,
                12);
            }
            set_bounds(win, i);
        }
        xvt_menu_set_item_enabled(win, M_FILE_NEW,
        TRUE);
    }
    static void
    set_bounds(WINDOW win, int n)
    {
        int ascent, descent,
        width;xvt_font_map(obj[n].font_id, win);
        xvt_font_get_metrics(obj[n].font_id, NULL,
        &ascent, &descent);
        xvt_dwin_set_font(win, obj[n].font_id);
        width = xvt_dwin_get_text_width(win,
        obj[n].text, -1);
        xvt_rect_set(&obj[n].bounds, obj[n].pos.h,
        obj[n].pos.v - ascent, obj[n].pos.h + width,
        obj[n].pos.v + descent);
}
```

E_HELP

Help-Request Event

Summary

```
typedef struct s_help {
    EVENT_TYPE type;   /* E_HELP */
    union {
       ...
      struct s_help {
        WINDOW obj;    /* target for help -- window,
                          dialog, or control */
        MENU_TAG tag;  /* target for help -- menu item */
        XVT_HELP_TID;  /* help topic, usually NULL_TID */
      } help;
      ...
    } v;
} EVENT;
```

Description

An `E_HELP` event is generated when the application user requests online help. Usually your application does not have to handle `E_HELP` explicitly, since the help system handles this event automatically and does not pass it on to your application's event handlers.

You can process the help event yourself if you're writing your own help system or creating special-case help services for certain containers.

Only one of the three members of the `s_help` structure is relevant for any single `E_HELP` event, depending on the type of object for which the user requested help.

If the user requests help for a window, dialog, or control:

- The `WINDOW` member of `s_help` contains the identifier of that object
- The `MENU_ITEM` member of `s_help` is `NULL`
- The `XVT_HELP_TID` member of `s_help` contains `NULL_TID`

If the user requests help for a menu item:

- The `MENU_ITEM` member of `s_help` is the identifier of the menu item for which help is requested
- The `WINDOW` member of `s_help` is `NULL_WIN`

- The XVT_HELP_TID member of s_help contains NULL_TID

If the user requests help for a specific topic (rather than for a specific GUI object):

- The XVT_HELP_TID member of s_help contains the topic identifier
- The WINDOW member of s_help is NULL_WIN
- The MENU_ITEM member of s_help is NULL

Implementation Note

Different user actions cause the E_HELP event, depending on the conventions of the native GUI system. For example, on MS-Windows, pressing the F1 key usually invokes help, whereas on the Macintosh the Help key does.

See Also

EVENT_TYPE

MENU_TAG

WINDOW

XVT_HELP_TID, NULL_TID

xvt_help_process_event

The "Events" and the "Hypertext Online Help" chapters in the *XVT Portability Toolkit Guide*

E_HSCROLL

Horizontal Scrollbar Events

Summary

```
typedef struct {         /* event description */
    EVENT_TYPE type;     /* type (E_HSCROLL) */
    union {
    ...
    struct {
    SCROLL_CONTROL what; /* site of activity */
    short pos;           /* thumb position */
    } scroll;
    ...
    } v;

} EVENT;
```

Description

XVT sends an E_HSCROLL event to a window's event handler to notify your application that the user has operated the horizontal scrollbar (which is part of the window's frame). Only windows with the WSF_HSCROLL or WSF_DECORATED flag set at creation time receive these events.

The v.scroll.what field of the event contains one of the SC_* constants, indicating which part of the scrollbar was manipulated. It is up to your application to define the semantics for scrollbar manipulations.

If XVT sets the v.scroll.what field to SC_THUMB or SC_THUMBTRACK, then it also sets the v.scroll.pos field to indicate the position to which the user dragged the thumb. For other scrollbar manipulations, v.scroll.pos is undefined. Keep in mind that the v.scroll.pos value always falls within the range you set by calling xvt_sbar_set_range and xvt_sbar_set_proportion.

Note: An E_HSCROLL event is not sent for the operation of scrollbar controls in a window; in that case, an E_CONTROL event would be sent.

See Also

E_CONTROL

EVENT_TYPE

E_VSCROLL

SCROLL_CONTROL

SC_* Values for SCROLL_CONTROL

WSF_* Options Flags

WSF_* Options Flags

xvt_sbar_get_pos

xvt_sbar_get_proportion

xvt_sbar_get_range

xvt_sbar_set_pos

xvt_sbar_set_proportion

xvt_sbar_set_range

The "Events" and the "Controls" chapters in the *XVT Portability Toolkit Guide*

E_MOUSE_DBL

Mouse Double-Click Event

Summary

```
typedef struct {       /* event description */
    EVENT_TYPE type;   /* type (E_MOUSE_DBL) */
    union {
    ...
    struct {
    PNT where;         /* location */
    BOOLEAN shift;     /* shift key down? */
    BOOLEAN control;   /* control/option key? */
    short button;      /* button */
    } mouse;
    ...
    } v;

} EVENT;
```

Description

XVT sends an `E_MOUSE_DBL` event to a window's event handler for an XVT window in response to the user double-clicking a mouse button while the mouse pointer is in the client area of the window. `E_MOUSE_DBL` events are *not* sent to dialog event handlers.

The location of the mouse pointer relative to the upper-left corner of the window is given by the `v.mouse.where` field. If the user holds down the Shift or Control key while clicking, then `v.mouse.shift` or `v.mouse.control` is set to `TRUE`, indicating the status of these keys.

The `v.mouse.button` field is set to "0" if it was the left button, "1" if it was the right button, or "2" if it was the middle button on a three-button mouse. Portable XVT programs should use buttons 0 and 1. Events for buttons numbered 2 or greater might occur on some XVT-supported platforms, but their use isn't portable.

Double-Click Definition

A double-click is defined as a button-down action that rapidly follows a button-up action. Each platform defines "rapidly" according to its own tolerances. XVT reports the button-up action separately as an `E_MOUSE_UP` event. A second `E_MOUSE_UP` follows the `E_MOUSE_DBL` as soon as the user lets up on the button. Hence, four events result (in this order) from a double-click:

```
E_MOUSE_DOWN
E_MOUSE_UP
E_MOUSE_DBL
E_MOUSE_UP
```

The application does not necessarily receive all four events. When one of the events occurs, the application might take action (such as bringing up a dialog box) that precludes receiving the other events. See the example below.

If your application must handle single-click events but ignore double-click events, you should either set an application flag when an `E_MOUSE_DBL` event occurs and then ignore `E_MOUSE_UP` events when that flag is set, or ignore `E_MOUSE_UP` events entirely.

Implementation Note

To emulate a right button press on XVT/Mac, the `v.mouse.button` is set to "1" if the user holds down the Command key while operating the (single) mouse button.

Mouse events are not sent to the event handler for the task window unless the application is running on XVT/Win32, and the attribute `ATTR_WIN_PM_DRAWABLE_TWIN` has been set.

See Also

`EVENT_TYPE`

`E_MOUSE_DOWN`

`E_MOUSE_MOVE`

`E_MOUSE_UP`

`PNT`

The "Events"chapter in the *XVT Portability Toolkit Guide*

Example:

In this example, when the user double-clicks within an object’s bounding rectangle, the application makes sure that object is selected (whether it already is or not) and then opens a dialog box that shows its point size.

```
static void do_double(WINDOW win, PNT where)
{
    int dbl_obj;
    long size;
    if ((dbl_obj = find_obj(where)) != NO_OBJ) {
    if (sel_obj != dbl_obj) {
    invert_selection(win);
    sel_obj = dbl_obj;
    invert_selection(win);
    }
    size = xvt_font_get_size(obj[sel_obj].font_id);
    xvt_dm_post_note("%d points", size);
    }

}
long XVT_CALLCONV1 win_eh(WINDOW win, EVENT *ep)
{
    switch (ep->type) {
    ...
    case E_MOUSE_DBL:
    do_double(win, ep->v.mouse.where);
    break;
    ...
    }

}
```

E_MOUSE_DOWN

Mouse-Down Event

Summary

```
typedef struct {       /* event description */
    EVENT_TYPE type;   /* type (E_MOUSE_DOWN) */
    union {
    ...
    struct {
    PNT where;         /* location */
    BOOLEAN shift;     /* shift key down? */
    BOOLEAN control;   /* control/option key? */
    short button;      /* button */
    } mouse;
    ...
    } v;

} EVENT;
```

Description

XVT sends an `E_MOUSE_DOWN` event to a window's event handler in response to the user clicking a mouse button while the mouse pointer is in the client area of the window. `E_MOUSE_DOWN` events are not sent to dialog event handlers.

The location of the mouse pointer relative to the upper-left corner of the window is given by the `v.mouse.where` field. If the user holds down the Shift or Control key while clicking, then `v.mouse.shift` or `v.mouse.control` is set to `TRUE`, indicating the status of these keys.

The `v.mouse.button` field will be set to "0" if it was the left button, "1" if it was the right button, or "2" if it was the middle button on a three-button mouse. Portable XVT programs should use buttons 0 and 1. Events for buttons numbered 2 or greater might occur in some XVT-supported platforms, but their use isn't portable.

Implementation Note

To emulate a right button press on XVT/Mac, the `v.mouse.button` is set to "1" if the user holds down the Command key while operating the (single) mouse button.

Mouse events are not sent to the event handler for the task window unless the application is running on XVT/Win32, and the attribute `ATTR_WIN_PM_DRAWABLE_TWIN` has been set.

See Also

```
EVENT_TYPE
E_MOUSE_DBL
E_MOUSE_MOVE
E_MOUSE_UP
PNT
```

The "Events" chapter in the *XVT Portability Toolkit Guide*

E_MOUSE_MOVE

Mouse-Movement Event

Summary

```
typedef struct {            /*event description */
    EVENT_TYPE type;        /* type (E_MOUSE_MOVE) */
    union {
         ...
        struct {
            PNT where;      /* location */
            BOOLEAN shift;  /* shift key down? */
            BOOLEAN control; /* control/option key? */
            short button;   /* button */
    } mouse;
         ...
    } v;
} EVENT;
```

Description

XVT sends an `E_MOUSE_MOVE` event to a window's event handler in response to the user moving the mouse pointer in the client area of an XVT window. Mouse events are *not* sent to dialog event handlers.

The location of the mouse pointer relative to the upper-left corner of the window is given by the `v.mouse.where` field. The `v.mouse.shift`, `v.mouse.control`, and `v.mouse.button` fields are not valid for this event.

`E_MOUSE_MOVE` events are also generated continuously when the mouse is trapped, even if the mouse isn't physically moved. In this case the coordinates given by the *where* member remain unchanged. This helps to implement automatic scrolling while the user drags the mouse.

Implementation Note

`E_MOUSE_MOVE` events are not sent to the event handler for `TASK_WIN` unless the application is running on XVT/Win32, and the attribute `ATTR_WIN_PM_DRAWABLE_TWIN` has been set.

See Also

```
E_MOUSE_DBL
E_MOUSE_DOWN
E_MOUSE_UP
EVENT_TYPE
PNT
TASK_WIN
xvt_win_release_pointer
xvt_win_trap_pointer
```

The "Events" chapter in the *XVT Portability Toolkit Guide*

E_MOUSE_UP

Mouse-Up Event

Summary

```
typedef struct {       /* event description */
    EVENT_TYPE type;   /* type (E_MOUSE_UP) */
    union {
    ...
    struct {
    PNT where;         /* location */
    BOOLEAN shift;     /* shift key down? */
    BOOLEAN control;   /* control/option key? */
    short button;      /* button */
    } mouse;
    ...
    } v;

} EVENT;
```

Description

XVT sends an `E_MOUSE_UP` event to a window's event handler in response to the user releasing a mouse button while the mouse pointer is in the client area of the window. Like other mouse events, `E_MOUSE_UP` events are *not* sent to dialog event handlers.

The location of the mouse pointer relative to the upper-left corner of the window is given by the `v.mouse.where` field. If the user holds down the Shift or Control keys while releasing the mouse, then the `v.mouse.shift` and `v.mouse.control` fields reflect that state. The `v.mouse.button` field indicates which mouse button was released.

The `v.mouse.button` field is set to "0" if it was the left button, "1" if it was the right button, or "2" if it was the middle button on a three-button mouse. Portable XVT programs should use buttons 0 and 1.

Events for buttons numbered 2 or greater might occur in some XVT-supported platforms, but their use isn't portable.

Implementation Note

On the Mac, if the user releases the mouse button while holding down the Command key, `v.mouse.button` is set to "1" indicating a "right button release."

Like other mouse events, `E_MOUSE_UP` is not sent to the event handler for `TASK_WIN` unless the application is running on XVT/Win32, and the attribute `ATTR_WIN_PM_DRAWABLE_TWIN` has been set.

See Also

```
E_MOUSE_DBL
E_MOUSE_DOWN
E_MOUSE_MOVE
EVENT_TYPE
PNT
TASK_WIN
```

The "Events" chapter in the *XVT Portability Toolkit Guide*

E_QUIT

Quit-Application Event

Summary

```
typedef struct {       /* event description */
    EVENT_TYPE type;   /* event type (E_QUIT) */
    union {
    ...
    BOOLEAN query;     /* query only? */
    ...
    } v;

} EVENT;
```

Description

XVT sends an `E_QUIT` event to the task event handler to notify your application that the user has initiated a system shutdown.

The `v.query` field is set to `TRUE` if the window system implements a multi-stage shutdown to get a consensus from all running applications that it is okay to shut down the window system. `v.query` is set to `FALSE` if the window system gives running applications no choice in the matter.

If your application gets an E_QUIT event with v.query set to TRUE, it should query the user about unsaved changes (by calling xvt_dm_post_ask, for example). If the user agrees to quit, your application should call xvt_app_allow_quit.

If your application gets an E_QUIT with v.query set to FALSE, then it will have already called xvt_app_allow_quit, and it should call xvt_app_destroy. Keep in mind that calling xvt_app_destroy closes all windows and dialogs in the application, but your application might only get an E_DESTROY event for the task window.

Be careful not to confuse the E_QUIT event with other events that might cause termination of your application. Specifically, E_QUIT is *not* sent when the user attempts to close the task window (E_CLOSE is sent). It is also *not* sent when the user chooses Quit from the File menu (E_COMMAND is sent). E_QUIT is sent by the native operating system to tell your application that the system is performing a system-wide shutdown. It is not an event that the user can directly generate.

Implementation Note

Only task window event handlers can receive E_QUIT events. E_QUIT events are generated only on platforms that can notify applications of a system-wide shutdown. XVT/Mac and XVT/Win32 are the only platforms that generate E_QUIT events.

See Also

E_CLOSE
E_COMMAND
EVENT
EVENT_TYPE
WINDOW
xvt_app_allow_quit
xvt_app_destroy
xvt_dm_post_ask

The "Events" chapter in the *XVT Portability Toolkit Guide*

Example

This code shows how to use E_QUIT:

```
long XVT_CALLCONV1 task_eh(WINDOW xdWindow,
     EVENT *xdEvent)
{
    switch (xdEvent->type) {
    ...
    case E_QUIT:
        if (xdEvent->v.query) {
            if (is_quit_ok())
                xvt_app_allow_quit();
        }
        else
            xvt_app_destroy();
        break;
    }
    return 0L;
}
```

E_SIZE

Resize-Window Event

Summary

```
typedef struct {       /* event description */
    EVENT_TYPE type;   /* event type (E_SIZE) */
    union {
    ...
    struct {
    short height;      /* new client height */
    short width;       /* new width */
    } size;
    ...
    } v;

} EVENT;
```

Description

The event handler for a window, dialog, or task window receives an `E_SIZE` event for any of the following reasons:

- XVT sends an `E_SIZE` event indicating the initial size to the event handler for a window when the user resizes the window using the border decorations. Only windows with the `WSF_SIZE` or `WSF_DECORATED` attribute set at creation time can have window resizing border decorations. On XVT/Win32, an `E_SIZE` might also be sent to the task window in response to a border decoration resize.

- XVT sends an `E_SIZE` event to the event handler for a window, dialog, or task window immediately following an `E_CREATE` event. Recall that the `E_CREATE` event is sent to notify your application that the window, dialog, or task window has been successfully created. Note that, on some platforms, performing certain operations during a window's `E_CREATE` (such as creating a dialog) might cause an `E_SIZE` event to be delivered to the window before the completion of the `E_CREATE` callback.
- XVT sends an `E_SIZE` event to a window or dialog event handler as a result of your application calling `xvt_vobj_move`.
- From the `width` and `height` fields of an `E_SIZE` event, you can know the new width and height of a `WINDOW`. Use the new size information to logically rearrange or scale the window contents. If your application adjusts child windows and controls to fit the new size, it should be done while processing this event. However, if your application adjusts the client area drawing (e.g., rewrapping text lines), then it should invalidate the client area by calling `xvt_dwin_invalidate_rect(win, NULL)`, and redraw the changed contents in the `E_UPDATE` that follows.

See Also

```
E_CREATE
E_SIZE
E_UPDATE
EVENT_TYPE
WINDOW
WSF_* Options Flags
WSF_* Options Flags
xvt_dwin_invalidate_rect
xvt_vobj_move
```

The "Event Ordering Rules" section of the "Events" chapter in the *XVT Portability Toolkit Guide*

Example

In the following code fragments, the scrollbar proportions are adjusted when the window is resized.

```
static void scroll_sync(WINDOW win, int height,
    int width)
{
    DOC *d;d =
    get_doc_data(win);xvt_sbar_set_range(win,
    VSCROLL, 0,
         d->nlines + height / d->line_height);
    xvt_sbar_set_range(win, HSCROLL, 0,
         d->maxwidth + width);
    xvt_sbar_set_proportion(win, VSCROLL,
         height / d->line_height);
    xvt_sbar_set_proportion(win, HSCROLL, width);
    xvt_sbar_set_pos(win, VSCROLL,
         d->org.v / d->line_height);
    xvt_sbar_set_pos(win, HSCROLL, d->org.h);
}
long XVT_CALLCONV1 win_eh(WINDOW win, EVENT *ep)
{
    switch (ep->type) {
    ...
    case E_SIZE:
        scroll_sync(win, ep->v.size.height, ep->
        v.size.width);
        break;
    ...
    }
}
```

E_TIMER

Interval-Elapsed Event

Summary

```
typedef struct {
    EVENT_TYPE type;     /* event type (E_TIMER) */
    union {
        struct {
             long id;
        } timer;
        ...
    }v;
} EVENT;
```

Description

XVT sends an `E_TIMER` event to the event handler of a window, dialog, or task window to notify your application that a specified time interval has elapsed. You can tell XVT to send these events and set the time interval by calling `xvt_timer_create`. XVT sets the

`v.timer.id` field of an `E_TIMER` event to the identifier returned by `xvt_timer_create`. By calling `xvt_timer_destroy`, your application can tell XVT to stop sending timer events to the event handler for the `WINDOW`.

The application program establishes timers on a per-window or per-dialog basis. Once a timer is set, `E_TIMER` events are sent to the event handler of the specified window or dialog at a specified regular time interval. The task window can also have timers.

Implementation Note

The number of timers that can be created for a window or dialog and the total number of timers that can be created are platform-specific.

On some platforms, it is possible that `E_TIMER` events will be sent even after the timer has been destroyed with xvt_timer_destroy. To protect your program from receiving events from recently destroyed timers, call `xvt_win_set_event_mask` (to mask out `E_TIMER` events) before calling `xvt_timer_destroy`.

See Also

```
EVENT_TYPE
WINDOW
xvt_timer_create
xvt_timer_destroy
xvt_win_set_event_mask
```

The "Events" chapter in the *XVT Portability Toolkit Guide*

E_UPDATE

Window-Update Event

Summary

```
typedef struct {       /* event description */
    EVENT_TYPE type;   /* event type (E_UPDATE) */
    union {
    struct{
    RCT rct;
    } update;
    ...
    } v;

} EVENT;
```

Description

XVT sends an `E_UPDATE` event to the event handler for an XVT window when part or all of the client area of the window needs to be redrawn. This event might be generated as the result of window creation, a change in the stacking order of the windows, user manipulation of the window, or by functions such as `xvt_dwin_scroll_rect` and `xvt_dwin_invalidate_rect`. Pending `E_UPDATE` events can also be expedited by `xvt_dwin_update`. `E_UPDATE` events are only sent to regular event handlers, not to dialog event handlers

XVT sets the `v.update.rct` field to the bounding rectangle of the area that must be redrawn. Note that, in general, the region that needs to be redrawn might be a subset of the bounding rectangle. Therefore, if you wish to optimize the redrawing of the damaged area, you can either use the bounding rectangle contained in v.update.rct, or call `xvt_dwin_is_update_needed`, which will operate directly on the update region, producing better optimization than is possible with the bounding rectangle.

Parameter Validity and Error Conditions

Some function calls cannot be made during an `E_UPDATE`. Enforcement of these rules is necessary for an application to be portable. For example, if an application developed on XVT/XM were allowed to call `xvt_vobj_move` during an update, that application would not work when moved to XVT/Win32.

If you call any of the following functions during an `E_UPDATE`, XVT issues an error:

xvt_app_process_pending_events
xvt_cb_* (except xvt_cb_has_format)
xvt_ctl_check_radio_button
xvt_ctl_create_def
xvt_ctl_set_checked
xvt_ctl_set_colors
xvt_ctl_set_font
xvt_ctl_set_text_sel
xvt_dlg_create_def
xvt_dlg_create_res
xvt_dm_post_ask
xvt_dm_post_error
xvt_dm_post_file_open
xvt_dm_post_file_save
xvt_dm_post_font_sel
xvt_dm_post_message
xvt_dm_post_note
xvt_dm_post_warning
xvt_dwin_invalidate_rect
xvt_dwin_scroll_rect
xvt_dwin_update
xvt_list_* (not including "query" functions)
xvt_menu_*
xvt_pmap_destroy
xvt_print_*
xvt_sbar_set_*
xvt_scr_set_focus_vobj
xvt_tx_add_par
xvt_tx_append
xvt_tx_clear
xvt_tx_create
xvt_tx_create_def
xvt_tx_destroy
xvt_tx_rem_par
xvt_tx_reset
xvt_tx_resume
xvt_tx_scroll_hor
xvt_tx_scroll_vert
xvt_tx_set_*
xvt_tx_suspend
xvt_vobj_destroy
xvt_vobj_move
xvt_vobj_set_enabled
xvt_vobj_set_palet
xvt_vobj_set_title
xvt_vobj_set_visible
xvt_win_create
xvt_win_create_def
xvt_win_create_res
xvt_win_set_caret_pos
xvt_win_set_caret_size
xvt_win_set_caret_visible
xvt_win_set_ctl_colors
xvt_win_set_ctl_font

xvt_win_set_doc_title

Note: Although xvt_dm_post_error and xvt_dm_post_warning are in the above list, the error signaled by calling them is posted by the XVT "last chance" error handler after the completion of the E_UPDATE event.

The XVT portable attribute ATTR_SUPPRESS_UPDATE_CHECK turns off checking for the above function calls during an E_UPDATE. Use this attribute if you absolutely *must* make one of these calls durning an E_UPDATE.

See Also

ATTR_SUPPRESS_UPDATE_CHECK
EVENT_TYPE
RCT
xvt_dm_post_error
xvt_dm_post_warning
xvt_dwin_invalidate_rect
xvt_dwin_is_update_needed
xvt_dwin_scroll_rect
xvt_dwin_update

The "Events" chapter in the *XVT Portability Toolkit Guide*

Example

In the following code fragments, the function do_update is called when an E_UPDATE event is received:

```
static void do_update(WINDOW win, RCT rct)
{
    DRAW_CTOOLS t;
    DOC *d;
    int i, ascent, descent, y,
    bottom;xvt_dwin_get_draw_ctools(win, &t);
    xvt_dwin_clear(win, t.backcolor);

    d = get_doc_data(win);
    xvt_dwin_set_font(win, d->font_id);
    xvt_dwin_get_font_metrics(win, NULL, &ascent,
    &descent);
    bottom = rct.bottom + d->line_height;/* start with
    first visible line */
    for (i = d->org.v / d->line_height, y = rct.top +
    ascent;
        i < d->nlines && y < bottom;
        i++, y += d->line_height) {
        rct.top = y - ascent;
        rct.bottom = y + descent;
        if (xvt_dwin_is_update_needed(win, &rct)) {
            xvt_dwin_draw_text(win, MGN - d->org.h, y,
            d->lines[i], -1);
        }
    }
}long XVT_CALLCONV1 win_eh(WINDOW win, EVENT *ep)
{
    RCT rct;switch (ep->type) {
    ...
    case E_UPDATE:
        xvt_vobj_get_client_rect(win, &rct);
        do_update(win, rct);
        break;
    ...
    }
}
```

E_USER

Application-Generated Event

Summary

```
typedef struct {
    EVENT_TYPE type;     /* E_USER */
    union {
        struct {
            long id;
            void *ptr;
        } user;
        ...
    } v;
} EVENT;
```

Description

XVT will not generate an `E_USER` event. However, your application can send `E_USER` events to itself by filling in the `type`, `v.user.id` and `v.user.ptr` field of an `EVENT` structure, and then calling `xvt_win_dispatch_event` to send that event to an event handler.

The application is free to use the `v.user.id` and `v.user.ptr` field in whatever way it chooses. However, a useful convention is to set `v.user.id` to the "sub-event type" and to set `v.user.ptr` to point to a data structure that will be sent with the event.

Note: The legal range of values for the `v.user.id` field is zero to `SHRT_MAX`. All values outside that range are reserved for future use in XVT.

See Also

`EVENT`

`EVENT_TYPE`

`SHRT_MAX`

`xvt_win_dispatch_event`

The "Events" chapter in the *XVT Portability Toolkit Guide*

E_VSCROLL

Vertical Scrollbar Event

Summary

```
typedef struct {       /* event description */
    EVENT_TYPE type;   /* type (E_VSCROLL) */
    union {
    ...
    struct {
    SCROLL_CONTROL what;
    short pos;
    } scroll;
    ... } v;

} EVENT;
```

Description

XVT sends an `E_VSCROLL` event to the event handler for a window in response to the user operating the vertical scrollbar on the window frame. Only windows with the `WSF_VSCROLL` or `WSF_DECORATED` flag set at creation time receive these events.

The `v.scroll.what` field of the event contains one of the `SC_*` constants, indicating what part of the scrollbar was manipulated. It is up to your application to define the semantics for scrollbar manipulations.

If XVT sets the `v.scroll.what` field to `SC_THUMB` or `SC_THUMBTRACK`, then it will also set the `v.scroll.pos` field to indicate the position to which the user dragged the thumb. For other scrollbar manipulations, `v.scroll.pos` is undefined. Keep in mind that the `v.scroll.pos` value always falls within the range that you set previously by calling `xvt_sbar_set_range` and `xvt_sbar_set_proportion`.

Note: An `E_VSCROLL` event is not sent for the operation of scrollbar controls in a window. In that case, an `E_CONTROL` event would be sent.

See Also

`E_CONTROL`

`E_HSCROLL`

`EVENT_TYPE`

`SCROLL_CONTROL`

`SC_* Values for SCROLL_CONTROL`

`WSF_* Options Flags`

`WSF_* Options Flags`

`xvt_sbar_get_pos`

`xvt_sbar_get_proportion`

`xvt_sbar_get_range`

`xvt_sbar_set_pos`

`xvt_sbar_set_proportion`

`xvt_sbar_set_range`

The "Events" and the "Controls" chapters in the *XVT Portability Toolkit Guide*

XVT Data Types

far
huge
near
A_* Values for ACCESS_CMD
RESP_* Values for ASK_RESPONSE
BOOLEAN
CB_* Values for CB_FORMAT
CBRUSH
COLOR
CONTROL_INFO
CPEN
CURSOR
DATA_PTR
DIRECTORY
DRAW_CTOOLS
M_* Values for DRAW_MODE
EOL_* Values for EOL_FORMAT
EVENT
EVENT_HANDLER
EVENT_MASK
EVENT_TYPE
FILE_SPEC
FL_* Values for FL_STATUS
GHANDLE
MENU_ITEM
MENU_TAG
PAT_* Values for PAT_STYLE
P_* Values for PEN_STYLE
PICTURE
PNT
PRINT_RCD
RCT
SCROLL_CALLBACK
SC_* Values for SCROLL_CONTROL
*SCROLL Values for SCROLL_TYPE
SLIST
SLIST_ELT
T_CNUM
T_CPOS
T_LNUM
T_PNUM
TXEDIT
U_* Values for UNIT_TYPE
WIN_DEF
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
XVT_BYTE
XVT_CODESET_MAP
XVT_COLLATE_FUNCTION
XVT_COLOR_ACTION
XVT_COLOR_COMPONENT

XVT_COLOR_TYPE
XVT_CONFIG
XVT_CXO
XVT_CXO_EVENT_HANDLER
XVT_CXO_INSERTION
XVT_ENUM_CHILDREN
XVT_ERRID
XVT_ERRMSG
XVT_ERRMSG_HANDLER
SEV_* Values for XVT_ERRSEV
XVT_FNTID
XVT_FONT_ATTR_MASK
XVT_FONT_DIALOG
XVT_FONT_MAPPER
XVT_FONT_STYLE_MASK
XVT_FORMAT_HANDLER
XVT_HELP_* Values for XVT_HELP_FLAVOR
XVT_HELP_INFO
XVT_HELP_TID, NULL_TID
XVT_HTML_URL_INTERCEPT_HANDLER
XVT_IMAGE_* Values for XVT_IMAGE_FORMAT
XVT_IMAGE_ATTR
XVT_IOSTR_CONTEXT
XVT_IOSTR_RWFUNC
XVT_IOSTR_SZFUNC
XVT_IOSTREAM
XVT_MEM
XVT_NAV
XVT_NOTEBK_ENUM_PAGES
XVT_PALETTE
XVT_PALETTE_ATTR
XVT_PALLETE_* Values
XVT_PATTERN
XVT_PG_ORIENT
XVT_PG_SIZE
XVT_PG_UNITS
XVT_PIXMAP
XVT_PIXMAP_ATTR
XVT_PIXMAP_* Values
XVT_POPUP_ALIGNMENT
XVT_PRINT_FUNCTION
XVT_RES
XVT_UBYTE
XVT_WCHAR

far

Global-Pointer Keyword

Description

The `far` keyword is available on compilers for Intel-based processors. XVT defines `far` on other platforms so that applications using this keyword will compile. However, `far` does not appear in the XVT API.

See Also

`huge`
`near`

huge

Global-Pointer Keyword

Description

The `huge` keyword is used on compilers for Intel-based processors to declare a pointer that addresses more than 64KB of memory at a time. XVT defines `huge` on other platforms so that applications using this keyword will compile. Currently, no XVT API functions accept or return `huge` pointers. This keyword exists mainly to aid XVT in implementing XVT features.

See Also

`far`
`near`

near

Global-Pointer Keyword

Description

The `near` keyword is available on compilers for Intel-based processors. XVT defines `near` on other platforms so that

applications using this keyword will compile. However, `near` does not appear in the XVT API.

See Also

`far`
`huge`

ACCESS_CMD

CMD Parameter to xvt_tx_get_line

See Also

This topic is discussed under `A_* Values for ACCESS_CMD` in XVT Constants.

ASK_RESPONSE

Response From xvt_dm_post_ask

See Also

This topic is discussed under `RESP_* Values for ASK_RESPONSE` in XVT Constants.

BOOLEAN

Boolean

Summary

```
#define BOOLEAN short int;/* Boolean type */
```

Description

The type `BOOLEAN` and its two values, `TRUE` and `FALSE`, are used by XVT whenever a Boolean type or Boolean value is needed.

See Also

`FALSE`
`TRUE`

Example

```
BOOLEAN window_is_a_checkbox(win)
WINDOW win;
{
    if (xvt_vobj_get_type(win) == WC_CHECKBOX)
        return TRUE;
    else
        return FALSE;
}
```

CB_FORMAT

Clipboard Format

See Also

This topic is discussed under `CB_* Values for CB_FORMAT` in XVT Constants.

CBRUSH

Color Brush Tool

Summary

```
typedef struct {/* color brush tool */
    PAT_STYLE pat;/* pattern */
    COLOR color;/* color */

} CBRUSH;
```

Description

A `CBRUSH` is a pattern used for the interior of the following shapes: rectangle, rounded rectangle, oval, pie, and polygon. (A `CPEN` is used for the perimeter of shapes.) Colors are indicated by RGB values.

The `pat` field of a `CBRUSH` indicates the fill pattern used to fill the interiors, and might include solid as well as hatched patterns. The `color` field specifies the color of the hatch marks for hatched patterns, and the color used by solid brushes.

Set the current brush with `xvt_dwin_set_cbrush`.

See Also

```
COLOR
CPEN
DRAW_CTOOLS
PAT_* Values for PAT_STYLE
xvt_dwin_draw_oval
xvt_dwin_draw_pie
xvt_dwin_draw_polygon
xvt_dwin_draw_rect
xvt_dwin_draw_roundrect
xvt_dwin_get_draw_ctools
xvt_dwin_set_cbrush
xvt_dwin_set_draw_ctools
```

The "Drawing and Pictures" chapter in the XVT Portability Toolkit Guide

Example

```
WINDOW win;
CBRUSH brush;
...
brush.pat = PAT_SOLID;
brush.color = COLOR_LTGRAY;
xvt_dwin_set_cbrush(win, &brush);
```

COLOR

Color

Summary

```
typedef unsigned long COLOR;
```

Description

Objects of this type specify an RGB color to be used in drawing tools. Some of the commonly used colors are available as predefined constants.

The `COLOR` type packs one byte each for the red, green, and blue intensities for a 24-bit RGB color. Your application cannot use the high byte, but XVT uses it for predefined colors. The four bytes of a `COLOR` are arranged as follows:

Byte	Color
byte 3 (MSB)	Reserved for XVT
byte 2	RED intensity
byte 1	GREEN intensity
byte 0 (LSB)	BLUE intensity

For convenience, your application can use the `XVT_MAKE_COLOR` macro to construct a `COLOR` from its components.

See Also

`CBRUSH`
`COLOR_*, COLOR_INVALID Constants`
`CPEN`
`DRAW_CTOOLS`
`XVT_COLOR_COMPONENT`
`XVT_MAKE_COLOR`
`xvt_ctl_get_colors`
`xvt_ctl_set_colors`
`xvt_dwin_clear`
`xvt_dwin_set_draw_ctools`
`xvt_dwin_set_back_color`
`xvt_dwin_set_cbrush`
`xvt_dwin_set_cpen`
`xvt_dwin_set_draw_ctools`
`xvt_dwin_set_fore_color`
`xvt_image_fill_rect`
`xvt_image_get_clut`
`xvt_image_get_pixel`
`xvt_image_set_clut`
`xvt_image_set_pixel`
`xvt_palet_add_colors`
`xvt_palet_get_colors`
`xvt_win_get_ctl_colors`
`xvt_win_set_ctl_colors`

The "Drawing and Pictures" chapter in the *XVT Portability Toolkit Guide*

CONTROL_INFO

Information About Activated Control

Summary

```
typedef struct s_ctlinfo {
    WIN_TYPE type;
    WINDOW win;
    union {
        struct s_scroll {
            SCROLL_CONTROL what;
            short pos;
        } scroll;
        struct s_edit {
            BOOLEAN focus_change;
            BOOLEAN active;
        } edit;
        struct s_lbox {
            BOOLEAN dbl_click;
        } lbox;
        struct s_listedit {
            BOOLEAN focus_change;
            BOOLEAN active;
        } listedit;
    } v;
} CONTROL_INFO;
```

Description

A structure of this type is passed to a window or dialog's event handler as part of the event structure for `E_CONTROL` event.

See Also

```
E_CONTROL
SCROLL_CONTROL
WINDOW
WIN_TYPE
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

CPEN

Color Pen Tool

Summary

```
typedef struct {/* color pen tool */
    short width;/* width */
    PAT_STYLE pat;/* pattern */
    PEN_STYLE style;/* style */
    COLOR color;/* color */

} CPEN;
```

Description

Objects of this type describe a color pen for drawing lines, but not for filling in shapes that have an interior (use a `CBRUSH` for that). Given a `CPEN`, you set it into a window's drawing tools with `xvt_dwin_set_cpen`. Each window has its own current `CPEN`.

The allowable values for `color`, `pat`, and `style` are discussed under the topics `COLOR`, `PAT_STYLE`, and `PEN_STYLE`.

Implementation Note

In systems that distinguish between pen width and height, for the logical pen point the `width` in pixels is used for both the height and the width of the pen. XVT logical pens always attempt to maintain constant aspect ratio on the actual physical pen. However, this may not always be possible since pixel dimensions may not be equal in height and width.

See Also

```
CBRUSH
COLOR
CPEN
DRAW_CTOOLS
PAT_* Values for PAT_STYLE
P_* Values for PEN_STYLE
xvt_dwin_get_draw_ctools
xvt_dwin_set_cbrush
xvt_dwin_set_cpen
xvt_dwin_set_draw_ctools
XVT_FAST_WIDTH
```

Example

```
WINDOW win;
CPEN pen;
...
pen.pat = PAT_SOLID;
pen.color = COLOR_LTGRAY;
pen.width = 2;
pen.style = P_SOLID;
xvt_dwin_set_cpen(win, &pen);
```

CURSOR

Cursor Type

Summary

```
typedef short CURSOR;
```

Description

`CURSOR` is the type used to represent the mouse cursor. The cursor is the pointer or other shape that indicates the current mouse position. Each XVT window has a current cursor that you can set either to a standard shape, described under `CURSOR_*` options, or to a shape that's defined as a resource.

You need to set the cursor for a window just once--XVT automatically takes care of setting it to the designated shape as the cursor moves from window to window.

See Also

```
CURSOR_* Options
xvt_scr_hide_cursor
xvt_scr_set_busy_cursor
xvt_win_get_cursor
xvt_win_set_cursor
```

DATA_PTR

Pointer to Arbitrary Data

Summary

```
typedef char* DATA_PTR;
```

Description

This data type defines a pointer to an arbitrary data type. Typically, this data type is used in XVT API functions that allocate and free memory.

See Also

```
PTR_LONG
XVT_BYTE
xvt_mem_alloc
xvt_mem_free
xvt_mem_realloc
xvt_mem_rep
xvt_mem_zalloc
```

Example

```
{
    long *tmp;
        ...
    /* xvt_mem_alloc returns a DATA_PTR so its return
        values must be cast */
    tmp = (long *) xvt_mem_alloc(sizeof(long) * 10);
        ...
    xvt_mem_free((DATA_PTR) tmp);
}
```

DIRECTORY

Directory

Summary

```
typedef struct { ... } DIRECTORY;
```

Description

A `DIRECTORY` is an abstract object that identifies a directory on the local file system. Its internals are hidden and are not necessarily a character string. However, to convert between `DIRECTORY`s and strings, you can call `xvt_fsys_convert_dir_to_str` and `xvt_fsys_convert_str_to_dir`.

XVT uses `DIRECTORY` objects to refer to directories in order to ensure portability across systems that differ in how they specify directories. XVT provides various functions to manipulate directories. (For details, see the topics listed below.)

Implementation Note

On XVT/Mac, the usage of DIRECTORYs works with hard drives, floppies, folders, and both HFS and MFS volumes. On XVT/Win32, it works with different drive letters.

See Also

FILE_SPEC
xvt_dm_post_file_open
xvt_dm_post_file_save
xvt_fsys_convert_dir_to_str
xvt_fsys_convert_str_to_dir
xvt_fsys_get_default_dir
xvt_fsys_get_file_attr
xvt_fsys_get_dir
xvt_fsys_set_dir
xvt_fsys_set_file_attr

Example

The topics xvt_fsys_get_default_dir and xvt_fsys_get_dir contain examples of the use of DIRECTORYs.

DRAW_CTOOLS

Color Drawing Tool Set

Summary

```
typedef struct {/* color drawing tools */
    CPEN pen;/* color pen */
    CBRUSH brush;/* color brush */
    DRAW_MODE mode;/* drawing mode */
    COLOR fore_color;/* foreground color */
    COLOR back_color;/* background color */
    BOOLEAN opaque_text;/* is text opaque*/

} DRAW_CTOOLS;
```

Description

This structure defines a set of color drawing tools: a color pen and brush, a drawing mode, foreground and background colors, and an indication of whether text should be opaque or not. You can get the current color tools with xvt_dwin_get_draw_ctools, and set them all at once with xvt_dwin_set_draw_ctools.

If the BOOLEAN member opaque_text is TRUE, text is drawn with an opaque background that covers what's underneath--it's not

necessary to first clear the area with a call to `xvt_dwin_draw_rect`. By default, `opaque_text` is `FALSE`.

A `DRAW_CTOOLS` object is position-independent--it can be written to a file or the clipboard and read in correctly to another address space.

See Also

```
CBRUSH
COLOR
CPEN
DRAW_MODE
DRAW_CTOOLS
xvt_app_get_default_ctools
xvt_dwin_get_draw_ctools
xvt_dwin_set_draw_ctools
```

The "Drawing and Pictures" chapter in the *XVT Portability Toolkit Guide*

Example

In this example, bracketing calls to `xvt_dwin_get_draw_ctools` and `xvt_dwin_set_draw_ctools` are used to avoid disturbing the current tool settings for the window.

```
RCT rct;
DRAW_CTOOLS save_tools, t;xvt_dwin_get_draw_ctools(win,
&save_tools);
xvt_app_get_default_ctools(&t);
t.pen.width = XVT_FAST_WIDTH;
t.brush.pat = PAT_HOLLOW;
t.mode = M_XOR;
xvt_dwin_set_draw_ctools(win, &t);rct.left = 100;
rct.top = 200;
rct.right = 100;
rct.bottom = 200;
xvt_dwin_draw_rect(win, &rct);
xvt_dwin_set_draw_ctools(win, &save_tools);
```

DRAW_MODE

Drawing Mode

See Also

This topic is discussed under `M_* Values for DRAW_MODE` in XVT Constants.

EOL_FORMAT

Terminator found by `xvt_str_find_eol`

See Also

This topic is discussed under `EOL_* Values for EOL_FORMAT`" in XVT Constants.

EVENT

Event Prototype

Summary

```
typedef struct s_event {
    EVENT_TYPE type;
    union {
        struct s_mouse {
            PNT where;
            BOOLEAN shift;
            BOOLEAN control;
            short button;
            XVT_INT32 scroll_x;
            XVT_INT32 scroll_y;
        } mouse;
        struct s_char {
            XVT_WCHAR ch;
            BOOLEAN shift;
            BOOLEAN control;
            BOOLEAN virtual_key;
            unsigned long modifiers;
        } chr;
        BOOLEAN active;
        BOOLEAN query;
        struct s_scroll_info {
            SCROLL_CONTROL what;
            short pos;
        } scroll;
        struct s_cmd {
            MENU_TAG tag;
            BOOLEAN shift;
            BOOLEAN control;
        } cmd;
        struct s_size {
            short height;
            short width;
        } size;
        struct s_efont {
            XVT_FNTID font id;
        } font;
        struct s_ctl {
            short id;
            CONTROL_INFO ci;
        } ctl;
        struct s_update {
            RCT rct;
        } update;
        struct s_timer {
            long id;
        } timer;
```

```
        struct s_user {
            long id;
            void *ptr;
        } user;
        struct s_help {
            window obj;
            MENU_TAG tag;
            XVT_HELP_TID tid;
        }help;
    } v;
} EVENT;
```

Description

XVT sends a structure of this type as the second parameter to your event handling functions whenever XVT generates an event. When your event handler receives an event, it can look at the `type` field of the `EVENT` structure to determine what kind of event it received. In addition to that information, the following substructures of the union also contain event-dependent information:

- `mouse` contains information for an `E_MOUSE_DOWN`, `E_MOUSE_DBL`, `E_MOUSE_MOVE`, `E_MOUSE_SCROLL`, or `E_MOUSE_UP` event
- `chr` contains information for an `E_CHAR` event
- `active` contains information for an `E_FOCUS` event
- `query` contains information for an `E_QUIT` event
- `scroll` contains information for an `E_HSCROLL` or `E_VSCROLL` event
- `cmd` contains information for an `E_COMMAND` event
- `size` contains information for an `E_SIZE` event
- `font` contains information for an `E_FONT` event
- `ctl` contains information for an `E_CONTROL` event
- `update` contains information for an `E_UPDATE` event
- `timer` contains information for an `E_TIMER` event
- `user` contains information for an `E_USER` event
- `help` contains information for an `E_HELP` event

The events `E_CREATE`, `E_DESTROY`, and `E_CLOSE` do not contain additional information in the `EVENT` structure.

See Also

XVT Events
EVENT_HANDLER
EVENT_MASK

```
EVENT_TYPE
XVT_CALLCONV*
xvt_event_get_font
xvt_event_set_font
xvt_win_dispatch_event
```

The "Events" chapter in the *XVT Portability Toolkit Guide*

EVENT_HANDLER

Window Event Handler Function Prototype

Summary

```
typedef long (* EVENT_HANDLER)
(WINDOW win, EVENT *ep);
```

Description

This type definition defines the prototype for event handling functions. Variables that will store event handling function pointers should be defined to be of type `EVENT_HANDLER`.

For platforms without prototyping compilers, `EVENT_HANDLER` is typedef'd to be a function with no arguments.

XVT functions that take a parameter of type `EVENT_HANDLER` are the `xvt_*_create_*` functions, `xvt_win_set_handler`, and `xvt_win_get_handler`. They can be called with the name of a function matching this prototype or a variable of this type.

Implementation Note

To insure portability across all platforms, you should include the macro `XVT_CALLCONV1` in the prototypes and headers of all callback functions used in XVT applications, including those of `EVENT_HANDLER`s. This macro defines the linkage conventions used in building XVT libraries.

See Also

EVENT
WINDOW
xvt_app_create
xvt_dlg_create_def
xvt_dlg_create_res
xvt_win_create
xvt_win_create_def
xvt_win_create_res
xvt_win_get_handler
xvt_win_set_handler

EVENT_MASK

Event Mask

Summary

```
typedef unsigned long EVENT_MASK;
```

Description

Variables of type EVENT_MASK restrict the events that are sent to window and dialog event handlers. You can set a bitwise-OR'd combination of event mask constants (see EM_* Constants) in an EVENT_MASK variable and pass it to xvt_win_set_event and the window/dialog creation functions: xvt_dlg_create_def, xvt_dlg_create_res, xvt_win_create, xvt_win_create_def, and xvt_win_create_res.

When calling these functions, you normally pass an EVENT_MASK variable set to a value of EM_ALL for no event restrictions. Setting it to EM_NONE causes no events to be sent to the event handler for the WINDOW. To get the current mask setting, call xvt_win_get_event_mask.

See Also

EM_* Constants
EVENT
EVENT_TYPE
xvt_dlg_create_def
xvt_dlg_create_res
xvt_win_create
xvt_win_create_def
xvt_win_create_res
xvt_win_get_event_mask
xvt_win_set_event_mask

The "Events" chapter in the *XVT Portability Toolkit Guide*

EVENT_TYPE

Event Type

See Also

This topic is discussed under `EVENT_TYPE` in the XVT Events.

FILE_SPEC

Filename Specification

Summary

```
typedef struct {/* file specification */
    DIRECTORY dir;/* directory */
    char type[6];/* file type/extension */
    char name[SZ_FNAME + 1];/* filename */
    char creator[6];/* file creator */

} FILE_SPEC;
```

Description

Structures of this type specify filenames in a portable way. A complete specification consists of the opaque `DIRECTORY` structure `dir`, a `NULL`-terminated file type, and a `NULL`-terminated filename `name`. The creator is not required on all platforms.

Implementation Note

Under NTFS and UNIX, file types are extensions, and are limited to six bytes. On XVT/Mac, the most standard types (e.g., `TEXT`) are six bytes. XVT/Mac is the only platform that uses the creator member of `FILE_SPEC`.

See Also

DIRECTORY
FL_* Values for FL_STATUS
SZ_FNAME
xvt_app_get_file
xvt_dm_post_file_open
xvt_dm_post_file_save
xvt_fsys_convert_dir_to_str
xvt_fsys_convert_str_to_dir
xvt_fsys_get_file_attr
xvt_fsys_rem_file
xvt_fsys_set_file_attr
xvt_help_open_helpfile

FL_STATUS

File Dialog Result

See Also

This topic is discussed under FL_* Values for FL_STATUS in XVT Constants.

GHANDLE

Global Memory Handle

Summary

```
typedef unsigned long GHANDLE;
```

Description

Objects of this type identify global memory blocks. You get one by calling xvt_gmem_alloc or xvt_gmem_realloc; you use it in calls to xvt_gmem_free, xvt_gmem_lock, xvt_gmem_realloc, xvt_gmem_get_size, and xvt_gmem_unlock.

You can assume that GHANDLEs are at least 32 bits in length, but you cannot assume anything about their internals.

You should not attempt to pass a GHANDLE to another application, nor should you save one in a file to be read back in by a future invocation of the application that allocated it. You cannot put a GHANDLE on the clipboard either; XVT has special clipboard functions for putting

data on the clipboard. If you violate these rules, your application might still work since XVT can't always enforce them, but your code might not port successfully to other XVT implementations.

See the topic `xvt_gmem_alloc` for a discussion of global memory.

Implementation Note

An XVT `GHANDLE` is not necessarily the same as a Mac or MS-Windows memory handle. You must never treat it as a pointer to a pointer, as you can with Mac handles.

See Also

```
xvt_gmem_alloc
xvt_gmem_free
xvt_gmem_get_size
xvt_gmem_lock
xvt_gmem_realloc
xvt_gmem_unlock
```

MENU_ITEM

Menu Item

Summary

```
typedef struct s_mitem {
    MENU_TAG tag;/* menu tag */
    char *text;/* text to appear in menu */
    short mkey;/* mnemonic */
    unsigned enabled: 1;/* enabled? */
    unsigned checked: 1;/* checked? */
    unsigned checkable: 1;/* checkable? */
    unsigned separator: 1;/* separator? */
    struct s_mitems *child;/* pointer to submenu */
    ...

} MENU_ITEM;
```

Description

Arrays of `MENU_ITEM`s can be linked together in a tree fashion to describe a menubar, all of its submenus, and the items within those submenus.

A menubar is a horizontal list of submenus, usually located below the titlebar of a window. A submenu is a vertical list of items that can be pulled down by the user. The items in a submenu are either commands that the user can choose, or submenus. Putting submenus

in other submenus is how "cascading" or "hierarchical" menus are implemented.

The menubar is represented by an array of contiguous `MENU_ITEM` structures, with one extra structure at the end with a `tag` field of zero, to terminate the array. Within that array, each `MENU_ITEM` structure has its `child` field set to point to an array of `MENU_ITEM`s representing a submenu.

Each submenu, in turn, follows the same format as the menubar, except that each element of the `MENU_ITEM` array can represent either a command or another submenu. Elements representing commands have the `child` field set to `NULL`, whereas elements representing submenus have the `child` field set to point to another array of `MENU_ITEM`s representing a submenu. This menu hierarchy can be continued to an arbitrary level.

Each element of a `MENU_ITEM` array has the following fields set:

`tag`

Contains the menu tag. The last `MENU_ITEM` in an array has a tag of zero.

`text field`

Contains the `NULL`-terminated text that will appear on the menubar or pull-down menu for an item.

`mkey field`

Used on those systems that allow the keyboard to be used universally for all menu items (i.e., XVT/Win32 and XVT/XM). On those systems, one letter in the menu item text is designated as the mnemonic. That letter should be the value of the `mkey` field. For example, if the menu item is Search, then the ASCII value for an "a" (97) should be used, even though the user has to first press the Alt key.

`enabled bit`

Indicates whether the menu item is enabled or not. Menu items can also be enabled or disabled dynamically with `xvt_menu_set_item_enabled`.

`checked bit`

Indicates whether the menu item has a check mark beside it or not. Menu items can also be checked or unchecked dynamically with `xvt_menu_set_item_checked`.

`checkable` bit

Indicates whether the menu item is able to be checked or not. This has a visual effect on some systems, such as leaving space for a check mark. You can set this only for submenu items.

`separator` bit

Indicates whether the menu item is a separator line used to delimit related groups of menu items. If this bit is set, then the `text`, `mkey`, `enabled`, `checked`, and `checkable` fields are all ignored. You should set this only in submenu items.

When allocating new `MENU_ITEM`s, the entire structure must be all zeros because the specific definition for a particular XVT implementation can include additional undocumented members after `child`. The `xvt_mem_zalloc` function is useful for this.

See Also

`MENU_TAG`
`xvt_mem_zalloc`
`xvt_menu_set_item_checked`
`xvt_menu_set_item_enabled`
`xvt_menu_set_tree`
`xvt_menu_get_tree`
`xvt_res_free_menu_tree`
`xvt_res_get_menu`

MENU_TAG

Menu-Item Tag

Summary

```
typedef short MENU_TAG;/* menu-item tag */
```

Description

Values of type `MENU_TAG` are used to refer to submenus or to individual menu items.

A submenu vertical list of items that can be pulled down by the user. The items in a submenu are either additional submenus or individual items (i.e., commands) that the user can choose. Your application can use values of this type as the case expression for any switch statement. It is typical for an XVT application to do so when processing `E_COMMAND` events.

The functions that take MENU_TAG arguments are xvt_help_get_menu_assoc, xvt_help_set_menu_assoc, xvt_menu_set_item_enabled, xvt_menu_set_item_checked, and xvt_menu_set_item_title.

The special tag FONT_MENU_TAG refers to the Font/Style menu taken as a whole. You can use this constant in a call to xvt_menu_set_item_enabled.

There is no way to refer to the individual items on the Font/Style menu, but you can check the ones that describe a particular XVT_FNTID with the function xvt_menu_set_font_sel.

All menu tags defined by your application must fall within the range (1 .. MAX_MENU_TAG) because XVT's predefined menu tags are above that range.

For the predefined menu constants for the standard Edit, File, and Help menus, see the M_EDIT_*, M_FILE_*, M_HELP_* Menu Tags.

See Also

E_COMMAND
FONT_MENU_TAG
M_EDIT_*, M_FILE_*, M_HELP_* Menu Tags
MENU_TAG
XVT_FNTID
xvt_help_get_menu_assoc
xvt_help_set_menu_assoc
xvt_menu_set_font_sel
xvt_menu_set_item_checked
xvt_menu_set_item_enabled
xvt_menu_set_item_title

The "Menus" chapter in the *XVT Portability Toolkit Guide*

Example

```
static void do_menu(WINDOW xdWindow, EVENT *xdEvent)
{
    MENU_TAG tag = xdEvent->v.cmd.tag;
    switch (tag) {
    case M_FILE_NEW:
        build_window(W_DOC);
        break;
    case M_FILE_OPEN:
        /* file open processing */
        break;
    case M_FILE_SAVE:
        /* file save processing */
        break;
    case M_FILE_CLOSE:
        xvt_vobj_destroy(win);
        break;
    case M_FILE_QUIT:
        xvt_app_destroy();
        break;
    ...
    }
}
```

PAT_STYLE

Pattern Style

See Also

This topic is discussed under `PAT_* Values for PAT_STYLE` in XVT Constants.

PEN_STYLE

Pen Style

See Also

This topic is discussed under `P_* Values for PEN_STYLE` in XVT Constants.

PICTURE

Encapsulated Picture

Summary

```
typedef long PICTURE;
```

Description

Values of this type reference an encapsulated picture. A `PICTURE` is an opaque data type that stores the result of a series of drawing operations.

Implementation Note

Only the XVT/Mac platform stores `PICTURE`s as Mac PICTs, which are a "meta-file" like sequence of drawing operations, and can be arbitrarily scaled. On XVT/Win32 and XVT/XM, a `PICTURE` is a bitmap, which can be scaled, but with a loss of resolution. Of course, none of these underlying data types can be accessed portably.

See Also

```
xvt_cb_put_data
xvt_dwin_close_pict
xvt_dwin_draw_pict
xvt_pict_create
xvt_pict_destroy
xvt_pict_lock
xvt_pict_unlock
```

The "Drawing and Pictures" chapter in the *XVT Portability Toolkit Guide*

PNT

Point

Summary

```
typedef struct {/* mathematical point */
    short v;/* vertical (y) coord. */
    short h;/* horizontal (x) coord. */

} PNT;
```

Description

Objects of this type are used whenever XVT needs to refer to a mathematical point, consisting of x- and y-coordinates, in the `h` and `v` members, respectively. The actual values are meaningful only if you know what coordinate system they are relative to.

Structures of this type that are passed to XVT functions, such as `xvt_vobj_translate_points`, must refer to a pixel-coordinate system. However, point structures kept around for your own use can be in the coordinate system of your choosing.

Implementation Note

Don't assume that an XVT `PNT` is the same as a Mac `Point`, a MS-Windows `POINT`, or any other similar structure in the underlying window system.

See Also

```
xvt_dwin_draw_aline
xvt_dwin_draw_line
xvt_dwin_draw_polygon
xvt_dwin_draw_polyline
xvt_dwin_draw_set_pos
xvt_rect_get_pos
xvt_rect_has_point
xvt_rect_set_pos
xvt_vobj_translate_points
xvt_win_set_caret_pos
```

The "Coordinate Systems" chapter in the *XVT Portability Toolkit Guide*

PRINT_RCD

Print Record

Summary

```
typedef struct { ... } PRINT_RCD;
```

Description

Objects of this type keep track of the current printing status. A `PRINT_RCD` contains information about the current page setup, and, depending on the platform, can also contain information about the job status as it progresses and the currently selected printer.

The contents of a PRINT_RCD are opaque to your application. In fact, the declaration of PRINT_RCD in the XVT header file is a counterfeit declaration supplied so that applications can declare pointers to a PRINT_RCD. However, applications can't directly do anything with the object itself, nor can they use variables of type PRINT_RCD. They can only declare pointers.

You get a PRINT_RCD with a call to xvt_print_create. A PRINT_RCD is a "flat" data structure, so you can save it to a file and read it back using the size returned by xvt_print_create. If you load from a file, then you should check its validity with a call to xvt_print_is_valid before passing it to other XVT functions. In addition, you can display a dialog box to let the user modify the PRINT_RCD's contents with a call to xvt_dm_post_page_setup.

See Also

xvt_dm_post_font_sel
xvt_dm_post_page_setup
xvt_fmap_get_family_sizes
xvt_fmap_get_family_styles
xvt_fmap_get_familysize_styles
xvt_fmap_get_familystyle_sizes
xvt_palet_create
xvt_print_close_page
xvt_print_create
xvt_print_create_win
xvt_print_destroy
xvt_print_is_valid
xvt_print_open_page

The "Printing" chapter in the *XVT Portability Toolkit Guide*

RCT

Rectangle

Summary

```
typedef struct {/* mathematical rect */
    short top;/* top coordinate */
    short left;/* left coordinate */
    short bottom;/* bottom coordinate */
    short right;/* right coordinate */

} RCT;
```

Description

Objects of this type contain the coordinates of rectangles, in any coordinate system.

See Also

```
xvt_ctl_create
xvt_dwin_draw_arc
xvt_dwin_draw_image
xvt_dwin_draw_oval
xvt_dwin_draw_pict
xvt_dwin_draw_pie
xvt_dwin_draw_pmap
xvt_dwin_draw_rect
xvt_dwin_draw_roundrect
xvt_dwin_get_clip
xvt_dwin_invalidate_rect
xvt_dwin_is_update_needed
xvt_dwin_open_pict
xvt_dwin_scroll_rect
xvt_dwin_set_clip
xvt_image_fill_rect
xvt_image_get_from_pmap
xvt_image_transfer
xvt_pict_create
xvt_print_get_next_band
xvt_rect_*
xvt_tx_create
xvt_tx_get_view
xvt_vobj_get_client_rect
xvt_vobj_get_outer_rect
xvt_vobj_move
xvt_win_create
```

The "Coordinate Systems" chapter in the XVT Portability Toolkit Guide

SCROLL_CALLBACK

Text Edit Scroll Callback Function Prototype

Summary

```
typedef void(* SCROLL_CALLBACK)
    (TXEDIT tx, T_LNUM org_line, T_LNUM nlines,
    T_CPOS org_offset);
```

Description

This is the type of the `fcn` argument to `xvt_tx_set_scroll_callback`. Variables that will store scroll callback function pointers should be defined as type `SCROLL_CALLBACK`. The `xvt_tx_set_scroll_callback` function takes a parameter of type `SCROLL_CALLBACK`. This XVT function can be called with the name of a function matching this prototype or with a variable of this type.

Implementation Note

To insure portability across all platforms, you should include the macro `XVT_CALLCONV1` in the prototypes and headers of all callback functions used in XVT applications, including those of `SCROLL_CALLBACK`s. This macro defines the linkage conventions used in building XVT libraries on all IBM-compatible platforms. On other platforms, it is defined as an empty string.

See Also

`XVT_CALLCONV*`
`xvt_tx_set_scroll_callback`

SCROLL_CONTROL

Scrollbar Component

See Also

This topic is discussed under `SC_* Values for SCROLL_CONTROL` in XVT Events.

SCROLL_TYPE

Type of Scrollbar

See Also

This topic is discussed under `*SCROLL Values for SCROLL_TYPE` in XVT Constants.

SLIST

String List

Summary

```
typedef struct { ... } *SLIST;
```

Description

Objects of this type refer to `SLIST`s, which are linked lists of strings and associated data. They are used by `xvt_list_add` and the `xvt_slist_*` functions. `SLIST`s are created by `xvt_scr_list_wins` and `xvt_res_get_str_list`.

See Also

```
xvt_fsys_list_files
xvt_list_add
xvt_list_get_all
xvt_list_get_sel
xvt_res_get_str_list
xvt_scr_list_wins
xvt_slist_*
```

SLIST_ELT

String List Element

Summary

```
typedef struct { ... } *SLIST_ELT;
```

Description

Objects of this type reference elements of an `SLIST`. They are used when an application must refer to a particular element of the `SLIST`, such as when inserting or deleting an item.

See Also

```
SLIST
xvt_slist_add_at_elt
xvt_slist_get
xvt_slist_get_data
xvt_slist_get_first
xvt_slist_get_next
xvt_slist_rem
```

T_CNUM

Text Edit Character Number

Summary

```
typedef unsigned short T_CNUM;
```

Description

This type is used throughout the text edit system for numbering characters. The first character in a sequence is always numbered zero.

See Also

```
T_CPOS
T_LNUM
T_PNUM
xvt_tx_get_num_chars
xvt_tx_get_sel
xvt_tx_get_tabstop
xvt_tx_set_sel
xvt_tx_set_tabstop
```

"Controls" in the *XVT Portability Toolkit Guide*

T_CPOS

Text Edit Character Position

Summary

```
typedef unsigned short T_CPOS;
```

Description

This type is used throughout the text edit system for numbering character positions in pixels. The first pixel position in a sequence is always numbered zero.

See Also

```
T_CNUM
T_LNUM
T_PNUM
xvt_tx_get_origin
```

"Controls" in the *XVT Portability Toolkit Guide*

T_LNUM

Text Edit Line Number

Summary

```
typedef unsigned short T_LNUM;
```

Description

This type is used throughout the text edit system for numbering lines. The first line in a sequence is always numbered zero.

See Also

```
T_CNUM
T_CPOS
T_PNUM
xvt_tx_get_line
xvt_tx_get_num_chars
xvt_tx_get_num_lines
xvt_tx_get_num_par_lines
xvt_tx_get_origin
xvt_tx_get_sel
xvt_tx_set_sel
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

T_PNUM

Text Edit Paragraph Number

Summary

```
typedef unsigned short T_PNUM;
```

Description

This type is used throughout the text edit system for numbering paragraphs. The first paragraph in a sequence is always numbered zero.

See Also

```
T_CNUM
T_CPOS
T_LNUM
xvt_tx_add_par
xvt_tx_append
xvt_tx_get_line
xvt_tx_get_num_chars
xvt_tx_get_num_par_lines
xvt_tx_get_num_pars
xvt_tx_get_origin
xvt_tx_get_sel
xvt_tx_rem_par
xvt_tx_set_par
xvt_tx_set_sel
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

TXEDIT

Text Edit Object

Summary

```
#define TXEDIT WINDOW
```

Description

A value of this type is returned by `xvt_tx_create` or `xvt_tx_create_def`. It is used in subsequent text edit system calls to identify the text edit object on which to operate. It is equivalent to an XVT `WINDOW`.

If a text edit object has been created by `xvt_win_create_res` or `xvt_win_create_def`, then its `TXEDIT` value can be retrieved by calling `xvt_win_get_tx` and passing it the ID of the text edit object of interest and its parent `WINDOW`.

See Also

```
WINDOW
xvt_tx_*
xvt_win_create_def
xvt_win_create_res
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

UNIT_TYPE

Identify Coordinate System used for `WIN_DEF` Elements

See Also

This topic is discussed under `U_* Values for UNIT_TYPE` in XVT Constants.

WIN_DEF

Specify Window, Dialog, and Control Creation

Summary

```
typedef struct s_win_def {
    WIN_TYPE wtype;/* WC_* or WO_* type */
    RCT rct;
    char *text;
    UNIT_TYPE units;
    XVT_COLOR_COMPONENT * ctlcolors;
    union {
        struct s_win_def_win {/* WINDOW's */
            short int menu_rid;/* menu resource id */
            MENU_ITEM *menu_p;/* pointer to menu tree */
            long flags;/* WSF_* flags */
            XVT_FNTID ctl_font_id;/* control font id */
        } win;struct s_win_def_dlg {/* DIALOG's */
            long flags;/* WSF_* flags */
            XVT_FNTID ctl_font_id;/* control font id */
        } dlg;struct s_win_def_ctl {/* CONTROL's */
            short int ctrl_id;
            short int icon_id;/* for icons only */
            long flags;/* CTL_* flags */
            XVT_FNTID font_id;/* logical font */
        } ctl;struct s_win_def_tx {/* textedit objects */
            unsigned short attrib;/* TX_* flags */
            XVT_FNTID font_id;/* logical font */
            short margin;/* right margin */
            short limit;/* max chars */
            short int tx_id;/* text ID */
        } tx;
    } v;
} WIN_DEF;
```

Description

Structures of this type specify window, dialog, and control creation to the following functions: `xvt_ctl_create_def`, `xvt_dlg_create_def`, `xvt_tx_create_def`, and `xvt_win_create_def`. The specific use of `WIN_DEF` structures as it relates to each of these functions is more completely described in the section for each function.

`xvt_ctl_create_def` and `xvt_tx_create_def` are passed a single `WIN_DEF` structure describing the control or text edit object to be created.

In contrast, `xvt_dlg_create_def` and `xvt_win_create_def` accept arrays of `WIN_DEF` structures. Both of these functions expect the following things in these structures:

- The first element of the `WIN_DEF` array must describe the window or dialog itself
- The following elements of the array must describe controls within the window or dialog
- The final element of the array must be a terminator identified by a `WIN_DEF` structure whose `wtype` field is set to `W_NONE`

`xvt_res_get_win_def` and `xvt_res_get_dlg_def` return arrays of `WIN_DEF` structures. The arrays of `WIN_DEF` structures loaded by these functions are appropriate for `xvt_win_create_def` and `xvt_dlg_create_def`. Arrays of `WIN_DEF`s returned by these two functions can be freed by calling `xvt_res_free_win_def`.

Implementation Note

If you create your own `WIN_DEF` strucutre, you must intialize unused fields to zero. You may use `xvt_mem_zmalloc` to allocate `WIN_DEF` array and set the values to zero.

See Also

MENU_ITEM
RCT
UNIT_TYPE
W_*, WC_*, WD_*, Values for WIN_TYPE
WIN_TYPE
XVT_COLOR_COMPONENT
XVT_FNTID
xvt_ctl_create_def
xvt_dlg_create_def
xvt_mem_zalloc
xvt_res_free_win_def
xvt_res_get_dlg_def
xvt_res_get_win_def
xvt_tx_create_def
xvt_win_create_def

WIN_TYPE

Window-Type

This topic is discussed under W_*, WC_*, WD_*, Values for WIN_TYPE in XVT Constants.

WINDOW

Window Descriptor

Summary

```
typedef long WINDOW;/* window descriptor */
```

Description

Objects of this type identify XVT windows. A WINDOW can be a regular window, a dialog, a control, a pixmap, or a print window. It can also be one of the container windows, TASK_WIN or SCREEN_WIN.

Your application gets valid window objects when it calls any of the window, dialog, or control creation functions. It gets a valid print window when it calls xvt_print_create_win. It gets a pixmap window when it calls xvt_pmap_create. It can also use the predefined container windows TASK_WIN or SCREEN_WIN at any time.

In addition, your application can call `xvt_win_get_ctl` to retrieve the parent `WINDOW` for a control in a window or dialog.

Once your application has a valid window object, then it can pass it to any of the XVT functions that take a `WINDOW`, subject to the rules of that function.

Implementation Note

Direct information about the contents of a window descriptor is unavailable to the application. In particular, the application must not assume that a `WINDOW` is a `WindowPtr` or `HWND`. For information on gaining access to native window types, see `ATTR_NATIVE_WINDOW` and the *XVT Platform-Specific Books*.

See Also

```
ATTR_NATIVE_WINDOW
TASK_WIN
SCREEN_WIN
xvt_ctl_*
xvt_dlg_create_def
xvt_dlg_create_res
xvt_dm_post_font_sel
xvt_dwin_*
xvt_errmsg_get_*
xvt_font_get_win
xvt_font_map
xvt_list_*
xvt_menu_*
xvt_pmap_create
xvt_pmap_destroy
xvt_print_create_win
xvt_sbar_*
xvt_scr_get_focus_topwin
xvt_scr_get_focus_vobj
xvt_scr_set_focus_vobj
xvt_timer_create
xvt_tx_create
xvt_tx_create_def
xvt_vobj_*
xvt_win_*
```

The *XVT Platform-Specific Books*

XVT_BYTE

Arbitrary Data

Summary

```
typedef char XVT_BYTE
```

Description

The use of `char` or `char*` for non-string one-byte storage should be replaced by this type. The existing `DATA_PTR` type should be replaced by `XVT_BYTE*` where appropriate.

See Also

```
DATA_PTR
XVT_UBYTE
```

XVT_CODESET_MAP

Character Codeset Mapping Descriptor

Summary

```
typedef struct { ... } *XVT_CODESET_MAP;
```

Description

Objects of this type refer to Character Codeset Maps, which are mapping tables that define the mapping of the character value from one character codeset to another character codeset. The specific character codesets to be used are defined at the time the XVT_CODESET_MAP object is created by the function xvt_str_create_codeset_map.

See Also

```
xvt_str_create_codeset_map
xvt_str_destroy_codeset_map
xvt_str_translate_codeset
```

XVT_COLLATE_FUNCTION

Application-supplied String Collation Function Prototype

Summary

```
typedef long (* XVT_COLLATE_FUNCTION)(const char* mbs1,
     const char *mbs2);
```

Description

This type defines the prototype for application-supplied functions that collate strings. Variables that store application-supplied string collation function pointers must be of this type.

Your application can use the function `xvt_vobj_set_attr` to set the `ATTR_COLLATE_HOOK` hook function. The "value" parameter of the call to `xvt_vobj_set_attr` must be a variable of type `XVT_COLLATE_FUNCTION` but cast to a `long`. Similarly, `xvt_vobj_get_attr` returns a variable of type `XVT_COLLATE_FUNCTION` cast to a `long` when called to inquire the current value of the `ATTR_COLLATE_HOOK` attribute.

If your application requires a collation algorithm different than that provided by the default collation function supplied by the toolkit, you must use an application-supplied collation function.

Implementation Note

To insure portability across all platforms, you should include the macro `XVT_CALLCONV1` in the prototypes and headers of all callback functions used in XVT applications, including those of the `XVT_COLLATE_FUNCTION`s. This macro defines the linkage conversions used in building XVT libraries.

See Also

```
ATTR_COLLATE_HOOK
XVT_CALLCONV*
xvt_str_collate
xvt_str_collate_ignoring_case
xvt_vobj_get_attr
xvt_vobj_set_attr
```

The "Multibyte Character Sets and Localization" chapter in the *XVT Portability Toolkit Guide*

XVT_COLOR_ACTION

Color Setting Action

Summary

```
typedef enum s_xvt_color_action {
    XVT_COLOR_ACTION_SET,/* set the colors */
    XVT_COLOR_ACTION_UNSET/* unset the colors */
} XVT_COLOR_ACTION;
```

Description

This action is used for control colors. You use this type in `xvt_clt_set_colors` and `xvt_win_set_ctl_colors` to set or unset the control color components.

See Also

```
XVT_COLOR_COMPONENT
xvt_ctl_set_colors
xvt_win_set_ctl_colors
```

XVT_COLOR_COMPONENT

Color Component Types

Summary

```
typedef s_xvt_color_component {
     XVT_COLOR_TYPE type; /* color component being
                defined */
    COLOR color;/* RGB color value */
} XVT_COLOR_COMPONENT;
```

Description

This type contains the XVT color values used in the various control color access functions. Structures of this type usually appear in an array. The last element in the array must have a `type` field of `XVT_COLOR_NULL`.

See Also

```
COLOR
XVT_COLOR_*
XVT_COLOR_TYPE
xvt_ctl_get_colors
xvt_ctl_set_colors
xvt_win_get_ctl_colors
xvt_win_set_ctl_colors
```

XVT_COLOR_TYPE

Control Color Component

Summary

```
typedef unsigned long XVT_COLOR_TYPE;
```

Description

This type is used for defining control color components in an `XVT_COLOR_COMPONENT` structure. It assigns values of `XVT_COLOR_*` to variables of type `XVT_COLOR_TYPE`.

See Also

```
XVT_COLOR_*
XVT_COLOR_COMPONENT
xvt_ctl_get_colors
xvt_ctl_set_colors
xvt_win_get_ctl_colors
xvt_win_set_ctl_colors
```

XVT_CONFIG

System-Initialization Structure

Summary

```
typedef struct s_xvt_config {
    short menu_bar_ID;/* task menubar ResID */
    short about_box_ID;/* default aboutbox ResID */
    char *base_appl_name;/* application's "filename" */
    char *appl_name;/* application's name */
    char *taskwin_title;/* title for task window */
} XVT_CONFIG;
```

Description

This data type provides system initialization information to `xvt_app_create`.

`menu_bar_ID`

Specifies the resource ID of a menubar that will be used as the task menubar. The task menubar will be available to the user at certain times during the life cycle of the application. This must be set to a valid menubar resource ID.

`about_box_ID`

Can either be set to zero for your application to use XVT's default About box, or it can be set to the resource ID of an alternate About box dialog that you have specified in URL. For more information, see `xvt_dm_post_about_box`.

`base_appl_name`

Specifies the base filename (without extension) to be used when XVT looks for **.uid** files.

`appl_name`

Specifies the application name that will be prepended to the titles of document windows when `xvt_win_set_doc_title` is called.

`taskwin_title`

Specifies the title of the task window for platforms where the task window has a physical representation. Currently, only XVT/Win32 and XVT/XM use this task window title.

To ensure that all values are in a default state, the application should `memset` this structure to 0 before it is initialized. Use `ATTR_TASKWIN_TITLE_RID` and `ATTR_APPL_NAME_RID` to load the task window title and application name from resources.

Implementation Note

On XVT/Mac, `appl_name` specifies the name that is displayed in the "About appl_name..." item at the top of the Apple menu.

On XVT/XM, `base_appl_name` must be set to the name of the **.uid** file.

See Also

```
ATTR_APPL_NAME_RID
ATTR_TASKWIN_TITLE_RID
TASK_WIN
xvt_app_create
xvt_dm_post_about_box
xvt_win_set_doc_title
```

The "GUI Components" and the "Multibyte Character Sets and Localization" chapter in the *XVT Portability Toolkit Guide*

XVT_CXO

Container Extension Object

Summary

```
typedef long XVT_CXO;
```

Description

Objects of this type identify XVT Container Extension Objects (CXO's). An `XVT_CXO` is used to modify the behavior of a container (an XVT WINDOW with an event handler). `XVT_CXO`'s have their own event handlers and messaging system. These objects can be created using `xvt_cxo_create`.

Implementation Note

Direct information about the contents of a window descriptor is unavailable to the application. All communication with an `XVT_CXO` should be done through the supplied API calls.

See Also

xvt_cxo_*

XVT_CXO_EVENT_HANDLER

Container Extension Object Event Handler Function Prototype

Summary

```
typedef long (* XVT_CXO_EVENT_HANDLER)( XVT_CXO cxo,
EVENT * ep );
```

Description

This type definition defines the prototype for CXO event handling functions.

XVT functions that take a parameter of type `XVT_CXO_EVENT_HANDLER` are `xvt_cxo_create` and `xvt_cxo_set_event_handler`. The function `xvt_cxo_get_event_handlef0 will return this type.`

Implementation Note

To insure portability across all platforms, you should include the macro `XVT_CALLCONVl` in the prototypes and headers of all callback functions used in XVT applications, including those of `XVT_CXO_EVENT_HANDLERs`. This macro defines the linkage convention used in building XVT libraries.

See Also

EVENT
XVT_CXO

XVT_CXO_INSERTION

Insertion Location for Container Extension Objects

See Also

This topic is discussed under `XVT_CXO_POS_* Values for XVT_CXO_INSERTION` in XVT Constants.

XVT_DISPLAY_TYPE

Value for `ATTR_DISPLAY_TYPE`

See Also

This topic is discussed under `ATTR_DISPLAY_TYPE` in XVT Portable Attributes.

XVT_ENUM_CHILDREN

Prototype for Application-supplied Functions Passed to `xvt_win_enum_wins`

Summary

```
typedef BOOLEAN (* XVT_ENUM_CHILDREN)(WINDOW child,
     long data)
```

`WINDOW child`

Child of `parent_win` specified in call to `xvt_win_enum_wins`

`long data`

Application supplied data

Description

This is the prototype for application-supplied functions supplied to `xvt_win_enum_wins`. Application-supplied callback functions used by `xvt_win_enum_wins` must follow this signature. The application registers this callback function by passing its address to the `xvt_win_enum_wins` function. The callback function, since it is application-supplied, can perform any application action. The application must be careful to avoid writing callbacks that cause unintended recursion.

Return Value

To continue enumeration, the callback function must return `TRUE`. To stop enumeration, it must return `FALSE`.

See Also

`WINDOW`
`xvt_win_enum_wins`

XVT_ERRID

Error Message Identifier

Summary

```
typedef unsigned long XVT_ERRID;
```

Description

Error messages are identified using an opaque data type `XVT_ERRID`, which is composed of several fields:

- Message number (16 bits unsigned short)
- Standard message flag (1 bit: distinguishes predefined, standard toolkit messages from the ones defined by an `xvt_errmsg_sig` call)
- Message category minor portion (4 bit)
- Message category major portion (4 bit)

You should not make any assumptions about the individual field position within the identifier. Always use the `xvt_errid_*` macros to access `XVT_ERRID`.

See Also

```
xvt_errid_create_*
xvt_errid_get_*
xvt_errid_is_*
```

The "Diagnostics and Debugging" chapter in the *XVT Portability Toolkit Guide*

XVT_ERRMSG

Error Message Object

Summary

```
typedef struct { ... } *XVT_ERRMSG;
```

Description

The error message object is available only to an error handler. Because the error message object is represented by an opaque identifier, you must access it by using the `xvt_errmsg_get_*` inquiry functions.

The `xvt_errmsg_get_*` inquiry functions give an error handler complete information about an error signalled by one of the `xvt_errmsg_sig` functions.

See Also

```
ATTR_ERRMSG_FILENAME
xvt_errid_get_*
xvt_errmsg_pop_handler
xvt_errmsg_push_handler
xvt_errmsg_sig
```

XVT_ERRMSG_HANDLER

Error Message Handler Function Prototype

Summary

```
typedef BOOLEAN(* XVT_ERRMSG_HANDLER)
     (XVT_ERRMSG err_msg, DATA_PTR context);
```

`XVT_ERRMSG err_msg`

Error message object describing a signaled error.

`DATA_PTR context`

`NULL` or error handler context provided in the `xvt_errmsg_push_handler` call.

Description

This type definition defines the prototype for error message handling functions. Variables that will store error message handling function pointers should be defined as type `XVT_ERRMSG_HANDLER`.

The error message handler is a function invoked by the error messaging facility when the `xvt_errmsg_sig` or `xvt_errmsg_sig_std` functions are called.

The handler examines the passed-in error message object using `xvt_errmsg_get_*` calls. After taking appropriate application-specific action, the handler either dismisses the error signal (returning `TRUE`), or passes the error to the next handler (returning `FALSE`). If the error is passed on, it ultimately reaches the XVT-provided "last chance" error handler.

XVT functions that take a parameter of type `XVT_ERRMSG_HANDLER` are `xvt_errmsg_push_handler` and `xvt_errmsg_pop_handler`. They can be called with the name of a function matching this prototype or with a variable of this type.

Return Value

`TRUE` if an error has been caught (handled) by the handler; `FALSE` if the error has been passed on to the next error handler.

Parameter Validity and Error Conditions

To prevent deadlock, any errors signaled from within an error handler are delivered only to the next error handler. You can use this feature to signal errors.

Implementation Note

To insure portability across all platforms, you should include the macro `XVT_CALLCONV1` in the prototypes and headers of all callback functions used in XVT applications, including those of `XVT_ERRMSG_HANDLER`s. This macro defines the linkage conventions used in building XVT libraries.

See Also

```
ATTR_ERRMSG_HANDLER
DATA_PTR
XVT_CALLCONV*
XVT_ERRMSG
xvt_errmsg_get_*
xvt_errmsg_pop_handler
xvt_errmsg_push_handler
xvt_errmsg_sig
xvt_errmsg_sig_std
```

Example

The following error handler performs application cleanup on fatal errors, letting the XVT "last chance" handler present the message to the user. Any other errors are simply masked out, with no action.

```
BOOLEAN XVT_CALLCONV1 my_handler(XVT_ERRMSG msg,
    DATA_PTR context)
{
    if (xvt_errmsg_get_sev_id (msg) == SEV_FATAL)) {
        app_perform_cleanup();
        return FALSE;/* let XVT report error */
    }else
        return TRUE;/* hide errors and warnings from user
*/
}
```

XVT_ERRSEV

Error Severity Type

See Also

This topic is discussed under `SEV_* Values for XVT_ERRSEV` in XVT Constants.

XVT_FNTID

Object That Identifies a Logical Font

Summary

```
typedef struct { ... } *XVT_FNTID;
```

Description

Objects of this type identify logical fonts. Your application gets a valid logical font when it calls `xvt_dwin_get_font`, `xvt_font_create`, `xvt_menu_get_font_sel`, or `xvt_res_get_font`, or receives an `E_FONT` event. Once your application has a valid logical font, it can pass it to any of the XVT functions that take an `XVT_FNTID`, subject to that function's rules.

Implementation Note

An application cannot get direct information about the contents of an `XVT_FNTID`. The application can access logical font attributes only through the logical font access functions.

See Also

```
xvt_dm_post_font_sel
xvt_dwin_get_font*
xvt_dwin_set_font_*
xvt_event_get_font
xvt_event_set_font
xvt_font_*
xvt_menu_get_font_sel
xvt_menu_set_font_sel
xvt_res_get_font
xvt_tx_create
```

XVT_FONT_ATTR_MASK

Logical Font Attribute Type

Summary

```
typedef unsigned long XVT_FONT_ATTR_MASK;
```

Description

This attribute mask specifies logical font attribute types for the `XVT_FNTID` access functions. The logical font attribute mask is composed of one or more `XVT_FA_*` flag values that are OR'd together. You can use the `XVT_FA_*` constants in combined bit masks to specify multiple values.

These are the valid flags you can use:

```
#define XVT_FA_FAMILY ...   /* family */
#define XVT_FA_SIZE ...     /* size */
#define XVT_FA_STYLE ...    /* style */
#define XVT_FA_NATIVE ...   /* native descriptor */
#define XVT_FA_APP_DATA ... /* application data */
#define XVT_FA_WIN ...      /* window */
#define XVT_FA_ALL ...      /* all attributes*/
```

See Also

`XVT_FA_* Constants`
`XVT_FFN_* Constants`
`XVT_FNTID`
`xvt_font_copy`

XVT_FONT_DIALOG

Application-Supplied Font Selection Dialog Function Prototype

Summary

```
typedef BOOLEAN(* XVT_FONT_DIALOG)
    (WINDOW win, XVT_FNTID default_font_id,
    PRINT_RCD *precp, unsigned long reserved);
```

Description

This type definition defines the prototype for application-supplied functions that invoke the Font Selection dialog. Variables that will

store application-supplied Font-Selection-dialog-invoking function pointers should be defined to be of type XVT_FONT_DIALOG.

Your application can use the xvt_vobj_set_attr function to set the ATTR_FONT_DIALOG portable attribute. The "value" parameter would be a variable of type XVT_FONT_DIALOG (but cast to a long). Similarly, xvt_vobj_get_attr returns a variable of type XVT_FONT_DIALOG (cast to a long) when called to inquire the ATTR_FONT_DIALOG portable attribute.

Implementation Note

To insure portability across all platforms, you should include the macro XVT_CALLCONV1 in the prototypes and headers of all callback functions used in XVT applications, including those of XVT_FONT_DIALOGs. This macro defines the linkage conventions used in building XVT libraries.

See Also

ATTR_FONT_DIALOG
PRINT_RCD
WINDOW
XVT_CALLCONV*
XVT_FNTID
xvt_vobj_get_attr
xvt_vobj_set_attr

The "Font Selection Dialogs" section of the "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

XVT_FONT_MAPPER

Application-Supplied Font Mapper Function Prototype

Summary

```
typedef BOOLEAN(* XVT_FONT_MAPPER)
    (XVT_FNTID font_id);
```

Description

This type definition defines the prototype for application-supplied Logical Font Mapping functions. Variables that will store application-supplied font mapping function pointers should be defined to be of type XVT_FONT_MAPPER.

Your application can use the xvt_vobj_set_attr function to set the ATTR_FONT_MAPPER portable attribute. The "value" parameter would

be a variable of type XVT_FONT_MAPPER (but cast to a long). Similarly, xvt_vobj_get_attr returns a variable of type XVT_FONT_MAPPER (cast to a long) when called to inquire the ATTR_FONT_MAPPER portable attribute.

Implementation Note

To ensure portability across all platforms, you should include the macro XVT_CALLCONV1 in the prototypes and headers of all callback functions used in XVT applications, including those of XVT_FONT_MAPPERs. This macro defines the linkage conventions used in building XVT libraries.

See Also

ATTR_FONT_MAPPER
XVT_CALLCONV*
XVT_FNTID
xvt_vobj_get_attr
xvt_vobj_set_attr

The "Application-Supplied Font Mappers" section of the "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

XVT_FONT_STYLE_MASK

Logical Font Style Type

Summary

```
typedef unsigned long XVT_FONT_STYLE_MASK;
```

Description

This style mask specifies logical font style for the XVT_FNTID access functions. The logical font style mask is composed of one or more XVT_FS_* flag values that are OR'd together. You can use the XVT_FS_* constants in combined bit masks to specify multiple values.

These are the valid flags you can use:

```
    #define XVT_FS_NONE .../* none */
    #define XVT_FS_BOLD .../* bold */
    #define XVT_FS_ITALIC .../* italic */
    #define XVT_FS_UNDERLINE .../* underline */
    #define XVT_FS_OUTLINE .../* outline */
    #define XVT_FS_SHADOW .../* shadow */
    #define XVT_FS_INVERSE .../* inverse */
    #define XVT_FS_BLINK .../* blinking */
    #define XVT_FS_STRIKEOUT .../* strikeout */
    #define XVT_FS_USER1 .../* for application use */
    #define XVT_FS_USER2 .../* for application use */
    #define XVT_FS_USER3 .../* for application use */
    #define XVT_FS_USER4 .../* for application use */
    #define XVT_FS_USER5 .../* for application use */
    #define XVT_FS_WILDCARD .../* used only in URL FONT
statement */
```

XVT provides the `XVT_FS_USER*` style flags for use by application-supplied font mappers. The `XVT_FS_WILDCARD` style is not useful in your application, but it is the resulting logical font style when you specify "`any`" style in an URL `font` statement.

The following list shows the availability of logical font style attributes on different platforms:

Style	Platform
`bold`	Available on all platforms
`italic`	Available on all platforms
`underline`	Available on XVT/Mac and XVT/Win32
`outline`	Available on XVT/Mac
`shadow`	Available on XVT/Mac
`strikeout`	Available on XVT/Win32

See Also

```
XVT_FONT_ATTR_MASK
XVT_FS_* Constants
xvt_dwin_get_font_style
xvt_dwin_get_font_style_mapped
xvt_dwin_set_font_style
xvt_fmap_get_family_styles
xvt_fmap_get_familysize_styles
xvt_fmap_get_familystyle_sizes
xvt_font_get_style
xvt_font_get_style_mapped
xvt_font_set_style
```

The "URL Font Mapper" and "Application-Supplied Font Mappers" sections of the "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

XVT_FORMAT_HANDLER

Prototype for Application-Supplied Functions for String Formatting

Summary

```
typedef const char *(*XVT_FORMAT_HANDLER)(WINDOW win,
    char *instr, int *start, int *end, void *data)
```

`WINDOW win`

The window or control containing the format function.

`char *instr`

The input string to format.

`int *start`

Return the cursor start position for WC_EDIT and WC_LISTEDIT control text selection.

`int *end`

Return the cursor end position for WC_EDIT and WC_LISTEDIT control text selection.

`void *data`

Application-supplied data.

Description

This is the prototype for application-supplied functions passed to xvt_vobj_set_formatter. This function can be attached to XVT windows, dialogs, and controls. The callback function is called when setting the text for the title of these objects or inserting text into certain controls. For WC_LISTBOX controls, the formatting function is invoked when xvt_list_add is called. For WC_EDIT and WC_LISTEDIT controls, the formatting function is invoked when xvt_vobj_set_title is called, Cut or Paste operations are performed, or a character is entered from the keyboard.

The formatter function gets passed the WINDOW object to which it is attached, the string to be tested, and the application data set by the application with xvt_vobj_set_formatter. The function also gets passed output parameters for setting the text selection for a WC_EDIT or WC_LISTEDIT control only.

The formatter function must test the input string and decide if formatting is necessary. If no formatting is needed, the function

should return the input string unchanged. If formatting is needed, the application should format the string, as desired, and return the new string. XVT is not responsible for allocating or freeing the memory for storing the new string. The application should either allocate global data space or use a data buffer passed as the data parameter to the formatter function. If the application wishes to terminate the current action (setting a window title), the function should return -1.

Return Value

In order for the title to be inserted in the WINDOW, the function must return a pointer to the newly formatted string (or the input string if no change is needed). If NULL or -1 is returned, the calling function returns without further action and the title will not be set.

See Also

```
XVT_PATTERN
xvt_pattern_create
xvt_pattern_destroy
xvt_pattern_match
xvt_pattern_format_string
xvt_vobj_get_formatter
xvt_vobj_set_formatter
```

XVT_HELP_FLAVOR

Configuration of the Help Viewer

See Also

This topic is discussed under `XVT_HELP_* Values for XVT_HELP_FLAVOR` in XVT Constants.

XVT_HELP_INFO

Help File Information Handle

Summary

```
typdef struct { ... } *XVT_HELP_INFO;
```

Description

XVT_HELP_INFO is an opaque type given to the application to identify the information contained in a compiled help file.

Applications receive an XVT_HELP_INFO value from xvt_help_open_helpfile. Every other help system function expects a value of this type as the first parameter.

See Also

xvt_help_*

XVT_HELP_TID, NULL_TID

Help Topic Identifier

Summary

```
typedef long XVT_HELP_TID#define NULL_TID ...
```

Description

A topic identifier is used to specify help file topics. Both the help file itself and the program must agree on a set of unique topic identifier values. A convenient way to ensure this is to place the identifiers in an #include file that is shared by the help file and the program source files.

NULL_TID specifies an undefined help topic.

See Also

xvt_help_*

XVT_HTML_URL_INTERCEPT_HANDLER

Prototype for URL Intercept Handler for HTML Control

Summary

```
typedef BOOLEAN (*XVT_HTML_URL_INTERCEPT_HANDLER)
    (WINDOW win,
    char **url)
```

WINDOW win

The HTML control assigned the URL intercept handler.

`char **url`

Pointer to the string containing the requested URL.

Description

Prototype for the application-defined Universal Resource Locator (URL) intercept handler passed to xvt_html_set_url_intercept. The URL intercept handler is called whenever a URL is to be set on an HTML control, either with xvt_html_set_url or when the user follows a link.

The URL passed into the URL intercept handler can be modified for the purpose of redirection. Because this parameter is a pointer to a character string, the existing character string must be freed using xvt_mem_free prior to reallocation of the new string using xvt_mem_alloc. Failure to free and reallocate or writing beyond the existing string bounds will result in memory leaks and potential memory protection faults.

Return Value

Return TRUE if the calling function is to further process and display the passed URL. Return FALSE if the calling function is to ignore the passed URL and not process the request.

See Also

```
WC_* Values for WIN_TYPE
xvt_html_get_url
xvt_html_set_url
xvt_html_get_url_intercept
xvt_html_set_url_intercept
```

Example

See example for xvt_html_set_url_intercept.

XVT_IMAGE

Image Data Object

Summary

```
typedef struct { ... } *XVT_IMAGE;
```

Description

Variables of this type contain images. Your application creates valid images using `xvt_image_create`, or by loading images from files

using the various `xvt_image_read_*` functions. `XVT_IMAGE` is an opaque data type. Your application can manipulate variables of this type only by calling `xvt_image_*` functions.

See Also

```
xvt_dwin_draw_image
xvt_image_*
xvt_palet_add_colors
xvt_res_get_image
```

XVT_IMAGE_ATTR

Attribute used in Image Object Creation

Summary

```
typedef struct { ... } XVT_IMAGE_ATTR;
```

Description

This attribute is not currently implemented; it is reserved for future use.

See Also

```
xvt_image_create
```

XVT_IMAGE_FORMAT

Color Format Enumerated Type for Images

See Also

This topic is discussed under `XVT_IMAGE_* Values for XVT_IMAGE_FORMAT` in XVT Constants.

XVT_IOSTR_CONTEXT

Opaque Type for Pointer to Stream Data Encapsulation

Summary

```
typedef struct { ... } *XVT_IOSTR_CONTEXT;
```

Description

The `XVT_IOSTREAM` object uses this type for the user-defined "context" of the stream data. It is typically defined as the address of a user-defined structure containing a stream data pointer and position indicator.

See Also

```
XVT_IOSTREAM
xvt_iostr_create_read
xvt_iostr_create_write
```

XVT_IOSTR_RWFUNC

Function Signature for Stream Read/Write Functions

Summary

```
typedef short(*XVT_IOSTR_RWFUNC)
    (XVT_IOSTREAM iostr, unsigned short nbytes,
    XVT_BYTE *buf);
```

`XVT_IOSTREAM iostr`

I/O stream to be read read or written.

`unsigned short nbytes`

Number of bytes needed for read or write operations.

`XVT_BYTE *buf`

Buffer to be written or read.

Description

This is the signature of the stream read and write functions. When creating user-defined `XVT_IOSTREAM` objects, you must provide functions matching this signature for reading from and writing to the stream context.

See Also

```
XVT_BYTE
XVT_IOSTREAM
xvt_iostr_create_read
xvt_iostr_create_write
xvt_iostr_get_context
```

XVT_IOSTR_SZFUNC

Function Signature for Stream Size Function

Summary

```
typedef long(*XVT_IOSTR_SZFUNC) (XVT_IOSTREAM
    iostr);

XVT_IOSTREAM iostr
```

I/O stream for which a size is to be returned.

Description

This is the signature of the stream size function. This function returns the size of the data stream (in bytes) and must be provided when user-defined `XVT_IOSTREAM` objects are created. The size value returned must always be the original creation size.

See Also

```
XVT_IOSTREAM
xvt_iostr_create_read
xvt_iostr_create_write
```

XVT_IOSTREAM

Input/Output Stream Object

Summary

```
typedef struct { ... } *XVT_IOSTREAM;
```

Description

Variables of this type contain information necessary for reading and writing sequential data streams (files or application-defined). Your

application creates valid stream objects using any of the xvt_iostr_create_* functions.

XVT_IOSTREAM is an opaque data type. Your application can manipulate variables of this type only by calling xvt_iostr_* functions.

See Also

xvt_image_read_bmp_from_iostr
xvt_image_read_macpict
xvt_image_read_xbm_from_iostr
xvt_image_read_xpm_from_iostr
xvt_image_write_bmp_to_iostr
xvt_image_write_macpict_to_iostr
xvt_iostr_create_fread
xvt_iostr_create_fwrite
xvt_iostr_create_read
xvt_iostr_create_write
xvt_iostr_destroy

XVT_MEM

Structure of Memory Manager Functions

Summary

```
typedef struct s_mem {
    DATA_PTR (*malloc) XVT_CC_ARGS((size_t size));
    VOID (*free) XVT_CC_ARGS((DATA_PTR data));
    DATA_PTR (*realloc) XVT_CC_ARGS((DATA_PTR data,
        size_t size));
    DATA_PTR (*zalloc) XVT_CC_ARGS((size_t size));
} XVT_MEM;
```

Description

This structure contains the addresses of the system-wide memory management functions that are called when the application invokes xvt_mem_alloc, xvt_mem_free, xvt_mem_realloc, or xvt_mem_zmalloc.

To reset the memory management functions from the default set of XVT-supplied functions to an application-specified set of functions, applications can create and initialize a variable of this type, and pass it to xvt_vobj_set_attr (using the ATTR_MEMORY_MANAGER attribute). When an application calls xvt_vobj_get_attr using the ATTR_MEMORY_MANAGER attribute, xvt_vobj_get_attr returns a pointer to a structure of type XVT_MEM (cast to long).

See Also

```
ATTR_MEMORY_MANAGER
DATA_PTR
xvt_vobj_get_attr
xvt_vobj_set_attr
```

XVT_NAV

Navigation Object

Summary

```
typedef long XVT_NAV;
```

Description

Objects of this type are XVT `WINDOW` navigation objects. Navigation objects allow users to navigate through GUI objects contained within a window using the keyboard. `W_DOC`, `W_PLAIN`, `W_DBL`, `W_MODAL`, and `W_NO_BORDER` are windows which may contain `XVT_NAV` objects.

See Also

```
xvt_nav_create
xvt_win_get_nav
```

XVT_NOTEBK_ENUM_PAGES

Callback Function Prototype for `xvt_notebk_enum_pages`

Summary

```
typedef BOOLEAN (* XVT_NOTEBK_ENUM_PAGES)
    (WINDOW notebk, short tab_no, short page_no,
    long data);
```

`WINDOW notebk`

Window ID of notebook control.

`short tab_no`

Tab number.

`short page_no`

Page number.

`long data`

Data to associate with tab and page.

Description

This type definition defines the required callback function for `xvt_notebk_enum_pages`.

The calback function is continuously called with `notebk`, `tab_no`, `page_no` and data being updated sequentially for each enumeration. The callbacks will continue until the last enumeration occurs or the callback returns `FALSE`.

Return Value

`TRUE` to continue next enumeration, if any; `FALSE` to discontinue before next enumeration, if any.

Implementation Note

To insure portability across all platforms, you should include the macro `XVT_CALLCONV1` in the prototypes and headers of all callback functions used in XVT applications, including those of `XVT_NOTEBK_ENUM_PAGES`. This macro defines the linkage conventions used in building XVT libraries.

See Also

`xvt_notebk_enum_pages`

Example

The following callback function checks the value of the associated data for the tab and page to see if is is set to `FALSE`.

```
BOOLEAN XVT_CALLCONV1 my_callback(WINDOW notebk,
    short tab_no, short page_no, long data)
{
    if (data == FALSE) {
        xvt_xm_post_note("Data is FALSE!");
        return FALSE; /* discontinue enumeration */
    }
    else
        return TRUE; /* continue enumeration */
}
```

XVT_PALETTE

Color Palette Object

Summary

```
typdef struct { ... } *XVT_PALETTE;
```

Description

`XVT_PALETTE` is a data type for manipulating color palettes. Color palettes map image colors onto colors available in the display hardware. `XVT_PALETTE` is an opaque data type; your application can only access and modify pixmaps through XVT functions.

See Also

```
XVT_PALLETE_* Values
xvt_pallet_*
xvt_vobj_get_palet
xvt_vobj_set_palet
```

The "Color Palettes" section of the "Portable Images" chapter in the *XVT Portability Toolkit Guide*

XVT_PALETTE_ATTR

Attribute used in Palette Object Creation

Summary

```
typedef struct { ... } XVT_PALETTE_ATTR;
```

Description

This attribute is not currently implemented; it is reserved for future use.

See Also

```
xvt_palet_create
```

XVT_PALLET_TYPE

Color Palette Types

See Also

This topic is discussed under `XVT_PALLETE_* Values` in XVT Constants.

XVT_PATTERN

Complex String Pattern Descriptor

Summary

```
typedef struct { ... } *XVT_PATTERN
```

Description

Objects of this type refer to complex string patterns, which are compiled parse trees of a string matching and formatting pattern.

See Also

```
XVT_FORMAT_HANDLER
xvt_pattern_create
xvt_pattern_destroy
xvt_pattern_match
xvt_pattern_format_string
xvt_vobj_get_formatter
xvt_vobj_set_formatter
```

XVT_PG_ORIENT

Page Orientation

Summary

```
typedef enum {
    PG_PORTRIAT,
    PG_LANGSCAPE
} XVT_PG_ORIENT;
```

Description

Enumerated type for identifing the page orientation.

See Also

`xvt_print_set_page_orient`

XVT_PG_SIZE

Page Dimensions

Summary

```
typedef struct {
    double width;
    double height;
    XVT_PG_UNITS pgunits;
} XVT_PG_SIZE;
```

Description

This structure contains page information that is passed in the function `xvt_print_set_page_size`. `XVT_PG_UNITS` is used to describe the units that the width and height use (inches or millimeters).

See Also

`XVT_PG_UNITS`
`XVT_PG_ORIENT`
`xvt_print_set_page_size`

XVT_PG_UNITS

Page Units

Summary

```
typedef enum {
    PG_IN,
    PG_MM
} XVT_PG_UNITS;
```

Description

Enumerated type for identifing the units within `XVT_PG_SIZE`. `PG_IN` describes inches and `PG_MM` describes millimeters.

XVT_PIXMAP

Color Image Object

Summary

```
typedef WINDOW XVT_PIXMAP;
```

Description

`XVT_PIXMAP` is a color image data type. `XVT_PIXMAPs` are almost equivalent to WINDOWs and are defined as such; however, not all graphics functions operate on pixmaps. `XVT_PIXMAP` is an opaque data type; your application can access and modify pixmaps only through XVT functions.

The following functions can accept a pixmap in addition to a window:

```
xvt_dwin_clear
xvt_dwin_draw_aline
xvt_dwin_draw_arc
xvt_dwin_draw_icon
xvt_dwin_draw_image
xvt_dwin_draw_line
xvt_dwin_draw_oval
xvt_dwin_draw_pic
xvt_dwin_draw_pie
xvt_dwin_draw_pmap
xvt_dwin_draw_polygon
xvt_dwin_draw_polyline
xvt_dwin_draw_rect
xvt_dwin_draw_roundrect
xvt_dwin_draw_set_pos
xvt_dwin_draw_text
xvt_dwin_get_draw_ctools
xvt_dwin_get_font_metrics
xvt_dwin_get_text_width
xvt_dwin_scroll_rect
xvt_dwin_set_back_color
xvt_dwin_set_cbrush
xvt_dwin_set_clip
xvt_dwin_set_cpen
xvt_dwin_set_draw_ctools
xvt_dwin_set_draw_mode
xvt_dwin_set_font
xvt_dwin_set_fore_color
xvt_dwin_set_std_cbrush
xvt_dwin_set_std_cpen
xvt_vobj_get_client_rect
xvt_vobj_get_data
xvt_vobj_get_outer_rect
xvt_vobj_get_parent
xvt_vobj_get_type
xvt_vobj_set_data
```

See Also

```
WINDOW
XVT_PIXMAP_* Values
xvt_dwin_*
xvt_image_get_from_pmap
xvt_pmap_create
xvt_pmap_destroy
xvt_vobj_*
```

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

XVT_PIXMAP_ATTR

Attribute used in Pixmap Object Creation

Summary

```
typedef struct { ... } XVT_PIXMAP_ATTR;
```

Description

This attribute is not currently implemented; it is reserved for future use.

See Also

`xvt_pmap_create`

XVT_PIXMAP_FORMAT

Color Image Types

See Also

This topic is discussed under `XVT_PIXMAP_* Values` in XVT Constants.

XVT_POPUP_ALIGNMENT

Aligns Popup Window

Summary

```
typedef enum {
    XVT_POPUP_CENTERED,
    XVT_POPUP_LEFT_ALIGNED,
    XVT_POPUP_RIGHT_ALIGNED,
    XVT_POPUP_OVER_ITEM
} XVT_POPUP_ALIGNMENT;
```

`XVT_POPUP_CENTERED`

Centered below specified position.

`XVT_POPUP_LEFT_ALIGN`

Left-aligned below specified position.

`XVT_POPUP_RIGHT_ALIGN`

Right-aligned below specified position.

`XVT_POPUP_OVER_ITEM`

Specified menu item centered over position.

Description

This type contains alignment information for popup windows.

See Also

`xvt_menu_popup`

XVT_PRINT_FUNCTION

Application-Supplied Printing Function Prototype

Summary

```
typedef BOOLEAN (* XVT_PRINT_FUNCTION)
     (long data);
```

Description

This type definition defines the prototype for application-supplied print functions. Variables that will store print function pointers should be defined to be of type `XVT_PRINT_FUNCTION`.

The XVT function that takes a parameter of type `XVT_PRINT_FUNCTION` is `xvt_print_start_thread`. You can call it with the name of a function matching this prototype or a variable of this type.

Implementation Note

To ensure portability across all platforms, you should include the macro `XVT_CALLCONV1` in the prototypes and headers of all callback functions used in XVT applications, including those of `XVT_PRINT_FUNCTIONs`. This macro defines the linkage conventions used in building XVT libraries.

See Also

`XVT_CALLCONV*`
`xvt_print_start_thread`
The "Printing" chapter in the XVT Portability Toolkit Guide

XVT_RES

Resource ID

Summary

```
typedef long XVT_RES; /* Resource ID */
```

Description

Objects of this type identify resource IDs. Your application gets a valid `XVT_RES` when calling `xvt_res_add_res`. This resource ID can then be used in `xvt_res_use_res` to indicate the active resouce.

Implementation Note

`xvt_res_use_res` returns the previously active resource ID. It is the programmers responsibility to keep such information.

See Also

```
xvt_res_add_res
xvt_res_use_res
xvt_res_remove_res
```

XVT_TREEVIEW_NODE

Treeview node

Summary

```
typedef void * XVT_TREEVIEW_NODE; /* Treeview node*/
```

Description

Objects of this type identify treeview nodes.

Implementation Note

See Also

```
xvt_res_add_res
xvt_res_use_res
xvt_res_remove_res
```

XVT_UBYTE

Non-string Unsigned One-byte Storage

Summary

```
typedef unsigned char XVT_UBYTE
```

Description

An external type for non-string unsigned one-byte storage. Any use of `unsigned char` or `unsigned char*` in this manner should be replaced by this type. The existing type `DATA_PTR` should be replaced by `XVT_UBYTE*` where appropriate.

See Also

```
DATA_PTR
XVT_BYTE
```

XVT_WCHAR

Wide Character Type

Summary

```
typedef wchar_t XVT_WCHAR
```

Description

An encapsulation of the wide character type. On some platforms, this may be typed to a `short` or some other integral type instead of a `wchar_t` type.

See Also

```
xvt_str_convert_mb_to_wc
xvt_str_convert_mbs_to_wcs
xvt_str_convert_wc_to_mb
xvt_str_convert_wcs_to_mbs
xvt_str_convert_wchar_to_lower
xvt_str_convert_wchar_to_upper
```

XVT Constants

A_* Values for ACCESS_CMD
CB_* Values for CB_FORMAT
CHAR_MAX
COLOR_*, COLOR_INVALID Constants
CTL_FLAG_* Options
CURSOR_* Options
DEFAULT_*_MENU Values
DIR_TYPE
DLG_* Control IDs
DLG_FLAG_* Options
EM_* Constants
EOL_* Values for EOL_FORMAT
EOL_SEQ
FALSE
FL_* Values for FL_STATUS
FONT_MENU_TAG
HSF_* Option Flags
INT_MAX
K_* Key Codes
LONG_MAX
M_* Values for DRAW_MODE
M_EDIT_*, M_FILE_*, M_HELP_* Menu Tags
M_FONT and M_STYLE
MAX_MENU_TAG
NO_STD_ABOUT_BOX
NO_STD_*_MENU Values
NULL
NULL_FNTID
NULL_IMAGE
NULL_PALETTE
NULL_PICTURE
NULL_PIXMAP
NULL_TXEDIT
NULL_WIN
P_* Values for PEN_STYLE
PAT_* Values for PAT_STYLE
RESP_* Values for ASK_RESPONSE
SC_* Values for SCROLL_CONTROL
SCREEN_WIN
*SCROLL Values for SCROLL_TYPE
SEV_* Values for XVT_ERRSEV
SHRT_MAX
Software Identifiers
SZ_CLASS_NAME
SZ_FNAME
SZ_LEAFNAME
TASK_WIN
TL_* Constants
TRUE
TX_* Attributes
U_* Values for UNIT_TYPE

```
UCHAR_MAX
UNIT_MAX
ULONG_MAX
USHRT_MAX
W_*, WC_*, WD_*, Values for WIN_TYPE
WSF_* Options Flags
XVT_CALLCONV*
XVT_CLUT_SIZE
XVT_COLOR_*
XVT_COLOR_GET_BLUE
XVT_COLOR_GET_GREEN
XVT_COLOR_GET_RED
XVT_CTOOLS_*
XVT_CXO_*_MSG
XVT_CXO_POS_* Values for XVT_CXO_INSERTION
XVT_DISPLAY_* Values
XVT_ESC_*
XVT_FA_* Constants
XVT_FAST_WIDTH
XVT_FFN_* Constants
XVT_FILE_ATTR_* Constants
XVT_FILESYS_* Values
XVT_FS_* Constants
XVT_HELP_* Values for XVT_HELP_FLAVOR
XVT_IMAGE_* Values for XVT_IMAGE_FORMAT
XVT_MAKE_COLOR
XVT_MAX_MB_SIZE
XVT_MAX_WINDOW_RECT
XVT_MOD_KEY
XVT_NAV_INSERTION
XVT_PALLETE_* Values
XVT_PALETTE_SIZE
XVT_PIXMAP_* Values
XVT_STRING_RES_BASE
XVT_TIMER_ERROR
XVT_TPC_* Constants
XVTWS, *WS Values
```

A_* Values for ACCESS_CMD

CMD Parameter to `xvt_tx_get_line`

Summary

```
typedef enum {
    A_LOCK,        /* lock text line */
    A_GET,         /* retrieve text pointer */
    A_UNLOCK     /* unlock text line */
} ACCESS_CMD;
```

Description

The use of these constants is described in `xvt_tx_get_line`.

See Also

`xvt_tx_get_line`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

CB_* Values for CB_FORMAT

Clipboard Format

Summary

```
typedef enum {/* std. clipboard format */
    CB_TEXT,/* ASCII text */
    CB_PICT,/* encapsulated picture */
    CB_APPL /* app's own (must have name) */

} CB_FORMAT;
```

Description

XVT supports two standard clipboard formats, text and encapsulated picture, plus whatever special formats an application uses for itself.

The enumeration constants `CB_TEXT`, `CB_PIC`, and `CB_APPL` identify the formats. When the format is `CB_APPL`, you must supply a name of four or fewer characters. The choice of a name is up to your application.

See Also

`xvt_cb_get_data`
`xvt_cb_free_data`
`xvt_cb_has_format`
`xvt_cb_put_data`

The "Clipboard" chapter in the *XVT Portability Toolkit Guide*

Example

This code fragment demonstrates the use of `CB_FORMAT`:

```
BOOLEAN paste_enabled = TRUE;
...
if (xvt_cb_has_format(CB_APPL, APPL_FORMAT))
    paste_fmt = CB_APPL;
else if (xvt_cb_has_format(CB_PICT, NULL))
    paste_fmt = CB_PICT;
else if (xvt_cb_has_format(CB_TEXT, NULL))
    paste_fmt = CB_TEXT;
else
    paste_enable = FALSE;
```

CHAR_MAX

Maximum `char` Value

Summary

```
#define CHAR_MAX ...
```

Description

This is XVT's implementation of the ANSI C symbol for the maximum value of a `char` variable.

See Also

```
INT_MAX
LONG_MAX
SHRT_MAX
UCHAR_MAX
UNIT_MAX
ULONG_MAX
USHRT_MAX
```

COLOR_*, COLOR_INVALID Constants

Predefined Colors

Summary

```
#define COLOR_INVALID ...
#define COLOR_BLACK ...
#define COLOR_BLUE ...
#define COLOR_CYAN ...
#define COLOR_DKGRAY ...
#define COLOR_GRAY ...
#define COLOR_GREEN ...
#define COLOR_LTGRAY ...
#define COLOR_MAGENTA ...
#define COLOR_RED ...
#define COLOR_WHITE ...
#define COLOR_YELLOW ...
```

Description

These constants provide a convenient way for you to refer to 11 commonly used colors. The last 5 listed are the most portable; they can be distinguished even on non-color monitors. The low 24 bits of these constants are the RGB encodings for the color. The high byte contains a value that is used internally by XVT.

You are not limited to these 11 colors if the underlying window system and your hardware support more. Use the `XVT_MAKE_COLOR` macro to make arbitrary colors from red, green, and blue values.

The `COLOR_INVALID` value is returned by some functions as an error indicator.

Implementation Note

Since XVT uses the high byte of the `COLOR` value, if you want to compare a `COLOR` to one of the standard `COLOR`s, you must compare only the low 24 bits. Mask off the high byte like this:

```
COLOR mycolor;if ((mycolor & 0x00FFFFFF) ==
    COLOR_BLUE & 0x00FFFFFF) {
    /* actual colors match */
}
```

See Also

COLOR
XVT_COLOR_GET_BLUE
XVT_COLOR_GET_GREEN
XVT_COLOR_GET_RED
XVT_MAKE_COLOR

The "Drawing and Pictures" chapter in the *XVT Portability Toolkit Guide*

CTL_FLAG_* Options

Control Flags

Summary

```
#define CTL_FLAG_CENTER_JUST ...
#define CTL_FLAG_CHECKED ...
#define CTL_FLAG_DEFAULT ...
#define CTL_FLAG_DISABLED ...
#define CTL_FLAG_GROUP ...
#define CTL_FLAG_INVISIBLE ...
#define CTL_FLAG_LEFT_JUST ...
#define CTL_FLAG_MULTIPLE ...
#define CTL_FLAG_NATIVE_JUST ...
#define CTL_FLAG_READONLY ...
#define CTL_FLAG_RIGHT_JUST ...
#define CTL_FLAG_MAC_GENEVA9 ...
#define CTL_FLAG_MAC_MONACO9 ...
#define CTL_FLAG_MAC_MULTILINE ...
#define CTL_FLAG_MAC_WORDWRAP ...
#define CTL_FLAG_PASSWORD ...
```

Description

`CTL_FLAG_*` attributes specify the special characteristics a control will have when it is created. Only `CTL_FLAG_DISABLED` and `CTL_FLAG_INVISIBLE` are generic to all controls. Other `CTL_FLAG_*` attributes apply only to a subset of XVT controls.

The attributes for a control are specified by ORing together the appropriate `CTL_FLAG_*` constants for the control into an "attribute value." When you call `xvt_ctl_create`, this value is passed via the `ctl_flags` parameter. If you call `xvt_ctl_create_def`, `xvt_win_create_def`, or `xvt_dlg_create_def`, this value is set in the `v.ctl.flags` field of the appropriate element of the `WIN_DEF` array.

`CTL_FLAG_CHECKED`

Used for check boxes and radio buttons only; it creates a control that is initially in the "checked" state. To toggle the "checked state" of a control, call `xvt_ctl_set_checked`.

`CTL_FLAG_DISABLED`

Creates a control that is initially disabled, meaning that it is not operable by the user. To enable the control, call `xvt_vobj_set_enabled`.

`CTL_FLAG_DEFAULT`

Used for push buttons only, and to create a push button with a default border style. It is only meaningful when creating a push button in a dialog whose ID is equal to `DLG_OK`.

`CTL_FLAG_GROUP`

Used for radio button controls only; it denotes the first or last element in a radio button grouping, for keyboard navigation purposes. For the purpose of keyboard navigation, a radio button group is treated as a single entity. Setting this flag does not affect how radio buttons are checked. Checking is determined by `xvt_ctl_check_radio_button`.

`CTL_FLAG_INVISIBLE`

Specifies that a control is to be created initially invisible. An invisible control cannot be seen or operated by the user. To make the control visible, call `xvt_vobj_set_visible`.

`CTL_FLAG_MULTIPLE`

Used for `WC_LBOX` controls only. It implies that the user can make multiple selections from a list box.

`CTL_FLAG_NATIVE_JUST`
`CTL_FLAG_CENTER_JUST`
`CTL_FLAG_LEFT_JUST`
`CTL_FLAG_RIGHT_JUST`

Specify special text justification (when possible for a particular toolkit) for `WC_PUSHBUTTON`, `WC_RADIOBUTTON`, `WC_CHECKBOX`, `WC_TEXT`, `WC_EDIT`, `WC_LISTEDIT`, `WC_LISTBUTTON`, and `WC_GROUPBOX` (text component only) controls.

`CTL_FLAG_READONLY`

Is valid only for creating controls of type `WC_LBOX`. If this flag is set, the list box will not allow selection but will allow the user to scroll to view the contents of the list box.

`CTL_FLAG_MAC_MULTILINE`

Enables multiline edit controls in XVT/Mac only. If this flag is set, new lines can be entered into edit control text.

`CTL_FLAG_MAC_WORDWRAP`

Enables multiline edit controls in XVT/Mac only. If this flag is set, text is autowrapped to the next line when it exceeds the edit control border.

`CTL_FLAG_MAC_GENEVA9`

Specifies the font as 9 point Geneva in any control containing text (XVT/Mac only).

`CTL_FLAG_MAC_MONACO9`

Specifies the font as 9 point Monaco in any control containing text (XVT/Mac only).

`CTL_FLAG_PASSWORD`

Is valid only for creating controls of type `WC_EDIT`. If this flag is set, the edit control hides the typed characters by always displaying a single character. The character displayed defaults to a character appropriate for the native platform look-and-feel. However, the displayed character can be set by defining a `STRING` resource with `resource id RID_PASSWORD_ECHO_CHAR_STR`. The first character of this `STRING` resource is used if it exists in resources but is not guaranteed to work with non-ASCII characters. This flag only affects the display of typed characters by displaying the echo character. The actual typed contents can still be retrieved with `xvt_vobj_get_title` and `xvt_vobj_set_title` works as expected.

For the final authoritative reference itemizing of valid `CTL_FLAG_*` combinations for each type of control, refer to topics listed in the See Also below.

Note: url uses a different method of specifying control characteristics. if you are creating controls by calling `xvt_win_create_res` or `xvt_dlg_create_res`, read the "Resources and Url" in the *`XVT Portability Toolkit Guide`*.

See Also

```
xvt_ctl_check_radio_button
xvt_ctl_create
xvt_ctl_create_def
xvt_ctl_set_checked
xvt_dlg_create_def
xvt_dlg_create_res
xvt_vobj_set_enabled
xvt_vobj_set_visible
xvt_win_create_def
xvt_win_create_res
URL Statements
Window/Dialog/Control Creation Function Parameters
```

The "Resources and URL" chapter in the *XVT Portability Toolkit Guide*
The *XVT Platform-Specific Books*

CURSOR_* Options

Cursor Shape

Summary

```
#define CURSOR_ARROW .../* arrow (default) */
#define CURSOR_CROSS .../* cross hair */
#define CURSOR_HELP .../* question mark */
#define CURSOR_IBEAM .../* I-beam */
#define CURSOR_PLUS .../* plus sign */
#define CURSOR_USER .../* user defined shape */
#define CURSOR_WAIT .../* waiting symbol */
```

Description

These values are used to specify a mouse cursor shape. Six shapes are predefined and can be used directly as arguments to `xvt_win_set_cursor`. Additionally, you can define your own cursor resources and use their IDs as arguments to `xvt_win_set_cursor`, provided their IDs are `CURSOR_USER` or greater.

The cursor is the pointer or other shape that indicates the current mouse position. Each XVT window has a current cursor that you can set to one of six standard shapes or to a shape that’s defined as a resource.

You need to set the cursor for a window just once--XVT automatically takes care of setting it to the designated shape as the cursor moves from window to window.

See Also

```
CURSOR
xvt_scr_hide_cursor
xvt_scr_set_busy_cursor
xvt_win_get_cursor
xvt_win_set_cursor
```

Example

```
xvt_win_set_cursor(win, CURSOR_IBEAM);
```

DEFAULT_*_MENU Values

Standard Menu URL Constants

Summary

```
#define DEFAULT_EDIT_MENU ...
#define DEFAULT_FILE_MENU ...
#define DEFAULT_FONT_MENU ...
#define DEFAULT_HELP_MENU ...
```

Description

You should define these macros in `MENU` statements of your URL file to automatically provide the predefined standard Edit, File, Font, and Help submenus. The predefined submenus are directly accessible only in URL.

Implementation Note

Font/Style submenus are available only on XVT/XM and XVT/Mac. On other platforms, no Font submenus exist. Instead, select the Font menu item to invoke a Font Selection dialog.

See Also

```
M_EDIT_*, M_FILE_*, M_HELP_* Menu Tags
M_FONT and M_STYLE
NO_STD_*_MENU Values
xvt_menu_*
```

Example

To add standard predefined File, Edit, and Font menus to your application, you can use code similar to the following in your URL file:

```
#define NO_STD_HELP_MENU/* prevents help menu from
     being built */
#include "url.h"
#include "sample.h"/* includes #define for
    SAMPLE_MENUBAR */

MENUBAR SAMPLE_MENUBAR

MENU SAMPLE_MENUBAR

DEFAULT_FILE_MENU
DEFAULT_EDIT_MENU
SUBMENU M_CHOICE "~Sample Choice"
DEFAULT_FONT_MENU
```

DIR_TYPE

File Type for Directories

Summary

```
#define DIR_TYPE .../* used with xvt_fsys_list_files */
```

Description

This constant is used only with the `xvt_fsys_list_files` function. Its usage is explained under that topic.

See Also

`xvt_fsys_list_files`

DLG_* Control IDs

Predefined Control Ids

Summary

```
#define DLG_CANCEL .../* cancel button clicked */
#define DLG_NO .../* other button clicked */
#define DLG_OK .../* default button clicked */
#define DLG_YES DLG_OK/* synonym */
```

Description

These are predefined button IDs common to many dialogs, and are provided only as a convenience. In general, your application can use

any control IDs for the controls in a dialog, with the following two exceptions:

- If you have a default push button, its ID should be `DLG_OK`
- If you have a cancel push button, its ID should be `DLG_CANCEL`

Implementation Note

On platforms that have "Close" menu item on a dialog, selecting this item either sends an `E_CONTROL` event for a button with an ID of `DLG_CANCEL`, if the button exists, or it simply sends an `E_CLOSE` event.

See Also

```
E_CONTROL
xvt_dlg_create_def
xvt_dlg_create_res
```

Example

```
long XVT_CALLCONV1 dlg_eh(WINDOW xdWindow,
    EVENT *xdEvent)
    {
        switch (xdEvent->type) {
        ...
        case E_CONTROL:
            switch(xdEvent->v.ctl.id) {
            case DLG_OK:
                ...do default behavior...
                break;
            case DLG_CANCEL:
                ...do_cancel_behavior...
                break;
            }
            break;

    ...
}
```

DLG_FLAG_* Options

Dialog Option Flags

Summary

```
#define DLG_FLAG_DISABLED ...
#define DLG_FLAG_INVISIBLE ...
```

Description

These flags are used only when calling `xvt_dlg_create_def`. They are set in the `win_def_p[0].v.dlg.flags` variable passed to `xvt_dlg_create_def`.

`DLG_FLAG_DISABLED`

Makes the dialog appear initially disabled. To enable the dialog, call `xvt_vobj_set_enabled`.

`DLG_FLAG_INVISIBLE`

Makes the dialog appear initially invisible. To make the dialog visible, call `xvt_vobj_set_visible`.

Note: Setting either of these flags for a modal dialog is not recommended and has an undefined effect.

See Also

`xvt_dlg_create_def`
`xvt_res_get_dlg_def`
`xvt_vobj_set_enabled`
`xvt_vobj_set_visible`

EM_* Constants

Event Mask Constants

Summary

```
#define EM_ALL ...
#define EM_CHAR ...
#define EM_CLOSE ...
#define EM_COMMAND ...
#define EM_CONTROL ...
#define EM_CREATE ...
#define EM_DESTROY ...
#define EM_FOCUS ...
#define EM_FONT ...
#define EM_HELP ...
#define EM_HSCROLL ...
#define EM_MOUSE_DBL ...
#define EM_MOUSE_DOWN ...
#define EM_MOUSE_MOVE ...
#define EM_MOUSE_SCROLL ...
#define EM_MOUSE_UP ...
#define EM_NONE ...
#define EM_QUIT ...
#define EM_SIZE ...
#define EM_TIMER ...
#define EM_UPDATE ...
#define EM_USER ...
#define EM_VSCROLL ...
```

Description

The event mask constants are used to restrict the events that can be sent to the event handler for a window or dialog. A bitwise-OR'd combination of the `EM_*` constants can be passed to `xvt_win_set_event_mask` and the window creation functions--`xvt_win_create_def`, `xvt_win_create_res`, and `xvt_win_create`. When calling these functions, you normally set the event mask to `EM_ALL` for no restriction. If you do not want any events sent to the event handler for the window, set the event mask to `EM_NONE`. `xvt_win_get_event_mask` returns the current mask setting.

See Also

```
xvt_dlg_create_def
xvt_dlg_create_res
xvt_win_create
xvt_win_create_def
xvt_win_create_res
xvt_win_get_event_mask
xvt_win_set_event_mask
```

The "Events" chapter in the *XVT Portability Toolkit Guide*

EOL_* VALUES for EOL_FORMAT

Terminator Found by xvt_str_find_eol

Summary

```
typedef enum e_eol {/* xvt_str_find_eol terminator */
    EOL_NORMAL,/* normal (or first) terminator */
    EOL_DIFF,/* terminator different from prev. */
    EOL_NONE/* end of buf before any terminator */

} EOL_FORMAT;
```

Description

This type is used for the return value from the `xvt_str_find_eol` function to indicate the reason for its return. For further information, see the topic `xvt_str_find_eol`.

See Also

`EOL_SEQ`
`xvt_str_find_eol`

EOL_SEQ

Local End-of-Line Sequence Constant

Summary

```
#define EOL_SEQ .../* local eol sequence */
```

Description

This string constant contains the standard end-of-line sequence for the local operating system. Depending on the system, the end-of-line sequence can be a return, a line feed, or return and a line feed. The type of `EOL_SEQ` is `char *`.

When formatting lines of text for the clipboard, you should add `EOL_SEQ` strings to the end of each line. If you are writing a file of text opened in text mode, you don’t have to add `EOL_SEQ` strings to the end of each line because the standard C library will add the local end-of-line sequence.

When breaking into lines a collection of text lines that are already in memory, you can scan for occurrences of the `EOL_SEQ` string, but it's much easier to call the function `xvt_str_find_eol`.

Implementation Note

The definition of `EOL_SEQ` is platform-specific. If you require the exact values of this macro, refer to the definition in the platform-specific file **xvt_plat.h** (or **xvt_plxs.h** on XVT/XM).

See Also

```
EOL_* Values for EOL_FORMAT
xvt_str_find_eol
```

xvt_plat.h (or **xvt_plxs.h** on XVT/XM)

Example

```
char buffer[1000];
buffer[0] = '0';
for (i = 0; text[i] != NULL; i++) {
    strcat(buffer, text[i]);
    strcat(buffer, EOL_SEQ);
}
```

FALSE

False Value

Summary

```
#define FALSE ...
```

Description

This symbol should be used for a false `BOOLEAN` value.

See Also

```
BOOLEAN
TRUE
```

Example

```
BOOLEAN flag = FALSE;
```

FL_* Values for FL_STATUS

File Dialog Result

Summary

```
typedef enum {/* file dialog result */
    FL_BAD, /* error occurred */
    FL_CANCEL,/* cancel button clicked */
    FL_OK/* OK button clicked */

} FL_STATUS;
```

Description

A structure of this type is returned by `xvt_dm_post_file_open` or `xvt_dm_post_file_save` to indicate the result of the user's interaction with the dialog box.

See Also

`xvt_dm_post_file_open`
`xvt_dm_post_file_save`

Example

See the examples for `xvt_dm_post_file_open` and `xvt_dm_post_file_save`.

FONT_MENU_TAG

Identifier for Entire Font/Style Menu

Summary

`FONT_MENU_TAG`

Description

`FONT_MENU_TAG` is a predefined constant that represents a combination of the Font and Style menus. The only use for this value is that you can call `xvt_menu_set_item_enabled` and pass the `FONT_MENU_TAG` as the `tag` parameter to enable or disable the Font/Style menus as a whole if they exist. This prevents your application from needing to know whether a Font menu or Style menu or both exist on the platform.

See Also

xvt_menu_set_item_enabled

HSF_* Option Flags

Help System Flags

Summary

```
#define HSF_APPNAME_TITLE ...
#define HSF_INDEX_ON_DISK ...
#define HSF_NO_BEEP_MODAL ...
#define HSF_NO_HELPMENU_ASSOC ...
#define HSF_NO_TOPIC_WARNING ...
```

Description

These flags are used when calling the `xvt_help_open_helpfile` function. They affect the behavior of the portable viewer in the help system.The `flags` information can contain zero or more of the following values, OR'd together:

`HSF_APPNAME_TITLE`

Normally, the help system displays the current topic in a help topic window title bar. If the `HSF_APPNAME_TITLE` flag option is set, your application name, as defined in the `XVT_CONFIG` structure, is used instead.

`HSF_INDEX_ON_DISK`

If this flag option is used, the topic index is maintained on disk. By default, the index is maintained in application memory. This option is useful for low-memory environments.

`HSF_NO_BEEP_MODAL`

Most native window systems do not allow the user to manipulate menus and windows when a modal dialog is active. Therefore, the user could not operate the help viewer if help was requested for a modal dialog. If `HSF_NO_BEEP_MODAL` is set, the system beeps if help is requested for a modal dialog. If this flag is not set, requests for help on modal dialogs are silently ignored.

`HSF_NO_HELPMENU_ASSOC`

Normally, the help system automatically associates help topics with the help menu items (e.g. "Help On Help", "Search"). If the

HSF_NO_HELPMENU_ASSOC flag is set, this association is not performed.

HSF_NO_TOPIC_WARNING

If this flag option is used, the help system does not display any error messages when a requested topic is not found in the help file. If this flag is not set, a "topic not found" message is displayed if a topic cannot be found.

Note: The HSF_NO_BEEP_MODAL flag only applies to the application-bound help viewer. It has no effect on native and standalone help viewers.

See Also

XVT_CONFIG
xvt_help_open_helpfile

INT_MAX

Maximum int Value

Summary

```
#define INT_MAX ...
```

Description

This is XVT's implementation of the ANSI C symbol for the maximum value of an int variable.

See Also

CHAR_MAX
LONG_MAX
SHRT_MAX
UCHAR_MAX
UNIT_MAX
ULONG_MAX
USHRT_MAX

K_* Key Codes

Virtual Key Codes

Summary

```
#define K_F1 .../* function key 1 */
#define K_F2 ...
#define K_F3 ...
#define K_F4 ...
#define K_F5 ...
#define K_F6 ...
#define K_F7 ...
#define K_F8 ...
#define K_F9 ...
#define K_F10 ...
#define K_F11 ...
#define K_F12 ...
#define K_F13 ...
#define K_F14 ...
#define K_F15 ...
#define K_F16 ...
#define K_F17 ...
#define K_F18 ...
#define K_F19 ...
#define K_F20 ...
#define K_F21 ...
#define K_F22 ...
#define K_F23 ...
#define K_F24 .../* function key 24 */

#define K_KP0 .../* keypad '0' */
#define K_KP1 ...
#define K_KP2 ...
#define K_KP3 ...
#define K_KP4 ...
#define K_KP5 ...
#define K_KP6 ...
#define K_KP7 ...
#define K_KP8 ...
#define K_KP9 .../* keypad '9' */

#define K_KPMULT .../* keypad '*' */
#define K_KPSUB .../* keypad '-' */
#define K_KPADD .../* keypad '+' */
#define K_KPDIV .../* keypad '/' */
```

```
#define K_COPY .../* copy */
#define K_CUT .../* cut */
#define K_PASTE .../* paste */
#define K_UP .../* up arrow */
#define K_DOWN .../* down arrow */
#define K_RIGHT .../* right arrow */
#define K_LEFT .../* left arrow */
#define K_PREV .../* previous screen */
#define K_NEXT .../* next screen */
#define K_LHOME .../* line home */
#define K_LEND .../* line end */
#define K_HOME .../* home */
#define K_END .../* end */
#define K_INS .../* insert */
#define K_WLEFT .../* word left */
#define K_WRIGHT .../* word right */
#define K_BTAB .../* back tab */
#define K_HELP .../* help */
#define K_CLEAR .../* clear */
#define K_DEL .../* del (same as ASCII) */
```

Description

These symbols represent XVT virtual key codes. The comments above indicate the proposed usage for each code, but the actual usage is up to your application.

XVT's virtual key codes map into various non-portable key combinations on the native systems. An application using the virtual key codes can rely on a sensible mapping to the native key codes, where possible.

You get one of these codes via an `E_CHAR event`. In the `EVENT` structure, any values of the `v.chr.ch` field that are greater than 255 (except for `K_DEL`) are virtual key codes.

Implementation Note

A mapping of native keyboard codes to XVT key codes is platform-specific. Not all key codes listed here can be generated by a given platform. An application can supplement the standard key translation by setting its own key hook function via `ATTR_KEY_HOOK`. See the *XVT Platform-Specific Books* for details.

See Also

```
ATTR_KEY_HOOK
E_CHAR
EVENT
```

The *XVT Platform-Specific Books*

LONG_MAX

Maximum `long` Value

Summary

```
#define LONG_MAX ...
```

Description

This is XVT's implementation of the ANSI C symbol for the maximum value of a `long` variable.

See Also

```
CHAR_MAX
INT_MAX
SHRT_MAX
UCHAR_MAX
UNIT_MAX
ULONG_MAX
USHRT_MAX
```

M_* Values for DRAW_MODE

Drawing Mode

Summary

```
typedef enum {/* drawing (transfer) mode *
    M_COPY,
    M_OR,
    M_XOR,
    M_CLEAR,
    M_NOT_COPY,
    M_NOT_OR,
    M_NOT_XOR,
    M_NOT_CLEAR

} DRAW_MODE;
```

Description

These symbols represent the eight drawing modes that determine how `CPEN`, `CBRUSH`, and icon pixels (called `source` pixels) affect pixels already present in a window when shapes are drawn. The most common modes are `M_COPY` and `M_XOR`.

`M_COPY`

Sets drawn pixels to match the source regardless of what is already in the window. The `M_COPY` mode uses the specified color reliably. `M_COPY` is XVT's normal mode.

`M_XOR`

Used for rubber banding. It can't be used for printing. `M_XOR` has the following two properties: drawing the same thing twice with this mode reproduces the original, and if a reasonable color map is present, the effects of `M_XOR` drawing are visible. It is possible, however, that the effect of an `M_XOR` draw operation on a color display might not result in a color that contrasts with the background.

To set the mode for a window, use `xvt_dwin_set_draw_mode` or `xvt_dwin_set_draw_ctools`.

The following chart shows how these modes act on a pixel in a window when the source pixel is black or white:

M_COPY	
Black:	Force to black
White:	Force to white
M_OR	
Black:	Force to black
White:	Leave as is
M_XOR	
Black:	Invert
White:	Leave as is
M_CLEAR	
Black:	Force to white
White:	Leave as is
M_NOT_COPY	
Black:	Force to white
White:	Force to black
M_NOT_OR	
Black:	Leave as is
White:	Force to black
M_NOT_XOR	
Black:	Leave as is
White:	Invert
M_NOT_CLEAR	
Black:	Leave as is
White:	Force to white

Implementation Note

The following chart gives examples of the XVT modes in terms of some of the native toolkit modes. In order to make sense of the MS-Windows modes, it's necessary to know that on MS-Windows a

white pixel is represented by a 1 bit, and a black pixel by a 0 bit, since white is "all colors" and black is "no colors."

M_COPY	
Mac Mode:	`patCopy`
Windows Mode:	`R2_COPYPEN`
Motif:	`GXcopy`
M_OR	
Mac Mode:	`patOr`
Windows Mode:	`R2_MASKPEN`
Motif:	`GXand`
M_XOR	
Mac Mode:	`PatXor`
Windows Mode:	`R2_NOTXORPEN`
Motif:	`GXxor`
M_CLEAR	
Mac Mode:	`patBic`
Windows Mode:	`R2_MERGENOTPEN`
Motif:	`GXor`
M_NOT_COPY	
Mac Mode:	`notPatCopy`
Windows Mode:	`R2_NOTCOPYPEN`
Motif:	`GXcopyInverted`
M_NOT_OR	
Mac Mode:	`notPatOr`
Windows Mode:	`R2_MASKNOTPEN`
Motif:	`GXandInverted`
M_NOT_XOR	
Mac Mode:	`notPatXor`
Windows Mode:	`R2_XORPEN`
Motif:	`GXequiv`
M_NOT_CLEAR	
Mac Mode:	`notPatBic`
Windows Mode:	`R2_MERGEPEN`
Motif:	`CXorInverted`

See Also

`xvt_dwin_set_draw_ctools`
`xvt_dwin_set_draw_mode`

Example

This code uses the `M_XOR` drawing mode to draw a shape that can track the mouse without disturbing what's already showing in the window. If `mark_point` is called *twice* with the same argument, the pixels that were marked are restored exactly to the way they were.

```
static void mark_point(win, p)WINDOW win;
PNT p;
{
    RCT rct;
    CBRUSH brush;xvt_dwin_set_std_cpen(win,
        TL_PEN_BLACK);
    brush.pat = PAT_SOLID;
    brush.color = COLOR_LTGRAY;
    xvt_dwin_set_cbrush(win, &brush);
    xvt_dwin_set_draw_mode(win, M_XOR);
    xvt_rect_set(&rct, p.h - 10,
        p.v - 10, p.h + 10, p.v + 10);
    xvt_dwin_draw_rect(win, &rct);
}
```

M_EDIT_*, M_FILE_*, M_HELP_* Menu Tags

Predefined Menu Tags

Summary

```
#define M_FILE ...
#define M_FILE_NEW ...
#define M_FILE_OPEN ...
#define M_FILE_CLOSE ...
#define M_FILE_SAVE ...
#define M_FILE_SAVE_AS ...
#define M_FILE_REVERT ...
#define M_FILE_PG_SETUP ...
#define M_FILE_PRINT ...
#define M_FILE_QUIT ...
#define M_FILE_ABOUT ...
#define M_EDIT ...
#define M_EDIT_UNDO ...
#define M_EDIT_CUT ...
#define M_EDIT_COPY ...
#define M_EDIT_PASTE ...
#define M_EDIT_CLEAR ...
#define M_EDIT_SEL_ALL ...
#define M_EDIT_CLIPBOARD ...
#define M_HELP ...
#define M_HELP_HELPMENU ...
#define M_HELP_ONCONTEXT ...
#define M_HELP_HELPONHELP ...
#define M_HELP_ONWINDOW ...
#define M_HELP_KEYBOARD ...
#define M_HELP_INDEX ...
#define M_HELP_TUTORIAL ...
#define M_HELP_SEARCH ...
#define M_HELP_OBJCLICK ...
#define M_HELP_VERSION ...
#define M_HELP_GOTO ...
#define M_HELP_GLOSSARY ...
#define M_HELP_CONTENTS ...
```

Description

These constants are used in XVT's predefined File, Edit, and Help submenus, and represent standard menu operations that are common to many GUI environments. Most of your window menubars contain File, Edit, and Help submenus, except for the task window's menubar, which normally has no use for an Edit menu.

XVT supplies pre-fabricated File, Edit, and Help submenus in the **url_plat.h** header for your platform. Those submenus contain items whose tags are from the list constants above. On a given platform,

the predefined submenus contain a subset of these tags as appropriate for the platform.

The predefined submenus are directly accessible only in URL. When building a menubar, use the macros DEFAULT_FILE_MENU, DEFAULT_EDIT_MENU, and DEFAULT_HELP_MENU in the proper positions (normally first and last) of a MENU statement in your URL file. If you do not want any portion of these menus incuded on the menubar, define the macros NO_STD_FILE_MENU, NO_STD_EDIT_MENU, or NO_STD_HELP_MENU prior to including **url.h** in your URL file.

All of the items in the File and Edit submenus, except M_FILE_QUIT and M_FILE_ABOUT, are initially disabled. If your application wants the user to be able to select any other items, then you must enable them with xvt_menu_set_item_enabled. The items in the Help menu are initially enabled.

When the user chooses an item from one of the predefined submenus, your window's event handler then receives an E_COMMAND event. However, the help system may intercept E_COMMAND events from the Help menu.

Implementation Note

Since the content of the submenus varies between platforms, you should code your application to handle all possible commands, even though some might not occur on a given platform. For example, because M_EDIT_SEL_ALL only appears on XVT/Mac, you are able to test the code for handing that command only on XVT/Mac.

Since the items appearing on the standard File, Edit, and Help submenus are platform-specific, not all of the M_FILE_* and M_EDIT_* tags listed above exist on the predefined submenus for a given platform. Because of this, the XVT menu item manipulation functions ignore operations on non-existent predefined tags. For example, the standard XVT/Win32 Edit submenu does not have the item M_EDIT_SEL_ALL. Because of that, XVT ignores the following call under XVT/Win32, instead of issuing an error:

```
xvt_menu_set_item_enabled(win, M_EDIT_SEL_ALL);
```

See Also

DEFAULT_*_MENU Values
M_FONT and M_STYLE
MENU_ITEM
NO_STD_*_MENU Values
xvt_menu_*
menu and menubar URL Statement

The "Resources and URL" and the "Menus" chapters in the *XVT Portability Toolkit Guide*

M_FONT and M_STYLE

Predefined Menu Tags

Summary

```
#define M_FONT ...
#define M_STYLE ...
```

Description

These constants are the tags for the Font and Style submenus.

Typically, these tags are useful for one thing. When your application calls `xvt_menu_get_tree` or `xvt_res_get_menu`, it is handed a tree of `MENU_ITEM` structures representing the menubar and all of its submenus. When processing this tree, your application needs to be able to identify the Font and Style submenus, so that it can safely ignore them, since they are inherently non-portable. In general, applications should ignore any tags greater than `MAX_MENU_TAG`, except those that it has explicitly defined.

See Also

`DEFAULT_*_MENU Values`
`MAX_MENU_TAG`
`M_EDIT_*, M_FILE_*, M_HELP_* Menu Tags`
`MENU_ITEM`
`NO_STD_*_MENU Values`
`xvt_menu_*`
`xvt_res_get_menu`

MAX_MENU_TAG

Upper Bound of Application Menu Tag Values

Summary

```
#define MAX_MENU_TAG ...
```

Description

MAX_MENU_TAG defines the upper bound of menu tag values that your application is allowed to define.

Your application is allowed to define menu tags in the range (1 . . . MAX_MENU_TAG) for its own menu items. In contrast, XVT's predefined menus (File, Edit, Font, Style, Help), define tags in the range (MAX_MENU_TAG+1 . . . 32767). Your application is allowed to use either its own tags or XVT's tags when calling functions like xvt_menu_set_item_enabled or xvt_menu_set_item_title.

The main reason for this distinction is that menu-manipulating functions will ignore attempts to change non-existent menu items in XVT's reserved range, but will issue an error for attempts to change non-existent menu tags in the range reserved for your application.

See Also

MENU_ITEM
xvt_menu_set_item_enabled
xvt_menu_set_item_title

NO_STD_ABOUT_BOX

Standard About Box Removal Constant for URL

Summary

```
#define NO_STD_ABOUT_BOX ...
```

Description

You should define this macro in your URL file prior to including **url.h** if you do not want the resources for the standard About box to be built. This allows your application to define the DB_ABOUT resource itself instead of relying on the system-defined DB_ABOUT.

Example

To remove the standard About box from your application, you can use code like this in your URL file:

```
#define NO_STD_ABOUT_BOX
#include "url.h"
...
```

NO_STD_*_MENU Values

Standard Menu Removal Constants for URL

Summary

```
#define NO_STD_EDIT_MENU ...
#define NO_STD_FILE_MENU ...
#define NO_STD_FONT_MENU ...
#define NO_STD_HELP_MENU ...
```

Description

You should define these macros in your URL file prior to including `url.h`, if you do not want the resources for the respective portion of the menubar to be built.

Note: If you do not define these macros, the predefined menus are not automatically included in your menubar. To include the standard menus, you must use the `DEFAULT_*_MENU` values in an URL statement.

See Also

```
DEFAULT_*_MENU Values
M_EDIT_*, M_FILE_*, M_HELP_* Menu Tags
M_FONT and M_STYLE
```

Example

To remove the Help menubar item and its submenu tree from your application, you can use code like this in your URL file:

```
#define NO_STD_HELP_MENU
#include "url.h"
...
```

NULL Constants

```
NULL
NULL_FNTID
NULL_IMAGE
NULL_PALETTE
NULL_PICTURE
NULL_PIXMAP
NULL_TXEDIT
NULL_WIN
```

NULL

`NULL` Value Macro

Summary

```
#define NULL ...
```

Description

Your application can use `NULL` anywhere it would normally need to use a value of zero, such as in testing or initializing pointers, or passing a `NULL` string parameter. Many XVT functions return `NULL` if unsuccessful.

See Also

Other `NULL Constants`

Example

```
xvt_font_get_metrics(font_id, NULL, &ascent, NULL);
```

NULL_FNTID

`NULL` Font ID Macro

Summary

```
#define NULL_FNTID ...
```

Description

This is a macro provided to help in identifying a `NULL XVT_FNTID`. It might be returned as an error by functions that return `XVT_FNTID`. XVT provides this macro so that you can avoid having to cast `NULL` to perform an error check.

See Also

`XVT_FNTID`

NULL_IMAGE

`NULL` Image Macro

Summary

```
#define NULL_IMAGE ...
```

Description

This constant is used as an error return by functions that return an `XVT_IMAGE` (such as `xvt_image_create`). It avoids having to cast `NULL` to perform an error check. You can use it as the value of an `IMAGE` parameter in some functions, depending on the specific requirements of those functions.

See Also

Other `NULL Constants`
`XVT_IMAGE`
`xvt_image_create`

Example

```
if ((new_image = xvt_image_read(file_spec->name)) !=
    NULL_IMAGE)
    ncolors = xvt_image_get_ncolors(new_image);
```

NULL_PALETTE

`NULL` Palette Macro

Summary

```
#define NULL_PALETTE ...
```

Description

This constant is used as an error return by functions that return an `XVT_PALETTE` (such as `xvt_palet_create`, `xvt_palet_default`, `xvt_vobj_get_palet`). It avoids having to cast `NULL` to perform an error check.

See Also

Other `NULL Constants`
`XVT_PALETTE`
`xvt_palet_create`

```
xvt_palet_default
xvt_vobj_get_palet
```

Example

```
if ((palette = xvt_palete_create(pixmap)) ==
    NULL_PALETTE)
    xvt_dm_post_error(
        "Error retrieving palette from pixmap.");
```

NULL_PICTURE

`NULL` Picture Macro

Summary

```
#define NULL_PICTURE ...
```

Description

This constant is used as an error return by functions that return a `PICTURE` (such as `xvt_pict_create`). It avoids having to cast `NULL` to perform an error check. You can use it as the value of a `PICTURE` parameter in some functions, depending on the specific requirements of those functions.

See Also

Other `NULL Constants`
`PICTURE`
`xvt_pict_create`

Example

```
if (!xvt_cb_put_data(CB_TEXT, NULL, size, NULL_PICTURE))
xvt_dm_post_error("Error putting text onto clipboard.");
```

NULL_PIXMAP

`NULL` Pixmap Macro

Summary

```
#define NULL_PIXMAP ...
```

Description

This constant is used as an error return by functions that return an XVT_PIXMAP (such as xvt_pmap_create). It avoids having to cast NULL to perform an error check. NULL_PIXMAP is often used in tests involving XVT_PIXMAP variables.

See Also

Other NULL Constants
XVT_PIXMAP
xvt_pmap_create

Example

```
PIXMAP pixmap;
         ....
    xvt_errmsg_sig_if(pixmap == NULL_PIXMAP,
        NULL_WIN, SEV_FATAL, ERR_FAIL_CREATE_PIXMAP,
        TXT_ERR_FAIL_CREATE_PIXMAP, NULL, NULL );
```

NULL_TXEDIT

NULL Text Edit Object

Summary

```
#define NULL_TXEDIT ...
```

Description

This constant indicates a non-existent text edit object.

See Also

xvt_tx_create
xvt_tx_create_def

NULL_WIN

NULL Appropriate for Window Checks

Summary

```
#define NULL_WIN ...
```

Description

This is a macro provided to help identify a `NULL WINDOW`. It might be returned as an error by functions that return a `WINDOW`. XVT provides this macro so that you can avoid having to cast `NULL` to perform an error check.

See Also

Other `NULL Constants`
`WINDOW`
`xvt_win_create`

P_* Values for PEN_STYLE

Pen Style

Summary

```
typedef enum e_pen_style {/* pen style */
    P_SOLID,                /* solid */
    P_DOT,                  /* dotted line */
    P_DASH                  f/* dashed line */

} PEN_STYLE;
```

Description

Values of this type are used for the `style` member of a `CPEN` structure.

Implementation Note

On XVT/PM, lines thicker than one pixel are not drawn as truly dotted or dashed. Instead, a dithered pattern is used for thick pens, which at least allows them to be distinguished from other pen styles.

On XVT/Win32 and XVT/Mac, dotted and dashed lines are always displayed as one pixel wide, regardless of width.

See Also

`CPEN`

The "Pens" section of the "Drawing and Pictures" chapter in the *XVT Portability Toolkit Guide*

PAT_* Values for PAT_STYLE

Pattern Style

Summary

```
typedef enum {
    PAT_NONE,       /* no pattern */
    PAT_HOLLOW,     /* hollow */
    PAT_SOLID,      /* solid fill */
    PAT_HORZ,       /* horizontal lines */
    PAT_VERT,       /* vertical lines */
    PAT_FDIAG,      /* diagonal lines -- top-left to
                       bottom-right */
    PAT_BDIAG,      /* diagonal lines -- top-right to
                       bottom-left */
    PAT_CROSS,      /* horizontal and vertical crossing
                       lines */
    PAT_DIAGCROSS, /* diagonol crossing lines */
    PAT_RUBBER,     /* rubber banding */
    PAT_SPECIAL
} PAT_STYLE;
```

Description

Values of this type are used for the `pat` member of the `CBRUSH` and `CPEN` structures. For pens, the only patterns allowed are `PAT_HOLLOW`, `PAT_SOLID`, and `PAT_RUBBER`. For brushes, all the patterns are allowed, except for `PAT_RUBBER`.

See Also

`CBRUSH`
`CPEN`

Section "Pens" and section "Brushes and Background Colors" of the "Drawing and Pictures" chapter in the *XVT Portability Toolkit Guide*

RESP_* Values for ASK_RESPONSE

Response from `xvt_dm_post_ask`

Summary

```
typedef enum {     /* response from xvt_dm_post_ask fcn */
    RESP_DEFAULT,  /* default button */
    RESP_2,         /* second button */
    RESP_3          /* third button */

} ASK_RESPONSE;
```

Description

`xvt_dm_post_ask` returns a value of `ASK_RESPONSE`. The value returned depends on the button selected by the user.

See Also

`xvt_dm_post_ask`

SC_* Values for SCROLL_CONTROL

Scrollbar Component

Summary

```
typedef enum {,    /* scrollbar activity */
    SC_NONE,       /* nowhere (ignore) */
    SC_LINE_UP,    /* one line up */
    SC_LINE_DOWN,  /* one line down */
    SC_PAGE_UP,    /* previous page */
    SC_PAGE_DOWN,  /* next page */
    SC_THUMB,      /* thumb repositioning */
    SC_THUMBTRACK  /* thumb tracking */

} SCROLL_CONTROL;
```

Description

Values of this type refer to parts of a scrollbar that the user can manipulate. They are used to identify the part that the user operated to generate an `E_HSCROLL` or `E_VSCROLL event`, if the scrollbar is on the frame of a window, or an `E_CONTROL` event if the scrollbar is a control.

See Also

E_CONTROL
E_HSCROLL
E_VSCROLL

The "Window Scrollbars and Scrolling" section of the "Windows" chapter in the *XVT Portability Toolkit Guide*

SCREEN_WIN

Application Container Window

Summary

```
#define SCREEN_WIN ...
```

Description

SCREEN_WIN is an abstract XVT WINDOW representing the screen. Its only purpose is to be used as a container for top-level windows, dialogs, and TASK_WINs. Your application can pass SCREEN_WIN as the parent argument to one of the xvt_win_create_* functions. All dialogs automatically get SCREEN_WIN as their parent.

SCREEN_WIN is one of the two containers available for top-level windows. The other container is TASK_WIN. The normal case is for applications to use TASK_WIN as the parent for top-level windows.

Besides using it as a container, you can do two other things with SCREEN_WIN: get its client area (useful for placing windows exactly within the screen container), and set and get its application data.

Implementation Note

On XVT/Win32 using SCREEN_WIN instead of TASK_WIN has the effect of detaching them from the task container window and making them appear on the task manager list. On other platforms, this difference has no visual appearance.

See Also

ATTR_SCREEN_WINDOW
TASK_WIN
WINDOW
xvt_win_create
xvt_win_create_def
xvt_win_create_res

*SCROLL Values for SCROLL_TYPE

Type of Scrollbar

Summary

```
typedef enum {   /* type of scrollbar */
    HSCROLL,     /* horizontal */
    VSCROLL,     /* vertical */
    HVSCROLL,    /* either */

} SCROLL_TYPE;
```

Description

Your application needs to use values of this type to specify a scrollbar orientation to any of the scrollbar-manipulation functions, such as `xvt_sbar_set_*` and `xvt_sbar_get_*`. To specify a scrollbar on a window's frame, pass `HSCROLL` or `VSCROLL` to these functions (`HSCROLL` speicifies a horizontal scrollbar, `VSCROLL` specifies a vertical scrollbar). To specify a scrollbar control in a window or dialog, you pass `HVSCROLL` to these functions for either a horizontal or vertical scrollbar.

See Also

```
xvt_sbar_get_pos
xvt_sbar_get_proportion
xvt_sbar_get_range
xvt_sbar_set_pos
xvt_sbar_set_proportion
xvt_sbar_set_range
```

The "`E_HSCROLL` and `E_VSCROLL` Events" sections of the "Events" chapter in the *XVT Portability Toolkit Guide*

SEV_* Values for XVT_ERRSEV

Error Severity Type

Summary

```
typedef enum {
    SEV_NONE,
    SEV_WARNING
    SEV_ERROR,
    SEV_FATAL,
} XVT_ERRSEV;
```

Description

This enumeration defines error severity codes used in error signaling.

See Also

`xvt_errmsg_sig`
`xvt_errmsg_sig_std`

The "Diagnostics and Debugging" chapter in the *XVT Portability Toolkit Guide*

SHRT_MAX

Maximum Short Value

Summary

```
#define SHRT_MAX ...
```

Description

This is XVT's implementation of the ANSI C symbol for the maximum value of a `short` variable.

See Also

`CHAR_MAX`
`INT_MAX`
`LONG_MAX`
`UCHAR_MAX`
`UNIT_MAX`
`ULONG_MAX`
`USHRT_MAX`

Software Identifiers

Software Version Identifiers

The following constants define the version number for the XVT Help system:

```
XVT_HELP_VERSION_MAJOR
XVT_HELP_VERSION_MINOR
XVT_HELP_VERSION_PATCH
```

The following constants define the version number for the XVT Portability Toolkit:

```
XVT_PTK_VERSION_MAJOR
XVT_PTK_VERSION_MINOR
XVT_PTK_VERSION_PATCH
```

The folowing constants define the version number for the Text Edit Object:

```
XVT_TX_VERSION_MAJOR
XVT_TX_VERSION_MINOR
XVT_TX_VERSION_PATCH
```

SZ_CLASS_NAME

Maximum Length of a Class Name

Summary

```
#define SZ_CLASS_NAME ...
```

Description

This constant specifies the maximum length of a class name.

SZ_FNAME

Maximum Size of Filename

Summary

```
#define SZ_FNAME .../* max filename length */
```

Description

This constant specifies the maximum filename length for the local system, and is used in the definition of `FILE_SPEC`.

See Also

```
FILE_SPEC
SZ_LEAFNAME
```

SZ_LEAFNAME

Maximum Size of Directory or Filename

Summary

```
#define SZ_LEAFNAME .../*max directory or filename
                        length*/
```

Description

This constant adds convenience for using `xvt_fsys_parse_pathname`. This constant is defined as the maximum length (in bytes) of a single directory or filename in a pathname for the local system. This length includes file extensions and the period (’.’).

See Also

```
FILE_SPEC
SZ_FNAME
xvt_fsys_parse_pathname
```

Example

```
char fname[SZ_LEAFNAME + 1];
xvt_fsys_parse_pathname(fullpath, NULL,
     NULL, fname, NULL, NULL);
```

TASK_WIN

Task Container Window

Summary

```
#define TASK_WIN ...
```

Description

The task window represented by the TASK_WIN macro serves two purposes in an XVT application. First, it acts as a central WINDOW object representing the application as a whole. Second, it is a container window that can be used as the parent for other windows.

TASK_WIN has an event handler to receive events that affect your application as a whole. It is this event handling function that you pass in your initial call to xvt_app_create. This task event handler receives the following events portably across all systems:

- An E_CREATE event signalling the start of an application.
- An E_DESTROY event when the application is terminated.
- An E_COMMAND event when the user operates the task menubar. The task menubar is available for the user to operate under different situations across platforms, but in general the user is able to operate the task menubar if no other window's menubar is available. Therefore, you should think of the task menubar as a "backup" menubar for the user to operate when there are no other menubars around.
- An E_FONT event if the user chooses a font from the font selection menu or dialog. This is possible only if the task menubar has Font/Style menus on it, or if a Font Selection dialog is requested.
- An E_TIMER event if your application calls xvt_timer_create and uses TASK_WIN as an argument.
- An E_USER event if your application sends them via xvt_win_dispatch_event.
- An E_CLOSE event on XVT/Win32 and XVT/XM, indicating that the user wishes to end the program.
- An E_QUIT event on XVT/MAC and XVT/Win32 indicating that the user has initiated a system shutdown.
- An E_SIZE event following its initial E_CREATE event to tell the application how large the task window's client area is

(and hence how much area is available for placing top-level windows within the task window). On XVT/Win32, it will also receive an E_SIZE event when the user resizes the physical task window. On other platforms, the size will never change because TASK_WIN maps onto the screen.

Although several events (i.e., E_QUIT, E_CLOSE, and E_SIZE) are delivered differently on some platforms, your application must still *handle* the events with portable code. If your development platform does not deliver E_QUIT events, your application must handle E_QUIT events if you want portable code. In addition, an E_CLOSE event sent to the task window should be handled in the same way it would be if the user chose Quit from the File menu. Your application can safely ignore E_SIZE events sent to the task handler, unless it wants to adjust the layout of windows contained in the task window when the size of the task window changes.

Since the task window is the XVT representation of an application, calling xvt_vobj_destroy(TASK_WIN) will terminate the application.

Implementation Note

In addition to the events received on all platforms, the following events can only be received on XVT/Win32, when you set the ATTR_WIN_PM_DRAWABLE_TWIN attribute before the calling xvt_app_create: E_CHAR, E_CONTROL, E_FOCUS, E_HSCROLL, E_MOUSE_*, E_UPDATE, and E_VSCROLL.

On XVT/Win32 a task window can also be used as a container window. On these platforms, the task window is a visible object that the user can manipulate. Specifying TASK_WIN as the parent argument to an xvt_*_create_* function causes a top-level window to be created inside the physical task window container. In contrast, using SCREEN_WIN as the parent causes a top-level window to be created outside of the physical task window (on "the screen"). On other platforms, TASK_WIN and SCREEN_WIN represent the same container, namely the screen itself.

See Also

XVT Events
ATTR_TASK_WINDOW
SCREEN_WIN
WINDOW
xvt_app_create

TL_* Constants

Standard Tool Constants

Summary

```
#define TL_PEN_BLACK ...
#define TL_PEN_HOLLOW ...
#define TL_PEN_RUBBER ...
#define TL_PEN_WHITE ...
#define TL_BRUSH_BLACK ...
#define TL_BRUSH_WHITE ...
```

Description

These constants represent standard XVT drawing tools. The `TL_PEN_*` constants can be used in a call to `xvt_dwin_set_std_cpen` to set a standard pen. The `TL_BRUSH_*` constants can be used in a call to `xvt_dwin_set_std_cbrush` to set a standard brush.

See Also

`xvt_dwin_set_std_cbrush`
`xvt_dwin_set_std_cpen`

The "Drawing and Pictures" chapter in the *XVT Portability Toolkit Guide*

TRUE

True Value

Summary

```
#define TRUE ...
```

Description

This symbol should be used for a true `BOOLEAN` value.

See Also

`BOOLEAN`
`FALSE`

Example

```
BOOLEAN flag = TRUE;
```

TX_* Attributes

Text Edit Attributes

Summary

```
#define TX_AUTOHSCROLL ...
#define TX_AUTOVSCROLL ...
#define TX_BORDER ...
#define TX_DISABLED ...
#define TX_ENABLECLEAR ...
#define TX_INVISIBLE ...
#define TX_NOCOPY...
#define TX_NOCUT ...
#define TX_NOMENU ...
#define TX_NOPASTE ...
#define TX_ONEPAR ...
#define TX_OVERTYPE ...
#define TX_READONLY ...
#define TX_WRAP ...
```

`TX_AUTOHSCROLL`

Enables automatic scrolling in the horizontal direction when the user drags the mouse outside of the text edit object.

`TX_AUTOVSCROLL`

Enables automatic vertical scrolling.

`TX_BORDER`

Draws a rectangular border around the text edit object.

`TX_DISABLED`

Disables the text edit system.

`TX_ENABLECLEAR`

Leaves the clear item in the edit menu always enabled.

`TX_INVISIBLE`

Sets text to invisible.

`TX_NOCOPY`

Disables the Copy command on the Edit menu.

`TX_NOCUT`

Disables the Cut command on the Edit menu.

`TX_NOMENU`

Prevents the text edit system from changing the menu of the window containing the text edit object. This attribute is especially useful if there is no edit menu.

`TX_NOPASTE`

Disables the Paste command on the Edit menu.

`TX_ONEPAR`

Ignores carriage returns, and thus limits editing to one paragraph.

`TX_OVERTYPE`

Enables "overtype mode" where users replace existing characters when typing instead of inserting characters in front of existing text.

`TX_READONLY`

Does not allow the user to make changes to the text.

`TX_WRAP`

Enables word wrap to the margin.

Description

The constants are OR'd together to form a set of attributes that control the operation of a text edit object. The functions accepting these OR'd combinations of `TX_*` attributes are `xvt_tx_create`, `xvt_tx_create_def`, and `xvt_tx_set_attr`. Indirectly, `xvt_win_create_def` also processes `TX_*` attributes because `xvt_win_create_def` can create text edit objects inside the window.

For a detailed description of each attribute, see `xvt_tx_create`.

See Also

```
xvt_tx_create
xvt_tx_get_attr
xvt_tx_set_attr
xvt_tx_*
xvt_win_create_def
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

```
unsigned attrib;
WINDOW win;
TXEDIT txedit;
RCT rct;
XVT_FNTID my_font
...
attrib = TX_BORDER | TX_WRAP |
        TX_AUTOHSCROLL | TX_AUTOVSCROLL;
if ((txedit = xvt_tx_create(win, &rct, attrib, my_font,
    rct.right - rct.left, INT_MAX)) == NULL_TXEDIT)
    xvt_dm_post_fatal_exit("Couldn't create txedit");
```

U_* Values for UNIT_TYPE

Identify Coordinate System used for WIN_DEF Elements

Summary

```
typedef enum e_unit_type {
    U_PIXELS,
    U_CHARS,
    U_SEMICHARS
} UNIT_TYPE;
```

Description

The `WIN_DEF` structure is currently the only place in the API where `UNIT_TYPE` is found. This type is used to identify the coordinate system used for individual elements defined in a `WIN_DEF` array passed to functions like `xvt_win_create_def` or `xvt_dlg_create_def`.

Individual elements of the `WIN_DEF` array can be specified in terms of either a pixel, semi-character, or character coordinate system. The XVT Toolkits ensure that a sensible mapping is made to the native pixel coordinate system.

See Also

```
WIN_DEF
xvt_dlg_create_def
xvt_win_create
```

UCHAR_MAX

Max Unsigned Char Value

Summary

```
#define UCHAR_MAX ...
```

Description

This is XVT's implementation of the ANSI C symbol for the maximum value of a `unsigned char` variable.

See Also

```
CHAR_MAX
INT_MAX
LONG_MAX
SHRT_MAX
UNIT_MAX
ULONG_MAX
USHRT_MAX
```

UNIT_MAX

Max Unsigned int Value

Summary

```
#define UINT_MAX ...
```

Description

This is XVT's implementation of the ANSI C symbol for the maximum value of a `unsigned int` variable.

See Also

```
CHAR_MAX
INT_MAX
LONG_MAX
SHRT_MAX
UCHAR_MAX
ULONG_MAX
USHRT_MAX
```

ULONG_MAX

Max Unsigned Long Value

Summary

```
#define ULONG_MAX ...
```

Description

This is XVT's implementation of the ANSI C symbol for the maximum value of a `unsigned long` variable.

See Also

```
CHAR_MAX
INT_MAX
LONG_MAX
SHRT_MAX
UCHAR_MAX
UNIT_MAX
USHRT_MAX
```

USHRT_MAX

Max Unsigned Short Value

Summary

```
#define USHRT_MAX ...
```

Description

This is XVT's implementation of the ANSI C symbol for the maximum value of an `unsigned short` variable.

See Also

```
CHAR_MAX
INT_MAX
LONG_MAX
SHRT_MAX
UCHAR_MAX
UNIT_MAX
ULONG_MAX
```

W_*, WC_*, WD_*, Values for WIN_TYPE

Window-Type

Summary

```
typedef enum {/* type of window */
    W_NONE,    /* marker for end of WIN_DEF array */
    W_DOC,          /* document window */
    W_PLAIN,        /* window with plain border */
    W_DBL,          /* window with double border */
    W_PRINT,        /* XVT internal use only */
    W_TASK,         /* task window */
    W_SCREEN,       /* screen window */
    W_NO_BORDER,    /* no border */
    W_PIXMAP,       /* pixmap */
    W_MODAL,        /* modal window */
    WD_MODAL,       /* modal dialog */
    WD_MODELESS,    /* modeless dialog */
    WC_PUSHBUTTON,  /* button control */
    WC_RADIOBUTTON, /* radio button control */
    WC_CHECKBOX,    /* check box control */
    WC_HSCROLL,     /* horizontal scroll bar control */
    WC_VSCROLL,     /* vertical scroll bar control */
    WC_EDIT,        /* edit control */
    WC_TEXT,        /* static text control */
    WC_LBOX,        /* list box control */
    WC_LISTBUTTON,  /* button with list */
    WC_LISTEDIT,    /* edit field with list */
    WC_GROUPBOX,    /* group box */
    WC_TEXTEDIT,    /* text-edit object */
    WC_ICON,        /* icon control */
    WC_NOTEBK,      /* notebook control */
    WC_HTML,        /* html control */
    WC_NUM_WIN_TYPES,/* number of WIN_TYPEs */
    XVT_ENUM_DUMMY12 = XVT_CC_ENUM_END
} WIN_TYPE;
```

Description

Values of this type are used to specify or identify objects referred to by the `WINDOW` data type. The value of this type is a parameter to all of the window, dialog, and control creation functions that specify the object type. In addition, to identify the type of an existing `WINDOW`, a value of this type is also returned by `xvt_vobj_get_type`.

See Also

WIN_DEF
WINDOW
xvt_ctl_create
xvt_dlg_create_res
xvt_vobj_get_type
xvt_win_create
Window/Dialog/Control Creation Function Parameters

WSF_* Options Flags

Window-Creation Flags

Summary

```
#define WSF_CLOSE ...
#define WSF_DECORATED ...
#define WSF_DEFER_MODAL ...
#define WSF_DISABLED ...
#define WSF_HSCROLL ...
#define WSF_ICONIZABLE ...
#define WSF_ICONIZED ...
#define WSF_INVISIBLE ...
#define WSF_MAXIMIZED ...
#define WSF_NO_MENUBAR ...
#define WSF_NONE ...
#define WSF_PLACE_EXACT ...
#define WSF_SIZE ...
#define WSF_SIZEONLY ...
#define WSF_VSCROLL ...
```

Description

These flags are used when calling the `xvt_win_create` and `xvt_win_create_def` functions. They determine the style of the window to be created. Once a window is created, its style cannot be changed, with the exceptions noted below. The OR'd combination of these flags can be obtained by calling `xvt_vobj_get_flags`.

`WSF_CLOSE`

Specifies that a window should be created with a close control, which is usually in the upper-left corner of the window. On some systems, it can also add a "close" item to a window-manager menu. Valid only for top-level windows of type `W_DOC`.

`WSF_DECORATED`

A convenient combination of the above four flags.

WSF_DEFER_MODAL

Specifies that a W_MODAL window will not enter a modal state until the function xvt_win_process_modal is called. Normally a W_MODAL window will enter the modal state inside the xvt_win_create* function and will only return after the window is destroyed and modal processing is complete. This is only valid for windows of type W_MODAL.

WSF_DISABLED

Specifies that the window should be created initially disabled. To enable it, call xvt_vobj_set_enabled.

WSF_HSCROLL

Specifies that the window should be created with a horizontal scrollbar on its frame. Valid only for top-level windows of type W_DOC, or child windows of type W_PLAIN.

WSF_ICONIZABLE

Specifies that the window is iconizable. This is only valid for top-level windows of type W_DOC. It is ignored on XVT/Mac which does not support iconized windows.

WSF_ICONIZED

Specifies that the window should be created initially iconized. A window created initially iconized can be deiconized only by the user. This is only valid for top-level windows of type W_DOC. This flag is ignored on XVT/Mac which does not support iconized windows.

WSF_INVISIBLE

Specifies that the window should be created initially invisible. To make it visible, call xvt_vobj_set_visible.

WSF_MAXIMIZED

Specifies that the window should be created initially maximized (or zoomed, as some call it). A maximized window occupies its entire container. In addition, a maximized window might not respond to xvt_vobj_move calls until the user un-maximizes it. This is only valid for top-level windows of type W_DOC. This flag is ignored on XVT/XM.

WSF_NO_MENUBAR

Specifies that a top-level window has no menubar (child windows never have menubars). Windows with this flag do not receive E_COMMAND events. Note that the absence of this flag

does *not* imply that the window will have a menubar physically attached to its frame--in general, the physical placement of menubars is determined by the platform. Rather, this flag controls the presence or absence of a `logical` menubar, which is somehow operable when the window has focus.

`WSF_PLACE_EXACT`

Specifies that for a `W_MODAL` window the position of the top and left fields should be determined by the RCT structure.

`WSF_SIZE`

Specifies that a window should be created with a sizing control. This may mean that the window has a thick, sizeable frame on some platforms, whereas it may create certain "size controls" on the window's frame for other platforms. Valid only for top-level windows of type `W_DOC`.

`WSF_SIZEONLY`

Causes the window to have a small box in the lower-right corner for sizing. This is only valid for top-level windows of type `W_DOC`. In addition, this is only used on XVT/Mac, and only for windows without scrollbars.

`WSF_VSCROLL`

Specifies that the window should be created with a vertical scrollbar on its frame. Valid only for top-level windows of type `W_DOC`, or child windows of type `W_PLAIN`.

See Also

```
WINDOW
W_*, WC_*, WD_*, Values for WIN_TYPE
xvt_vobj_get_flags
Window/Dialog/Control Creation Function Parameters
```

Example

See the examples for `xvt_win_create` and `xvt_vobj_get_flags`.

XVT_CALLCONV*

Linkage Macros

Summary

```
#define XVT_CALLCONV1 ...
#define XVT_CALLCONV2 ...
```

Description

The `XVT_CALLCONV1` macro defines the linkage convention of functions on IBM-compatible PC platforms (XVT/Win32). On XVT/Mac, they define a C calling convention for linkage between C and C++ compiler-generated code. On other platforms, they are defined as empty macros. For the exact values used, see the **xvt_env.h** and **xvt_plat.h** files.

To ensure portability, you should use `XVT_CALLCONV1` in all prototypes and headers for XVT callback functions (including event handlers, hook functions, print threads, application-supplied font mappers, and application-supplied Font Selection dialogs). You should also use the `XVT_CALLCONV1` macro in the declaration of the `main` function.

Note: `XVT_CALLCONV2` is not currently defined and is reserved for future use.

See Also

xvt_env.h
xvt_plat.h

The *XVT Platform-Specific Books*
The compiler manuals for your platform

Example

You should declare a callback function prototype like this:

```
BOOLEAN XVT_CALLCONV1 key_hook(...)
```

You should declare the `main` function like this:

```
int XVT_CALLCONV1 main(int argc, char **argv);
```

XVT_CLUT_SIZE

Maximum Size of an Image Object Color Look-Up Table

Summary

```
#define XVT_CLUT_SIZE ...
```

Description

This constant defines the maximum number of colors in any `XVT_IMAGE` object's color look-up table. Note that this constant only applies to images of type `XVT_IMAGE_CL8`.

See Also

```
XVT_IMAGE
xvt_image_get_clut
xvt_image_set_clut
xvt_image_set_ncolors
```

XVT_COLOR_*

Color Constants

Summary

```
#define XVT_COLOR_NULL...
#define XVT_COLOR_BACKGROUND...
#define XVT_COLOR_BLEND...
#define XVT_COLOR_BORDER...
#define XVT_COLOR_FOREGROUND...
#define XVT_COLOR_HIGHLIGHT...
#define XVT_COLOR_SELECT..
#define XVT_COLOR_TROUGH...
```

`XVT_COLOR_BACKGROUND`

Fill color of rectangular region occupied by control

`XVT_COLOR_BLEND`

Secondary background for some controls. This provides that colors blend into their container window's background without visual indication of a border

`XVT_COLOR_BORDER`

Outside edge of control (rectangular)

`XVT_COLOR_FOREGROUND`

Control text and the arrows on scrollbars

`XVT_COLOR_HIGHLIGHT`

Visual indication that a control has keyboard focus

`XVT_COLOR_SELECT`

Indication that a control has been selected

`XVT_COLOR_TROUGH`

Slider area behind scrollbar thumb

Description

These constants define (in `XVT_COLOR_COMPONENT`) the component of a control (`fcXVT_COLOR_TYPE`) to which a color is applied.

See Also

`XVT_COLOR_COMPONENT`
`XVT_COLOR_TYPE`
`xvt_ctl_get_colors`
`xvt_ctl_set_colors`
`xvt_win_get_ctl_colors`
`xvt_win_set_ctl_colors`

XVT_COLOR_GET_BLUE

Returns the Blue Component of a Color

Summary

```
#define XVT_COLOR_GET_BLUE(color) ...
```

Description

This macro returns the blue component (`unsigned char`) of an XVT `COLOR` value.

See Also

`XVT_COLOR_GET_GREEN`
`XVT_COLOR_GET_RED`
`XVT_MAKE_COLOR`

XVT_COLOR_GET_GREEN

Returns the Green Component of a Color

Summary

```
#define XVT_COLOR_GET_GREEN(color) ...
```

Description

This macro returns the green component (`unsigned char`) of an XVT `COLOR` value.

See Also

```
XVT_COLOR_GET_BLUE
XVT_COLOR_GET_RED
XVT_MAKE_COLOR
```

XVT_COLOR_GET_RED

Returns the Red Component of a Color

Summary

```
#define XVT_COLOR_GET_RED(color) ...
```

Description

This macro returns the red component (`unsigned char`) of an XVT `COLOR` value.

See Also

```
XVT_MAKE_COLOR
XVT_COLOR_GET_BLUE
XVT_COLOR_GET_GREEN
```

XVT_CTOOLS_*

User-Modifiable Drawing Tool Constants

Summary

```
#define XVT_CTOOLS_PEN ...
#define XVT_CTOOLS_PEN_ALL ...
#define XVT_CTOOLS_BRUSH ...
#define XVT_CTOOLS_FORE_COLOR ...
#define XVT_CTOOLS_BACK_COLOR ...
#define XVT_CTOOLS_CTOOL ...
#define XVT_CTOOLS_ALL ...
```

`XVT_CTOOLS_PEN`

Allows the user to change color, width and style of the `DRAW_CTOOLS` pen.

`XVT_CTOOLS_PEN_ALL`

Allows the user to change all members of the `DRAW_CTOOLS` pen.

`XVT_CTOOLS_BRUSH`

Allows the user to change all members of the `DRAW_CTOOLS` brush.

`XVT_CTOOLS_FORE_COLOR`

Allows the user to change the `DRAW_CTOOLS` foreground color.

`XVT_CTOOLS_BACK_COLOR`

Allows the user to change the `DRAW_CTOOLS` background color.

`XVT_CTOOLS_CTOOL`

Allows the user to change all `DRAW_CTOOLS` members except mode and opaque_text.

`XVT_CTOOLS_ALL`

Allows the user to change all members of the `DRAW_CTOOLS` construct.

Description

These constants are used for calls to `xvt_dm_post_ctools_sel` and are based on the `DRAW_CTOOLS` construct.

XVT_CXO_*_MSG

Container Extension Object Message Constants

Summary

```
#define XVT_CXO_CREATE_MSG -1
#define XVT_CXO_DESTROY_MSG -2
```

Description

These constants represent standard XVT `E_CXO` messages which XVT dispatches when a CXO is created or destroyed. Additional messages may be defined in the future.

Implementation Note

XVT reserves the negative range for CXO messages, but CXO's may use positive values for their own messages.

See Also

`E_CXO`

Example

See the example for `xvt_cxo_create`.

XVT_CXO_POS_* Values for XVT_CXO_INSERTION

Container Extension Object Insertion Type

Summary

```
typedef enum
{
        XVT_CXO_POS_FIRST,
        XVT_CXO_POS_LAST
} XVT_CXO_INSERTION;
```

Description

The enumeration defines locations for inserting a CXO into a container's CXO chain.

Implementation Note

XVT reserves the negative range for CXO messages, but CXO's may use positive values for their own messages.

See Also

`xvt_cxo_create`

XVT_DISPLAY_* Values

Value for `ATTR_DISPLAY_TYPE`

Summary

`xvt_vobj_get_attr`

returns one value of an enumerated type, defined as follows:

```
typedef enum {
    XVT_DISPLAY_MONO,
    XVT_DISPLAY_GRAY_16,
    XVT_DISPLAY_GRAY_256,
    XVT_DISPLAY_COLOR_16,
    XVT_DISPLAY_COLOR_256,
    XVT_DISPLAY_DIRECT_COLOR
} XVT_DISPLAY_TYPE;
```

`XVT_DISPLAY_MONO`

A monochromatic, black-and-white display.

`XVT_DISPLAY_GRAY_16`

A grayscale display, capable of displaying 16 shades of gray (but no colors).

`XVT_DISPLAY_GRAY_256`

A grayscale display, capable of displaying 256 shades of gray (but no colors).

`XVT_DISPLAY_COLOR_16`

A color display, capable of displaying 16 distinct colors.

`XVT_DISPLAY_COLOR_256`

A color display, capable of displaying 256 distinct colors.

`XVT_DISPLAY_DIRECT_COLOR`

A full-color display, capable of displaying thousands (or more) of distinct colors.

See Also

`ATTR_DISPLAY_TYPE`

The "Portable Images" chapter in the *XVT Portability Toolkit Guide*

XVT_ESC_*

XVT Escape Codes

Description

Typically, these constants are used as arguments to `xvt_app_escape` to specify non-portable actions.

XVT defines only one portable escape code, `XVT_ESC_GET_PRINTER_INFO`, which gives you the printer information from the passed print record. Any of the return parameter pointers can be `NULL`. The call signature is:

```
xvt_app_escape(XVT_ESC_GET_PRINTER_INFO,
    PRINT_RCD* print_rcd, long* heightp,
    long* widthp, long* vresp, long* hresp);
```

`PRINT_RCD* print_rcd`

Current print record.

`long* heightp`

Height in dots.

`long* widthp`

Width in dots.

`long* vresp`

Vertical resolution in dots per inch.

`long* hresp`

Horizontal resolution in dots per inch.

See Also

`xvt_app_create`
`xvt_app_escape`
`xvt_dm_post_page_setup`

For details of non-portable XVT escape codes, see the *XVT Platform-Specific Books*

XVT_FA_* Constants

Logical Font Attribute Constants

Summary

```
#define XVT_FA_ALL ...       /* all attributes*/
#define XVT_FA_APP_DATA ... /* application data */
#define XVT_FA_FAMILY ...    /* family */
#define XVT_FA_NATIVE ..     /* native descriptor */
#define XVT_FA_SIZE ...      /* size */
#define XVT_FA_STYLE ...     /* style */
#define XVT_FA_WIN ...       /* window */
```

Description

These constants are used in an `XVT_FONT_ATTR_MASK` to specify logical font attribute types with `XVT_FNTID` access functions. You can use these constants combined in bit masks to specify multiple values.

See Also

```
XVT_FONT_ATTR_MASK
XVT_FNTID
XVT_FFN_* Constants
XVT_FS_* Constants
xvt_font_*
```

Example

Here is an example of how an application might combine the constants in a mask:

```
XVT_FNTID font_id1 = xvt_font_create();
XVT_FNTID font_id2 = xvt_font_create();
xvt_font_set_family(font_id1, "times");
xvt_font_set_style(font_id1, XVT_FS_BOLD |
    XVT_FS_ITALIC);
xvt_font_set_size(font_id1, 18);
xvt_font_copy(font_id2, font_id1,
    XVT_FA_FAMILY | XVT_FA_SIZE | XVT_FA_STYLE);
```

XVT_FAST_WIDTH

Fastest Pen Width For CPEN's

Summary

```
#define XVT_FAST_WIDTH ...
```

Description

This constant defines the `CPEN` width to use for fastest drawing speed.

Implementation Note

The definition of `XVT_FAST_WIDTH` is platform-specific. If you require the exact value of this macro, see the definition in the **xvt_plat.h** or the platform-specific file (or **xvt_plxs.h** on XVT/XM).

See Also

`CPEN`

XVT_FFN_* Constants

Logical Font Family Name Constants

Summary

```
#define XVT_FFN_COURIER ...   /* "courier" */
#define XVT_FFN_FIXED ...     /* "fixed" */
#define XVT_FFN_HELVETICA ... /* "helvetica" */
#define XVT_FFN_SYSTEM ...    /* "system" */
#define XVT_FFN_TIMES ...     /* "times" */
```

Description

XVT provides these constants as a convenience. They represent logical fonts guaranteed to have a good, consistent appearance on all toolkits. Any other logical font families may undergo some mapping substitutions when being ported across platforms.

Your application can specify the above predefined logical font family names strings as parameters to `xvt_font_set_family`.

See Also

```
XVT_FA_* Constants
XVT_FS_* Constants
xvt_font_*
```

XVT_FILE_ATTR_* Constants

XVT File Attribute Constants

Summary

```
#define XVT_FILE_ATTR_ATIME ...
#define XVT_FILE_ATTR_CREATORSTR ...
#define XVT_FILE_ATTR_CTIME ...
#define XVT_FILE_ATTR_DIRECTORY ...
#define XVT_FILE_ATTR_DIRSTR ...
#define XVT_FILE_ATTR_EXECUTE ...
#define XVT_FILE_ATTR_EXIST ...
#define XVT_FILE_ATTR_FILESTR ...
#define XVT_FILE_ATTR_NUMLINKS ...
#define XVT_FILE_ATTR_MTIME ...
#define XVT_FILE_ATTR_READ ...
#define XVT_FILE_ATTR_SIZE ...
#define XVT_FILE_ATTR_TYPESTR ...
#define XVT_FILE_ATTR_WRITE ...
```

`XVT_FILE_ATTR_EXIST`

Return type: `BOOLEAN`
Returns `TRUE` if the file exists and can be reached via a searchable path of directories.

`XVT_FILE_ATTR_READ`

Return type: `BOOLEAN`
Returns `TRUE` if the file exists and is readable.

`XVT_FILE_ATTR_WRITE`

Return type: `BOOLEAN`
Returns `TRUE` if the file exists and is writable.

`XVT_FILE_ATTR_EXECUTE`

Return type: `BOOLEAN`
Returns `TRUE` if the file exists and can be executed.

`XVT_FILE_ATTR_DIRECTORY`

Return type: `BOOLEAN`
Returns `TRUE` if the file is a directory.

`XVT_FILE_ATTR_NUMLINKS`

Return type: `long`
Returns the number of hard links to the file. This is meaningful only on XVT/XM. It returns 1 on other platforms.

`XVT_FILE_ATTR_SIZE`

Return type: `long`
Returns the number of bytes in the file. On XVT/Mac, this is the size of the data segment.

`XVT_FILE_ATTR_ATIME`

Return type: `time_t`
Returns the access time of the file (i.e., the last time and date the file was read/written/executed). On the XVT/Mac platform, this is the same as the modified time.

`XVT_FILE_ATTR_CTIME`

Return type: `time_t`
Returns the creation date of the file.

`XVT_FILE_ATTR_MTIME`

Return type: `time_t`
Returns the date the file was last modified.

`XVT_FILE_ATTR_DIRSTR`

Return type: `char *`
Returns the directory component of the input `FILE_SPEC` structure as a `NULL`-terminated string. This is equivalent to calling `xvt_fsys_convert_dir_to_str` on the directory component of the input `FILE_SPEC` structure.

`XVT_FILE_ATTR_FILESTR`

Return type: `char *`
Returns the file component of the input `FILE_SPEC` structure as a `NULL`-terminated string.

`XVT_FILE_ATTR_TYPESTR`

Return type: `char *`
Returns the type component of the input `FILE_SPEC` structure as a `NULL`-terminated string. On the XVT/Mac platform, this is the system file type.

`XVT_FILE_ATTR_CREATORSTR`

Return type: `char *`
Returns the creator component from the input `FILE_SPEC` structure. On XVT/Mac, this is the system file creator.

Description

These constants are used in `xvt_fsys_set_file_attr` and `xvt_fsys_get_file_attr` to specify the file attribute that is being set or inquired.

See Also

`xvt_fsys_get_file_attr`
`xvt_fsys_set_file_attr`

Example

```
if (FL_OK == xvt_dm_post_file_open(&fs,
    "Open File ...")) {
    sprintf(lines[i++], "testing file: %s",
        (char *) xvt_fsys_get_file_attr(&fs,
            XVT_FILE_ATTR_FILESTR));
if (!xvt_fsys_get_file_attr(&fs, XVT_FILE_ATTR_EXIST))
        sprintf(lines[i++], "File does not exist");
    ...
}
```

XVT_FILESYS_* Values

File System Macros

Summary

```
#define XVT_FILESYS_MAC ... /* Apple Macintosh file
                               system */
#define XVT_FILESYS_NTFS .../* MS-Windows NT File System
*/
#define XVT_FILESYS_UNIX .../* UNIX file system */
```

Description

These macros are set to a value of either `TRUE` or `FALSE`, depending on whether or not the *operating* system under which the application is executing *may* be supporting the *file* system. Because some operating systems can support more than one file system simultaneously, the macros indicate what *may* be supported and not the actual file system configuration.

You can use these macros both in coding and in URL resources. These macros are defined in the **xvt_env.h** file and are set in the platform-specific file **xvt_plat.h** (**xvt_plxs.h** on XVT/XM).

Implementation Note

Although you could use the XVT_OS macro to determine the file system, XVT strongly encourages you to use the XVT_FILESYS_* macros. They provide a more consistent and simpler way to determine file systems. Also, the supported values of the XVT_OS macro are subject to change between releases of XVT as support for various operating systems is added or removed.

See Also

XVTWS, *WS Values
xvt_env.h
xvt_plat.h (or **xvt_plxs.h** on XVT/XM)

Example

```
#if XVT_FILESYS_NTFS || XVT_FILESYS_DOS
    /* platform-specific file system code with DOS-like
        file naming and directory conventions */
#endif#if XVT_FILESYS_UNIX
    /* platform-specific file system code with UNIX
        file naming and directory conventions */
#endif
```

XVT_FS_* Constants

Logical Font Style Constants

Summary

```
#define XVT_FS_BLINK ...    /* blinking */
#define XVT_FS_BOLD ...     /* bold */
#define XVT_FS_INVERSE ...  /* inverse */
#define XVT_FS_ITALIC ...   /* italic */
#define XVT_FS_NONE ...     /* none */
#define XVT_FS_OUTLINE ...  /* outline */
#define XVT_FS_SHADOW ...   /* shadow */
#define XVT_FS_STRIKEOUT .../* strikeout */
#define XVT_FS_UNDERLINE .../* underline */
#define XVT_FS_USER1 ...    /* for application use */
#define XVT_FS_USER2 ...    /* for application use */
#define XVT_FS_USER3 ...    /* for application use */
#define XVT_FS_USER4 ...    /* for application use */
#define XVT_FS_USER5 ...    /* for application use */
#define XVT_FS_WILDCARD ... /* used only in URL created
                            logical fonts */
```

Description

These constants are used in an `XVT_FONT_STYLE_MASK` to specify logical font style with `XVT_FNTID` access functions. You can use the `XVT_FS_*` constants combined in bit masks to specify multiple values. The `XVT_FS_WILDCARD` logical font style is the only style a logical font has if it was created from an URL `font` resource with a style of "`any`."

See Also

```
XVT_FA_* Constants
XVT_FFN_* Constants
XVT_FNTID
XVT_FONT_STYLE_MASK
font URL Statement
```

The "Menus" and the "Resources and URL" chapters in the *XVT Portability Toolkit Guide*

XVT_HELP_* Values for XVT_HELP_FLAVOR

Configuration of the Help Viewer

Summary

```
typedef enum {
    XVT_HELP_FLAVOR_NONE,        /* no viewer */
    XVT_HELP_FLAVOR_NTVSRV,      /* native standalone
                                 (server) viewer */
    XVT_HELP_FLAVOR_NTVBND,      /* native bound viewer */
    XVT_HELP_FLAVOR_PORTSRV, /* portable standalone
                                 (server) viewer */
    XVT_HELP_FLAVOR_PORTBND      /* portable bound viewer
*/
}  XVT_HELP_FLAVOR;
```

Description

XVT supports several help viewer configurations. These enumeration constants identify which configuration is being used by an application.

See Also

`xvt_help_get_flavor`

XVT_IMAGE_* Values for XVT_IMAGE_FORMAT

Color Format Enumerated Type for Images

Summary

```
typedef enum {
    XVT_IMAGE_NONE,
    XVT_IMAGE_CL8,
    XVT_IMAGE_RGB
    XVT_IMAGE_MONO,
} XVT_IMAGE_FORMAT;
```

Description

Enumerated type for identifying the color format of an `XVT_IMAGE` variable. The values correspond to the following color formats:

`XVT_IMAGE_NONE`

No image format.

`XVT_IMAGE_CL8`

Indexed image with a 256-entry color look-up table, one byte per pixel.

`XVT_IMAGE_RGB`

Full color image, one 24-bit `COLOR` variable per pixel.

`XVT_IMAGE_MONO`

Monochrome (black and white) or two-color image, one bit per pixel.

See Also

`xvt_image_create`
`xvt_image_get_format`

XVT_MAKE_COLOR

Create a Color

Summary

```
#define XVT_MAKE_COLOR(r, g, b) ...
```

Description

Use this macro to create `COLOR`s from separate red, green, and blue intensity values.

In addition to creating your own `COLOR` values, you can use one of the eleven predefined `COLOR`s described under the `COLOR_*` constants topic. The range of values for `red`, `green`, and `blue` are `0` to `0xFF` (`unsigned char`).

For a description of how RGB colors are stored in XVT, see the topic `COLOR`.

Return Value

The `COLOR`.

See Also

```
COLOR_*, COLOR_INVALID Constants
XVT_COLOR_GET_BLUE
XVT_COLOR_GET_GREEN
XVT_COLOR_GET_RED
```

The "Drawing and Pictures" chapter in the *XVT Portability Toolkit Guide*

Example

This code shows how to create red:

```
COLOR red = XVT_MAKE_COLOR (0xFF, 0x00, 0x00)
```

This code shows how to create magenta:

```
COLOR magenta = XVT_MAKE_COLOR (0xFF, 0x00, 0xFF)
```

XVT_MAX_MB_SIZE

Maximum Size of the Largest Multibyte Character

Summary

```
#define XVT_MAX_MB_SIZE ...
```

Description

This constant is defined as the maximum size in bytes of the largest multibyte character on a particular platform and compiler.

Example

```
char mbc[XVT_MAX_MB_SIZE];
int len;len = xvt_str_convert_wc_to_mb(mbc, wc);
```

XVT_MAX_WINDOW_RECT

Maximum Window Size Constant

Summary

```
#define XVT_MAX_WINDOW_RECT ...
```

Description

This constant is passed to the `xvt_win_create` window-creation function to create a top-level window that occupies its entire container.

This constant is slightly different than specifying the `WSF_MAXIMIZED` flag, as the latter has particular implications about the state of the resizing frame controls for the window. In addition, a window created with `WSF_MAXIMIZED` might not respond to `xvt_vobj_move` calls until the user un-maximizes it, whereas a window created with `XVT_MAX_WINDOW_RECT` will respond to `xvt_vobj_move` calls.

See Also

`WSF_* Options Flags`
`xvt_win_create`
`xvt_win_create_def`
`xvt_win_create_res`
`xvt_vobj_move`

XVT_MOD_KEY

Modify Keys

Summary

```
#define XVT_MOD_KEY_ALT
#define XVT_MOD_KEY_CMD
#define XVT_MOD_KEY_COMPOSE
#define XVT_MOD_KEY_CTL
#define XVT_MOD_KEY_LSHIFT
#define XVT_MOD_KEY_NONE
#define XVT_MOD_KEY_OPTION
#define XVT_MOD_KEY_RSHIFT
#define XVT_MOD_KEY_SHIFT
```

Description

These symbols represent modifier key bitfields. All available modifier keys are passed in the `E_CHAR` event for use by the application. Constant values are defined for testing for particular modifier keys. Combinations of modifiers may be tested against the bitfield.

Some keys never occur with certain keyboards or on some platforms. The application should take this into account for portability.

See Also

```
ATTR_KEY_HOOK
ATTR_MULTIBYTE_AWARE
E_CHAR
```

XVT_NAV_INSERTION

Navigation Object Insertion Flag

Summary

```
typedef enum e_nav_insertion
{
    XVT_NAV_POS_FIRST,
    XVT_NAV_POS_LAST,
    XVT_NAV_POS_BEFORE,
    XVT_NAV_POS_AFTER
} XVT_NAV_INSERTION;
```

Description

`xvt_nav_add_win` uses values of this type to indicate where a new GUI object is inserted in the navigation order of a window.

See Also

```
XVT_NAV
xvt_nav_add_win
```

XVT_PALLETE_* Values

Color Pallet Types

Summary

```
typedef enum {
    XVT_PALETTE_NONE,
    XVT_PALETTE_STOCK,
    XVT_PALETTE_CURRENT,
    XVT_PALETTE_CUBE16,
    XVT_PALETTE_CUBE256,
    XVT_PALETTE_USER
} XVT_PALETTE_TYPE
```

Description

XVT_PALETTE_TYPE enumerates the available types of XVT_PALETTE color palettes. None of the values are modifiable by your application except for XVT_PALLET_USER. The values are interpreted as follows:

XVT_PALETTE_STOCK

Contains platform-specific colors that are intended to be compatible with the platform's default color scheme. This palette type is used for the initial default palette.

XVT_PALETTE_CURRENT

Contains the color values currently used by the system's display color palette. This palette type minimizes color flashes and other undesireable effects produced when switching between different windows and applications on one display. The number of colors and their values will vary depending on the system's current display palette.

XVT_PALETTE_CUBE16

Contains 16 "basic" color values. It is primarily intended for systems limited to 16-color displays.

XVT_PALETTE_CUBE256

Contains 256 evenly distributed color values, including 16 shades of grey and a uniform set of color hues and saturations. It is based on the Macintosh default color palette.

XVT_PALETTE_USER

Is freely modifiable by your application. When created, it initially contains enough basic system color values to ensure that menus and window decorations can be rendered. There will be no more than 32 of these pre-allocated colors for 256-color systems, and no more than two for 16-color systems.

See Also

XVT_PALETTE
xvt_palet_create
xvt_palet_get_type

XVT_PALETTE_SIZE

Maximum of a Pallette Object

Summary

```
#define XVT_PALETTE_SIZE ...
```

Description

This constant defines the maximum number of colors in any `XVT_PALETTE` object. Note that if `ATTR_DISPLAY_TYPE` is `XVT_DISPLAY_DIRECT_COLOR`, this constant is ignored since unlimited colors are available.

See Also

```
ATTR_DISPLAY_TYPE
XVT_PALETTE
xvt_palet_add_colors
xvt_palet_get_ncolors
xvt_palet_get_size
```

XVT_PIXMAP_* Values

Color Image Types

Summary

```
typedef enum {
    XVT_PIXMAP_NONE
    XVT_PIXMAP_DEFAULT,
} XVT_PIXMAP_FORMAT;
```

Description

`XVT_PIXMAP_FORMAT` enumerates the available types of `XVT_PIXMAP` color images. Currently, the only supported format is `XVT_PIXMAP_DEFAULT`, which represents a pixmap whose color format matches the display hardware.

See Also

```
XVT_PIXMAP
xvt_pmap_create
```

XVT_STRING_RES_BASE

Start of XVT String Constants

Summary

```
#define XVT_STRING_RES_BASE ...
```

Description

This constant indicates the start of string resources used internally by XVT. You should create string resources with IDs less than `XVT_STRING_RES_BASE`.

XVT_TIMER_ERROR

Timer Error

Summary

```
#define XVT_TIMER_ERROR ...
```

Description

This constant is returned by `xvt_timer_create` to indicate an error. See that function for details.

See Also

`xvt_timer_create`

XVT_TPC_* Constants

Help System Macros

Summary

```
#define XVT_TPC_HELPONHELP .../* Information about the
                                help system */
#define XVT_TPC_KEYBOARD ...  /* Information about
                                 special keys */
#define XVT_TPC_INDEX          /* Help index */
#define XVT_TPC_CONTENTS      /* Help table of
                                 contents */
#define XVT_TPC_TUTORIAL      /* Application tutorial
                                 information */
#define XVT_TPC_ONVERSION     /* Application version
                                 information */
#define XVT_TPC_GLOSSARY      /* Glossary of terms */
#define XVT_TPC_FILE_OPEN ... /* xvt_dm_post_file_open
*/
#define XVT_TPC_FILE_SAVE ... /* xvt_dm_post_file_save
*/
#define XVT_TPC_ASK ...          /* xvt_dm_post_ask */
#define XVT_TPC_NOTE ...      /* xvt_dm_post_note */
#define XVT_TPC_ERROR ...     /* xvt_dm_post_error */
#define XVT_TPC_WARNING ...   /* xvt_dm_post_warning */
#define XVT_TPC_STRING_PROMPT.../*
xvt_dm_post_string_prompt */
#define XVT_TPC_FONT_SEL ...  /* xvt_dm_post_font_sel */
#define XVT_TPC_PAGE_SETUP .../*  xvt_dm_post_page_setup
*/
#define XVT_TPC_MESSAGE ...   /* xvt_dm_post_message */
#define XVT_TPC_FATAL ...     /* xvt_dm_post_fatal_exit
*/
```

Description

The first set of symbols are reserved topic identifiers, which correspond to the items on the predefined Help menu.

The second set of symbols are predefined IDs. When help is requested while a predefined XVT modal dialog is active, an `E_HELP` event is dispatched to the task event handler. The `tid` member of the help event structure is set to one of the predefined IDs.

See Also

`E_HELP`
`xvt_help_*`

The "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*

XVTWS, *WS Values

Windowing System Macros

Summary

```
#define XVTWS ...              /* window system macro */
#define XVT_WS_UNKNOWN .../* unsupported window system
                               (error condition) */
#define MACWS ...              /* Apple Macintosh */
#define MTFWS ...              /* Motif */
#define WIN32WS ...            /* Win32 */
```

Description

You can use the `XVTWS` macro to determine the window system under which an XVT application is executing. It is assigned the value of one of the `*WS` macros. You can use it either in coding your application or in your URL resources. These macros are defined in the **xvt_env.h** file and are set in the platform-specific file **xvt_plat.h**.

Implementation Note

Although you could use the `XVT_OS` macro to determine the window system, XVT strongly encourages you to use the `XVTWS` macro. It provides a more consistent and simpler way to determine the window system. Also, the supported values of the `XVT_OS` macro are subject to change between releases of XVT as support for various operating systems is added or removed.

See Also

```
XVT_FILESYS_* Values
xvt_env.h
xvt_plat.h
```

XVT Functions

Listed by Object

`xvt_app_*`	Application Objects (global executable context)
`xvt_cb_*`	Clipbard Functions
`xvt_ctl_*`	Functionality Specific to Controls (contrary to windows)
`xvt_cxo_*`	Container Extension Object Functions
`xvt_debug_*`	Debugging Facility
`xvt_dlg_*`	User-Writen Dialog Support
`xvt_dm_*`	Dialog Manager, controlling built-in dialogs
`xvt_dwin_*`	Object Supporting Drawing Operations (windows and pixmaps)
`xvt_errid_*`	Error Message Identifiers
`xvt_errmsg_*`	Error Handling Facility
`xvt_event_*`	Event Access
`xvt_fmap_*`	Font Mapper Facility
`xvt_font_*`	Font Objects
`xvt_fsys_*`	File System Under the Application
`xvt_gmem_*`	Global Memory Management (Mac relocatable)
`xvt_help_*`	Help System
`xvt_html_*`	HTML Object
`xvt_image_*`	Image Objects
`xvt_iostr_*`	Input/Output Byte Stream
`xvt_list_*`	List Box, List Edit
`xvt_mem_*`	Memory Allocation Facility
`xvt_menu_*`	Application Menu Components
`xvt_nav_*`	Navigation Objects
`xvt_notebk_*`	Notebook Functions
`xvt_palet_*`	Color Palette Object
`xvt_pattern_*`	Complex Pattern Facility
`xvt_pict_*`	Picture Objects
`xvt_pmap_*`	Pixmap Objects
`xvt_print_*`	Printing Context
`xvt_rect_*`	Rectangle Objects
`xvt_res_*`	Resource Manager
`xvt_sbar_*`	Scrollbar Objects
`xvt_scr_*`	Screen Objects
`xvt_slist_*`	List of Tagged Strings
`xvt_str_*`	String Operations
`xvt_timer_*`	Timer Objects
`xvt_tx_*`	Portable, XVT Look-and-Feel Text Object
`xvt_vobj_*`	Visible Objects (windows, dialogs, and controls)

Miscellaneous Functions

max
min
NOREF
PTR_LONG

max

Get Maximum of Two Quantities

Summary

scalar_type max(scalar_type x, scalar_type y)

scalar_type x

First value to compare.

scalar_type y

Second value to compare.

Description

This macro returns the maximum of two quantities of any scalar type. Don't call it with arguments that have side effects (e.g., i++), because it might evaluate an argument more than once.

Return Value

Maximum (same type as arguments).

See Also

min

Example

```
/* ensure value is between 0 & 100 */
value = max (0, value);
value = min (100, value);
```

min

Get Minimum of Two Quantities

Summary

```
scalar_type min(scalar_type x, scalar_type y)
```

`scalar_type x`

First value to compare.

`scalar_type y`

Second value to compare.

Description

This macro returns the minimum of two quantities of any scalar type. Don't call it with arguments that have side effects (e.g., `i++`), as it may evaluate an argument more than once.

Return Value

Minimum (same type as arguments).

See Also

`max`

Example

See the example for `max`.

NOREF

Avoid "Unused Argument" Warning

Summary

```
void NOREF(any_type arg)
```

`any_type arg`

Argument to reference.

Description

This macro establishes a reference to an otherwise unused argument to a function so as to suppress a compiler warning. The `NOREF` statement must follow all other variable definitions.

Example

```
void fcn(x, y)
int x, y;
{
    int i;
    NOREF(x);
    i = fcn2(y);
}
```

PTR_LONG

Cast Pointer to Long

Summary

```
long PTR_LONG(pointer-type any_pointer)
```

`pointer-type` `any_pointer`

Any pointer-type.

Description

This macro casts a pointer to `long` in a way that avoids a compiler warning. Use it with the window, control, and dialog creation functions or with `xvt_vobj_set_data` or with `xvt_vobj_set_attr` to cast a data structure pointer to a `long` argument.

When casting a `long` back into a pointer, a simple cast suffices.

Return Value

A pointer represented as `long` integer.

See Also

`xvt_ctl_create_def`
`xvt_dlg_create_def`
`xvt_dlg_create_res`
`xvt_win_create_def`
`xvt_win_create_res`
`xvt_vobj_set_data`
`xvt_vobj_set_attr`

Example

```
struct sel_state {
    ...
};
    ...
struct sel_state *state = (struct scl_state *)
        xvt_mem_alloc(sizeof(struct sel_state));
...
xvt_vobj_set_data(win, PTR_LONG(state));
```

xvt_app_*

Application Objects (Global Executable Context)

```
xvt_app_allow_quit
xvt_app_create
xvt_app_destroy
xvt_app_escape
xvt_app_get_default_ctools
xvt_app_get_file
xvt_app_get_files_count
xvt_app_process_pending_events
xvt_app_set_file_processed
```

xvt_app_allow_quit

Agree to Termination of Application

Summary

```
void xvt_app_allow_quit(void)
```

Description

This function tells XVT that the application is willing to quit the next time it receives an `E_QUIT` event with the `query` member of the `EVENT` structure set to `FALSE`. This is used on systems where a two-stage shutdown occurs. Only the task event handler will receive `E_QUIT` events.

Since there will be no opportunity to cancel, call `xvt_app_allow_quit` only upon receiving an `E_QUIT` event with the `query` member set to `TRUE`, and only after all documents have been saved and the user has agreed to quit.

Implementation Note

This function has no effect on most systems. Subsequently your usage of it can be adequately tested only on XVT/Win32, since this is the only platform that receives `E_QUIT` events.

See Also

XVT Events
`E_QUIT`
`xvt_app_destroy`

The "Events" chapter in the *XVT Portability Toolkit Guide*

Example

```
case E_QUIT:
    if (xdEvent->v.query)
        xvt_app_allow_quit();
    else
        xvt_app_destroy();
```

xvt_app_create

Initiate and Initialize System

Summary

```
void xvt_app_create(int argc, char *argv[],
    unsigned long flags, EVENT HANDLER eh,
    XVT_CONFIG *config)
```

`int argc`

Set to the `argc` argument passed to your application's `main` function.

`char *argv[]`

Set to the `argv` argument passed to your application's `main` function.

`unsigned long flags`

Currently unused; pass zero for this parameter.

`EVENT HANDLER eh`

Set to your application's task event handler. It will be the first event handler to receive events, and the first event it receives will be an `E_CREATE` event.

`XVT_CONFIG *config`

Set to point to an `XVT_CONFIG` structure that specifies various aspects of your application. For details, see `XVT_CONFIG`.

Description

This function, usually called from `main`, invokes the XVT system.

It does not return a value, and does not return to the application. As events occur, designated event handlers for windows, dialogs, and the application itself are called.

A call to `xvt_app_create` is required before an application can access the XVT library. No other XVT functions can be called before `xvt_app_create`, with the exception of some attribute values that can be accessed via `xvt_vobj_get_attr` and `xvt_vobj_set_attr`.

Since this function does not return, your application must wait for an event to be sent to event handlers in order to gain control. The first such event is an `E_CREATE` event sent to your task event handler, and at that point you normally perform application initialization.

Parameter Validity and Error Conditions

XVT issues an error if any of the following parameter conditions are not met:

- `flags` value must be zero
- `eh` must not be `NULL`
- `config` must not be `NULL`
- `config->menu_bar_ID` must specify a valid menubar resource
- `config->base_appl_name`, `config->appl_name`, and `config->taskwin_title` must all be non-`NULL`

See Also

XVT Portable Attributes
`E_CREATE`
`EVENT_HANDLER`
`XVT_CONFIG`
`TASK_WIN`
`xvt_app_destroy`
`xvt_vobj_get_attr`
`xvt_vobj_set_attr`

The "Events" and the "About the XVT API" chapters in the *XVT Portability Toolkit Guide*
The *XVT Platform-Specific Books*

Example

The following example is taken from the Image Editor example:

```
    long XVT_CALLCONV1 task_eh(WINDOW, EVENT
*);XVT_CONFIG xvt_config = {
        TASK_MENUBAR, /* task window menu bar ResID */
        0, /* default aboutbox ResID */
        "imagedit", /* application's "file name" */
        "XVT Image Editor", /* application's name */
        "XVT Image Editor" /* title for task window */
    };int XVT_CALLCONV1 main(int argc, char *argv[])
    {
        xvt_app_create(argc, argv, 0L, task_eh,
            &xvt_config);
    }
```

xvt_app_destroy

Terminate Execution

Summary

```
void xvt_app_destroy(void)
```

Description

This function immediately terminates the application and sends an `E_DESTROY` event to the task event handler. It does not return.

You normally call `xvt_app_destroy` when you get an `E_QUIT` event with the `v.query` member set to `FALSE`. In addition, you might want to call `xvt_app_destroy` when the task handler gets an `E_CLOSE` event, or when any handler gets an `E_COMMAND` event with its tag set to `M_FILE_QUIT`. If you need to abort the application in an emergency, call `xvt_dm_post_fatal_exit`, which will call `xvt_app_destroy` after displaying an error message.

Note: Terminating the application does not necessarily send `E_DESTROY` events to any event handler other than the task event handler.

See Also

```
E_CLOSE
E_COMMAND
E_QUIT
E_DESTROY
M_EDIT_*, M_FILE_*, M_HELP_* Menu Tags
TASK_WIN
xvt_app_create
xvt_dm_post_fatal_exit
```

Example

```
case E_COMMAND:
    if (xdEvent->v.cmd.tag == M_FILE_QUIT)
        xvt_app_destroy();
    break;
```

xvt_app_escape

Perform Platform-Specific Action

Summary

```
BOOLEAN xvt_app_escape(int esc_code [, arg]...)int
    esc_code [, arg]...
```

Platform-specific escape code and zero or more arguments.

Description

This function allows access to non-portable, platform-specific functionality via a consistent interface.

Each platform can define escape codes that your application can use to perform non-portable actions. For each escape code so defined, there is also a definition of the arguments that should be passed to `xvt_app_escape` for a particular escape code.

XVT defines only one portable escape code, `XVT_ESC_GET_PRINTER_INFO`, which gives you the printer information from the passed print record. Any of the return parameter pointers can be `NULL`. The call signature is:

```
xvt_app_escape(XVT_ESC_GET_PRINTER_INFO,
    PRINT_RCD* print_rcd, long* heightp,
    long* widthp, long* vresp, long* hresp);
```

```
PRINT_RCD* print_rcd
```

Current print record.

`long* heightp`

Height in dots.

`long* widthp`

Width in dots.

`long* vresp`

Vertical resolution in dots per inch.

`long* hresp`

Horizontal resolution in dots per inch.

Note: You might find that two or more platforms support the same escape code. In that case, the arguments for the escape code will be the same between the platforms that share that escape code. However, you should not assume that all platforms implement the same escape code. You must always refer to the *XVT Platform-Specific Books* for definitions of supported escape codes.

Return Value

`TRUE` if successful; `FALSE` otherwise.

Parameter Validity and Error Conditions

XVT issues an error if you pass an unsupported escape code to this function. In addition, each platform might define error conditions for a particular escape code. Refer to the *XVT Platform-Specific Books* for these conditions.

See Also

`XVT_ESC_*`
`xvt_dm_post_page_setup`

For definitions of particular escape codes and their arguments, see the *XVT Platform-Specific Books*.

xvt_app_get_default_ctools

Get Normal Color Drawing Tools

Summary

```
DRAW_CTOOLS *xvt_app_get_default_ctools
    (DRAW_CTOOLS *ctoolsp)

DRAW_CTOOLS *ctoolsp
```

Pointer to the color drawing tools.

Description

This function stores a set of normal (or standard) color tools used to specify the colors and patterns that drawing functions use when drawing in a regular window or print window.

These colors of the ctools are set in the `DRAW_CTOOLS` structure pointed to by `ctoolsp`:

- A one-pixel-wide black `CPEN`
- A white `CBRUSH`
- A `DRAW_MODE` of `M_COPY`
- A foreground color of `COLOR_BLACK`
- A background color of `COLOR_WHITE`
- A `FALSE` value for opaque text

The drawing functions that use the ctools are `xvt_dwin_*`.

To set the tools for a particular window, you have to use the `DRAW_CTOOLS` retrieved by `xvt_app_get_default_ctools` in a call to `xvt_dwin_set_draw_ctools`.

Return Value

Value of `ctoolsp`.

Parameter Validity and Error Conditions

XVT issues an error if `ctoolsp` is `NULL`.

See Also

```
CBRUSH
COLOR
CPEN
DRAW_CTOOLS
DRAW_MODE
M_* Values for DRAW_MODE
xvt_dwin_get_draw_ctools
xvt_dwin_set_draw_ctools
```

Example

See the example for `xvt_dwin_get_draw_ctools`.

xvt_app_get_file

Get Next File to be Printed or Opened

Summary

```
FILE_SPEC *xvt_app_get_file(void)
```

Description

This function returns a pointer to a `FILE_SPEC` describing the next file to be opened or printed, among those files that were selected by the user when the application was invoked. XVT does not check to see if the selected files are valid files (for example, if they were entered on the command line). Therefore, to determine if the files exist your application should call:

```
xvt_fsys_get_file_attr(fp, XVT_FILE_ATTR_EXIST)
```

Use `xvt_app_get_files_count` to determine the number of selected files, and whether they are to be opened or printed. After each file is opened or printed, you should call `xvt_app_set_file_processed` to tell XVT that the file has been processed.

To retrieve all the files to be opened or printed, `xvt_app_get_file` is typically called in the `E_CREATE` case of an application's task event handler. When `xvt_app_get_file` returns `NULL`, all selected files have been passed to the application.

If the files are to be printed, your application should exit after printing them, by calling `xvt_app_destroy`. If they are to be opened, you should open them (usually by creating a document window for each file) and simply return from your task event handler so that normal processing continues.

Return Value

A pointer to a `FILE_SPEC` if successful; `NULL` when no files remain.

Implementation Note

On all platforms except XVT/Mac, this function uses the `argv` that you pass to `xvt_app_create`. Unlike other platforms, the Mac doesn't have a command line, so the names are guaranteed to be valid files.

See Also

```
FILE_SPEC
XVT_FILE_ATTR_* Constants
xvt_app_create
xvt_app_get_files_count
xvt_app_set_file_processed
xvt_fsys_build_pathname
xvt_fsys_get_file_attr
```

The "Files" chapter in the *XVT Portability Toolkit Guide*

Example

```
int num_files;
BOOLEAN print_files;/* convert files on command line to
bmp format */
xvt_app_get_files_count(&print_files, &num_files);
if (num_files > 0 && print_files == FALSE)
{
    while (num_files--)
    {
        if (convert_to_bmp(xvt_app_get_file(), FALSE))
            xvt_app_set_file_processed();
        }xvt_dm_post_note(
            "Images have been converted to bmp");
}
```

xvt_app_get_files_count

Get Count of Files to be Printed or Opened

Summary

```
void xvt_app_get_files_count(BOOLEAN *printp,
    int *countp)
```

`BOOLEAN *printp`

Files to be printed or opened.

`int *countp`

Number of files.

Description

This function returns through `countp` the number of files that were selected by the user when the application started. The value returned through `printp` indicates whether the files should be opened (`FALSE`) or printed (`TRUE`).

You should call xvt_app_get_files_count when your application starts up, preferably in the E_CREATE case of the task event handler. If the count is greater than zero, you should then call xvt_app_get_file repeatedly to get each file's name, until it returns NULL. After processing a file, you call xvt_app_set_file_processed to indicate that you are done with it.

If the files are to be printed, your application should exit after printing them, by calling xvt_app_destroy. If they are to be opened, you should open them (usually by creating a document window for each file) and simply return from your task event handler so that normal processing continues.

Parameter Validity and Error Conditions

printp must not be NULL. countp must not be NULL.

Implementation Note

If the user starts the application without specifying a file (e.g., the user double clicks on the application in XVT/Mac or XVT/Win32), xvt_app_get_files_count returns a count of 0. If the user starts the application by opening (or printing) a file associated with the application (e.g., the user double clicks on a data file associated with the application on XVT/Mac and XVT/Win32), xvt_app_get_file_count returns a count of 1. A count larger than 1 is returned when the user starts the application with more than one data file. For example, on XVT/Mac, the user can "drop-launch" an application by selecting several files, then dragging and dropping them onto the application icon.

On XVT/XM, printp is always set to FALSE.xvt_app_get_files_count returns information about files passed as command-line arguments on the argv that you pass to xvt_app_create.

See Also

E_CREATE
xvt_app_create
xvt_app_destroy
xvt_app_get_file
xvt_app_set_file_processed

The "Files" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for xvt_app_get_file.

xvt_app_process_pending_events

Process Pending Events

Summary

```
void xvt_app_process_pending_events(void)
```

Description

This function causes XVT to empty the event queue of all pending events and to dispatch them to the appropriate event handlers. After all events have been dispatched and the functions that received them have returned, `xvt_app_process_pending_events` returns.

Since you can only call `xvt_app_process_pending_events` from an event handler, it's possible to make a recursive call to that function. You should plan carefully for this by, among other things, restricting the use of global variables. Especially, make sure that the recursive call won't call `xvt_app_process_pending_events` again.

Calling `xvt_app_process_pending_events` during an otherwise unbroken operation (such as loading a file) allows user input to be processed. On non-preemptive multitasking window systems (Mac and MS-Windows), this also gives other applications a chance to execute.

Therefore, call this function often (every 1/10th second suffices) during long operations such as reading or writing a file, or when performing a time-consuming computation such as sorting. During that operation you can put up a dialog box that offers the user the opportunity to Cancel. For the dialog to function, you must call `xvt_app_process_pending_events`.

You can also use `xvt_app_process_pending_events` to implement a crude sort of multi-threading. By extending the above concepts, you can set up a system whereby several "background" tasks can be running simultaneously, timesliced in a non-preemptive fashion. Be careful, as it is quite tricky to keep the user input allowed during `xvt_app_process_pending_events` calls from interfering with the running tasks. This code shows a skeleton example of such a setup:

```
while (running) {
    xvt_app_process_pending_events();
    task1_timeslice();
    task2_timeslice();
    ...
}
```

Parameter Validity and Error Conditions

This function should not be called during `E_UPDATE`.

See Also

The "Events" chapter in the *XVT Portability Toolkit Guide*

xvt_app_set_file_processed

Indicate that File has been Processed

Summary

```
void xvt_app_set_file_processed(void)
```

Description

This function tells XVT that a file the user selected prior to starting the XVT application has been processed. This function implicitly refers to the file returned by the previous call to `xvt_app_get_file`.

Implementation Note

This function doesn't have any effect on platforms other than XVT/Mac. On XVT/Mac, `xvt_app_set_file_processed` indicates that a selected file has been processed by deselecting the file's icon.

See Also

```
xvt_app_get_file
xvt_app_get_files_count
```

The "Files" chapter in the *XVT Portability Toolkit Guide*.

Example

See the example for `xvt_app_get_file`.

xvt_cb_*

Clipboard Objects

```
xvt_cb_alloc_data
xvt_cb_close
xvt_cb_free_data
xvt_cb_get_data
xvt_cb_has_format
xvt_cb_open
xvt_cb_put_data
```

xvt_cb_alloc_data

Allocate Memory for Clipboard Data

Summary

```
char *xvt_cb_alloc_data(long size)long size
```

The number of bytes to allocate for memory.

Description

This function allocates a block of memory in which to place a data structure to be put onto the clipboard in `CB_TEXT` or `CB_APPL` format. It is analogous to the standard C function `malloc`. After setting up the data structure, call `xvt_cb_put_data` to put it onto the clipboard. Then call `xvt_cb_free_data` to free the block.

Don't allocate memory with `xvt_cb_alloc_data` for `PICTURE`s, because `xvt_cb_put_data` takes that format directly.

You have to use this function to allocate `CB_TEXT` and `CB_APPL` clipboard memory--you can't use `malloc`, `xvt_gmem_alloc`, or any other method.

Return Value

A pointer to the block is returned if successful; `NULL` is returned if memory can't be allocated.

See Also

```
CB_* Values for CB_FORMAT
xvt_cb_free_data
xvt_cb_put_data
```

The "Clipboard" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_cb_put_data`.

xvt_cb_close

Close the Clipboard

Summary

```
BOOLEAN xvt_cb_close(void)
```

Description

This function closes the clipboard that was previously opened with `xvt_cb_open`. After putting or retrieving data from the clipboard, it should be closed immediately to avoid interference with other applications. Future access to the clipboard can be gained via another call to `xvt_cb_open`.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful (on error).

See Also

`xvt_cb_open`

The "Clipboard" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_cb_get_data` and `xvt_pict_create`.

xvt_cb_free_data

Free Memory for Clipboard Data

Summary

```
void xvt_cb_free_data(void)
```

Description

This function frees the clipboard memory previously allocated by `xvt_cb_alloc_data`. Your application must call

xvt_cb_free_data; it is not called automatically by xvt_cb_put_data.

See Also

xvt_cb_alloc_data
xvt_cb_put_data

The "Clipboard" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for xvt_cb_put_data.

xvt_cb_get_data

Get Data from Clipboard

Summary

```
char *xvt_cb_get_data(CB_FORMAT cbfmt, char *name,
    long *sizep)
```

`CB_FORMAT cbfmt`

Format requested.

`char *name`

Application-determined format name (if CB_APPL). This argument is ignored (and can be NULL) when the format is CB_TEXT or CB_PICT.

`long *sizep`

Number of bytes retrieved from the clipboard.

Description

This function gets data from the clipboard, in whatever format you request. The data always comes in as a sequence of bytes whose length is returned through sizep.

For CB_APPL data, name specifies the application-determined format to be gotten. The interpretation of the resulting bytes is up to the application. The size returned through sizep is guaranteed to be at least the size of the data put onto the clipboard via xvt_cb_put_data. Use the returned size to allocate a buffer for making a copy of the data before closing the clipboard or getting more data. Because the clipboard buffer may be larger than the data,

the application must be able to determine the size of the data based on the format name and the buffer size (e.g., use a known termination byte value, divide the buffer size by record size, etc.).

For `CB_TEXT` data, the bytes may be broken into text lines, each of which ends with an end-of-line sequence equal to `EOL_SEQ` (which is platform-specific). Do not assume that the last line is terminated by an end-of-line sequence, although it might be. The data is not terminated by a `NULL` byte. Use the size returned through the `sizep` argument to determine the end of the data. For portability, use `xvt_str_find_eol` to break the data into lines.

For `CB_PICT` data, convert the bytes to a `PICTURE` by calling `xvt_pict_create`. The format of the unconverted sequence is in the same (undefined) format as that returned by `xvt_pict_lock`.

Return Value

A pointer to the data if successful; `NULL` if the desired format is unavailable or if an error occurred. The returned address points to data buffers internal to XVT. Do not attempt to free this pointer by calling any function, including `free` or `xvt_mem_free`. The returned pointer is only valid until the next call to `xvt_cb_get_data` or until you call `xvt_cb_close`.

Therefore, make sure to copy the data from this pointer using whatever means is appropriate before you close the clipboard or get additional data.

See Also

```
CB_* Values for CB_FORMAT
xvt_cb_close
xvt_cb_put_data
xvt_pict_create
xvt_pict_lock
```

The "Clipboard" chapter in the *XVT Portability Toolkit Guide*

Example

```
char *p;if (xvt_cb_open(FALSE) == TRUE)
{
    p = xvt_cb_get_data(format.fmt, format.name,
          &format.size );
    if (p == (char *) NULL)
        xvt_dm_post_note(
        "Clipboard contains no data in chosen format");
    else
        switch(format.fmt) {
        case CB_TEXT:
            ...
            break;
        case CB_APPL:
            ...
            break;
        case CB_PICT:
            ...
            break;
        }
    xvt_cb_close();
}
```

xvt_cb_has_format

Test If Format is on Clipboard

Summary

```
BOOLEAN xvt_cb_has_format(CB_FORMAT fmt, char
    *name)
```

`CB_FORMAT fmt`

Format of the data to be tested.

`char *name`

The application-determined format name (if `CB_APPL`). When the format is other than `CB_APPL`, the `name` argument should be `NULL`.

Description

This function tests to see if data in a particular format is on the clipboard. When the format is `CB_APPL`, `name` is the name of the format as determined by the application. It must be a `NULL`-terminated character string of 4 characters or less.

To call `xvt_cb_has_format`, the clipboard does *not* need to be opened with `xvt_cb_open`.

Return Value

`TRUE` if the data of the requested format is present; `FALSE` otherwise.

See Also

`CB_* Values for CB_FORMAT`
`xvt_cb_open`

The "Clipboard" chapter in the *XVT Portability Toolkit Guide*

Example

```
/* enable paste if clipboard has data */
BOOLEAN paste_enable = TRUE;
if (xvt_cb_has_format(CB_APPL, APPL_FORMAT))
    paste_fmt = CB_APPL;
else if (xvt_cb_has_format(CB_PICT, NULL))
    paste_fmt = CB_PICT;
else if (xvt_cb_has_format(CB_TEXT, NULL))
    paste_fmt = CB_TEXT;
else
    paste_enable = FALSE;
```

xvt_cb_open

Open Clipboard for Reading or Writing

Summary

`BOOLEAN xvt_cb_open(BOOLEAN writing)BOOLEAN writing`

If `TRUE`, clipboard is open for writing, or if `FALSE`, for reading.

Description

This function opens the clipboard for access. You must call it before calling any other clipboard function other than `xvt_cb_has_format`. As soon as you have gotten data from or put data onto the clipboard, close it with `xvt_cb_close`.

`writing` should be `TRUE` if you will be calling `xvt_cb_put_data`, and `FALSE` if you will be calling `xvt_cb_get_data`.

If you are putting multiple formats onto the clipboard, or taking multiple formats off of the clipboard, then do so within a single `xvt_cb_open`/`xvt_cb_close pair`.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful (on error).

See Also

```
xvt_cb_close
xvt_cb_get_data
xvt_cb_has_format
xvt_cb_put_data
```

The "Clipboard" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_cb_get_data` and `xvt_pict_create`.

xvt_cb_put_data

Put Data on Clipboard

Summary

```
BOOLEAN xvt_cb_put_data(CB_FORMAT cbfmt, char *name,
    long size, PICTURE pic)
```

`CB_FORMAT cbfmt`

Format of the data to be put on the clipboard.

`char *name`

Format name (if `CB_APPL`). For `CB_TEXT` or `CB_PICT` formats, if `name` is not used, `name` should be `NULL`.

`long size`

Number of bytes to put on clipboard. A `size` is required for formats `CB_TEXT` and `CB_APPL`, but not for `CB_PICT` (in which case `size` can be zero).

`PICTURE pic`

Picture (if `CB_PICT`). For `CB_TEXT` or `CB_APPL` formats, if `pic` is not used, `pic` should be `(PICTURE)0`.

Description

This function puts data on the clipboard. For the format `CB_PICT`, you supply an object of type `PICTURE` as the `pic` argument. For formats `CB_TEXT` and `CB_APPL`, the data is automatically taken from

the global memory allocated earlier with a call to `xvt_cb_alloc_data`.

After a call to `xvt_cb_put_data`, any block allocated via `xvt_cb_alloc_data` is no longer valid and should not be accessed. In addition, once you have put the data on the clipboard, free any block allocated via `xvt_cb_alloc_data` by calling `xvt_cb_free_data`.

If you are putting multiple data formats on the clipboard at one time, put *all* formats on the clipboard within a single `xvt_cb_open`/ `xvt_cb_close` pair.

Return Value

`TRUE` if successful; `FALSE` if unsuccessful (on error).

See Also

```
CB_* Values for CB_FORMAT
xvt_cb_alloc_data
xvt_cb_close
xvt_cb_open
```

The "Clipboard" chapter in the *XVT Portability Toolkit Guide*

Example

This function assumes that `xvt_cb_open` has been called, and that `xvt_cb_close` will be called later.

```
/*
    Function to put some text onto the clipboard.
    Although the text is static here, the algorithm to
    put it onto the clipboard is general, and can be
    used for any collection of text lines.
*/
void put_text(void);
static void put_text()
{
    char *p;
    static char *text[] = {
        "The quick brown fox",
        "jumped over the",
        "lazy dogs.",
        NULL
    };int i, eol_len;
    long size;eol_len = strlen(EOL_SEQ);
    size = 0;
    for (i = 0; text[i] != NULL; i++)
        size += strlen(text[i]) + eol_len;
    p = xvt_cb_alloc_data(size);
    if (p != (char *) NULL) {
        p[0] = '0';
        for (i = 0; text[i] != NULL; i++) {
            strcat(p, text[i]);
            strcat(p, EOL_SEQ);
        }
        xvt_cb_put_data(CB_TEXT, NULL, size, NULL);
        xvt_cb_free_data();
    }
}
```

xvt_ctl_*

Control Functions

```
xvt_ctl_check_radio_button
xvt_ctl_create
xvt_ctl_create_def
xvt_ctl_get_color_component
xvt_ctl_get_colors
xvt_ctl_get_font
xvt_ctl_get_id
xvt_ctl_get_native_color_component
xvt_ctl_get_native_colors
xvt_ctl_get_text_sel
xvt_ctl_is_checked
xvt_ctl_set_checked
xvt_ctl_set_color_component
xvt_ctl_set_colors
xvt_ctl_set_font
xvt_ctl_set_text_sel
xvt_ctl_unset_color_component
```

xvt_ctl_check_radio_button

Check a Radio Button in a Window

Summary

```
void xvt_ctl_check_radio_button(WINDOW win,
    WINDOW *ctls, int nctls)
```

`WINDOW win`

Radio button to set.

`WINDOW *ctls`

Array of grouped radio buttons.

`int nctls`

Count of the items in `ctls`.

Description

This function highlights exactly one of a group of radio button controls. `ctls` points to an array of the related radio button `WINDOW`s

(as returned by a control creation function or `xvt_win_get_ctl`) that are to be treated as a group. `win` is the `WINDOW` of the radio button to be checked. Except for `win`, all of the controls in `ctls` are unchecked. `nctls` is a count of the items in `ctls`.

Note: The group of radio buttons affected by this function is *entirely* independent of the keyboard navigation groups established by the radiobutton `GROUP` option in URL.

Parameter Validity and Error Conditions

XVT issues an error if each element of `ctls` and `win` are not of type `WC_RADIOBUTTON`.

See Also

`WINDOW`
`XVT_COLOR_TYPE`
`xvt_ctl_create`
`xvt_ctl_create_def`
`xvt_ctl_is_checked`
`xvt_ctl_set_checked`
`xvt_dlg_create_def`
`xvt_dlg_create_res`
`xvt_win_create`
`xvt_win_create_def`
`xvt_win_create_res`
`xvt_win_get_ctl`
`radiobutton Control` URL statement

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

```
static void check_radio(WINDOW win, int id)
{
    WINDOW ctls[3];ctls[0] = xvt_win_get_ctl(win,
RADIO_1);
    ctls[1] = xvt_win_get_ctl(win, RADIO_2);
    ctls[2] = xvt_win_get_ctl(win, RADIO_3);
    xvt_ctl_check_radio_button(
        xvt_win_get_ctl(win, id), ctls, 3);
}
```

xvt_ctl_create

Create a Control in a Window

Summary

```
WINDOW xvt_ctl_create(WIN_TYPE wtype, RCT *rct_p,
    char *title, WINDOW parent_win, long ctl_flags,
    long app_data, int ctrl_id)
```

`WIN_TYPE wtype`

Determines the type of control to be created and should be one of the `WC_*` constants. For the list of possible types, see `WIN_TYPE`.

`RCT *rct_p`

Defines the bounding rectangle for the control in terms of the parent window's client area. This parameter must not be `NULL` and must point to a valid rectangle. If you are creating combo controls of type `WC_LISTEDIT` or `WC_LISTBUTTON`, then set this rectangle to include the area that will be occupied by the dropped-down list when the user activates it. XVT ignores the height of the rectangle when the list is not activated by the user. Instead, XVT displays the edit field or button using the current font for the control and default height of the platform. Because the users activate the dropped-down list only when they want to see the list, it doesn't matter if other controls overlap with the bounding rectangle.

`char *title`

Used to set the text of the control in the fashion appropriate for the control. For `WC_PUSHBUTTON`, `WC_RADIOBUTTON`, `WC_CHECKBOX`, `WC_GROUPBOX`, `WC_TEXT`, `WC_EDIT`, and `WC_LISTEDIT` controls, it has the effect of setting the title of the control. It has no effect for other controls. If it is `NULL`, the control will have no title.

`WINDOW parent_win`

Window in which the control should be placed. It must be a valid window of type `W_*`.

`long ctl_flags`

Controls the attributes and the initial state of a control. The applicable control flags vary among controls, and a complete

table listing the valid control flags for each control can be found in Window/Dialog/Control Creation Function Parameters.

`long app_data`

Contains any application data that you wish to attach to a control. Typically, this will be a pointer to some structure allocated from the heap, cast into a `long` such that later your application can retrieve the structure and look at it.

`int ctrl_id`

An ID number for the control relative to its parent window. When XVT sends an `E_CONTROL` event to the event handler for the window containing the control, it sets the `v.ctl.id` field of the `EVENT` structure to the ID of the control that was activated. A control ID-parent window combination is a way of uniquely identifying a control independent of its window handle. Keep in mind that it is not necessary to use control IDs, but if you choose to use them, then all of the IDs for the controls in a window must be unique. You can also call the `xvt_win_get_ctl` function with an ID and parent window to retrieve the `WINDOW` for a control.

Description

This function adds a control to `parent_win`. This function does *not* add controls to dialogs.

Note: You cannot create icon controls with `xvt_ctl_create`, because icon controls require that `xvt_ctl_create` have an extra parameter to specify an icon resource ID. Use `xvt_ctl_create_def` instead.

Return Value

A `WINDOW` if successful; `NULL_WIN` if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if your application does not meet the following conditions for parameters passed to `xvt_ctl_create`:

- `wtype` must be one of the `WC_*` control types.
- `rct_p` must point to a valid rectangle.
- `parent_win` must be a valid XVT window of type `W_*`. You cannot create a control in a dialog, a print window, a task window, or a screen window. (The exception to this is if your application is running with XVT/Win32, and has set the non-portable attribute `ATTR_WIN_PM_DRAWABLE_TWIN`. In that case, it can create controls in the task window.)

- `ctl_flags` must be appropriate for the control you want to create, as defined in Window/Dialog/Control Creation Function Parameters.

See Also

```
CTL_FLAG_* Options
E_CONTROL
EVENT
RCT
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
WIN_TYPE
xvt_ctl_create_def
xvt_dlg_create_def
xvt_vobj_get_data
xvt_vobj_get_title
xvt_vobj_set_data
xvt_vobj_set_title
xvt_win_create_def
xvt_win_create_res
xvt_win_get_ctl
Window/Dialog/Control Creation Function Parameters
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

```
#define BUTTON_ID 101
    .
    :
RCT rect;
WINDOW button;
xvt_rect_set(&rect, BUTTON_X, BUTTON_Y, BUTTON_X + 100,
    BUTTON_Y + (int) xvt_vobj_get_attr(window,
    ATTR_CTL_BUTTON_HEIGHT));
button = xvt_ctl_create(WC_PUSHBUTTON, &rect,
    "Push Me", window, 0L, 0L, BUTTON_ID);
```

xvt_ctl_create_def

Create a Control from a Data Structure

Summary

```
WINDOW xvt_ctl_create_def(WIN_DEF *win_def_p,
    WINDOW parent_win, long app_data)

WIN_DEF *win_def_p
```

Pointer to a WIN_DEF structure describing the control to be created.

WINDOW parent_win

Window in which the control is to be placed. It must be a valid window of type W_*.

long app_data

Contains any application data that you wish to attach to a control. Typically, this will be a pointer to some structure allocated from the heap cast into a long, such that later your application can retrieve the structure and look at it.

Description

This function adds a control, based on a data structure definition, to parent_win. This function does not add controls to dialogs.

The win_def_p parameter points to a WIN_DEF structure describing the control to be created. When filling in the WIN_DEF, keep in mind the following:

- win_def_p->wtype determines the type of control to be created. The field should contain one of the WC_* constants. For the list of possible types, see WIN_TYPE.
- win_def_p->rct defines the bounding rectangle for the control in terms of the parent window's client area. If you are creating combo controls of type WC_LISTEDIT or WC_LISTBUTTON, then set this rectangle to include the area that will be occupied by the dropped-down list when the user activates it. XVT will ignore the height of the rectangle when the list is not activated by the user. Instead, XVT will display the edit field or button using the font set for the control or default height of the platform. Because the users activate the dropped-down list only when they want to see the list, it doesn't matter if other controls overlap with the bounding rectangle.
- win_def_p->text is used to set the text of the control in the fashion appropriate for the control. For WC_PUSHBUTTON, WC_RADIOBUTTON, WC_CHECKBOX, WC_GROUPBOX, WC_TEXT, WC_EDIT, and WC_LISTEDIT controls, it sets the label of the control. It has no effect for other controls. If it is NULL, the control will have no title.
- win_def_p->units is one of U_PIXELS, U_CHARS, or U_SEMICHARS, and specifies the units used to measure the bounding rectangle win_def_p->rct.

- `win_def_p->ctlcolors` points the array of `XVT_COLOR_COMPONENT` structures that define the control's colors. If it is `NULL`, the control uses the default colors for the container control colors. The last element of the `XVT_COLOR_COMPONENT` array should have an `XVT_COLOR_TYPE` of `XVT_COLOR_NULL` to indicate the end of the array.
- `win_def_p->v.ctl.ctrl_id` is an ID number for the control. When XVT sends an `E_CONTROL` event to the event handler for the window containing the control, it sets the `v.ctl.id` field of the `EVENT` structure to the ID of the control that was activated. A control ID-parent window combination is a way of uniquely identifying a control independent of its window handle. Keep in mind that it is not necessary to use control IDs, but if you choose to use them, then all of the IDs for the controls in a window must be unique. You can also call the `xvt_win_get_ctl` function with an ID and parent window to retrieve the `WINDOW` for a control.
- `win_def_p->v.ctl.icon_id` contains the resource ID for an icon resource to be used for controls of type `WC_ICON`. The icon resource ID refers to an icon resource that is declared non-portably in your resource file. For more information, see the *XVT Platform-Specific books*.
- `win_def_p->v.ctl.flags` controls the attributes and the initial state of a control. The applicable control flags vary among controls, and a complete table listing the valid control flags for each control can be found in Window/Dialog/ Control Creation Function Parameters.
- `win_def__p->v.ctl.font_id` is the `XVT_FNTID` that defines the font used in the control.

Return Value

A `WINDOW` if successful; `NULL_WIN` if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if your application does not meet the following conditions for parameters passed to `xvt_ctl_create`:

- `win_def_p->wtype` must be one of the `WC_*` control types.
- `parent_win` must be a valid XVT window of type `W_*`. You cannot create a control in a dialog, a print window, a task window, or a screen window. (The exception to this is if your application is running with XVT/Win32, and has set the non-

portable attribute ATTR_WIN_PM_DRAWABLE_TWIN. In that case, it can create controls in the task window.)

- win_def_p->v.ctl.flags must be appropriate for the control you want to create, as defined in Window/Dialog/Control Creation Function Parameters.
- win_def_p->v.ctl.icon_id must specify a valid icon resource ID if the control to be created is of type WC_ICON.
- win_def_p->units must specify U_PIXELS, U_CHARS, or U_SEMICHARS.
- win_def_p->v.ctl.font_id must be either NULL_FNTID or a valid logical font.
- win_def_p->ctlcolors must either be NULL or a valid array of XVT_COLOR_COMPONENT structures.

See Also

CTL_FLAG_* Options
W_*, WC_*, WD_*, Values for WIN_TYPE
WIN_DEF
WINDOW
XVT_COLOR_COMPONENT
xvt_ctl_create
xvt_dlg_create_def
xvt_font_*
xvt_vobj_get_data
xvt_vobj_get_title
xvt_vobj_set_data
xvt_vobj_set_title
xvt_win_create_def
xvt_win_create_res
xvt_win_get_ctl
Window/Dialog/Control Creation Function Parameters

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

This code is nearly equivalent to the example for xvt_ctl_create:

```
#define BUTTON_ID 101
static WIN_DEF button_def = {
    WC_PUSHBUTTON,
    {BUTTON_Y, BUTTON_X, BUTTON_Y + 30, BUTTON_X +
    100},
    "Push Me",
    U_PIXELS,
    NULL
};
    .
    :
WINDOW button;
button_def.v.ctl.ctrl_id = BUTTON_ID;
button_def.v.ctl.flags = 0L;
button_def.v.ctl.font_id = NULL_FNTID;
button = xvt_ctl_create_def(&button_def, window, 0L);
```

xvt_ctl_get_color_component

Get the Control Color for a Color Type From a Single Control

Summary

COLOR xvt_ctl_get_color_component(WINDOW ctl_win,
XVT_COLOR_TYPE ctype)

WINDOW ctl_win

WINDOW ID of the control.

XVT_COLOR_TYPE ctype

Control component to get the color for.

Description

This function gets the control color used in a single control for a single control component. This color is either the control color set in the WIN_DEF structure during the control's creation, or the control colors set by xvt_ctl_set_colors or xvt_ctl_set_color_component.

Return Value

INVALID_COLOR if no color is set for the requested component of this control or if an error occurs, otherwise the COLOR value of the control component.

Parameter and Validity Conditions

XVT issues an error if win is not a valid control WINDOW or ctype is not a valid XVT_COLOR_TYPE.

See Also

WIN_DEF
XVT_COLOR_TYPE
XVT_COLOR_COMPONENT
xvt_ctl_set_colors
xvt_ctl_set_color_component
xvt_ctl_unset_color_component
xvt_ctl_get_colors
xvt_win_set_ctl_colors
xvt_win_get_ctl_colors
ATTR_APP_CTL_COLORS
COLOR

xvt_ctl_get_colors

Get the Colors From a Single Control

Summary

```
XVT_COLOR_COMPONENT *xvt_ctl_get_colors(WINDOW
    ctl_win)

window ctl_win
```

WINDOW ID of control.

Description

This function provides the application with a copy of the colors used in a single control. These colors are either the control colors set in the WIN_DEF structure during the control's creation, or the control colors set by xvt_ctl_set_colors.

Return Value

NULL if no colors are set for the control or if an error occurs; a pointer to an array of XVT_COLOR_COMPONENT structures otherwise. The application owns this array: when finished with it, use xvt_mem_free to destroy it.

Parameter and Validity Conditions

ctl_win must be a valid control.

See Also

```
WIN_DEF
WINDOW
XVT_COLOR_COMPONENT
xvt_ctl_get_native_colors
xvt_ctl_set_colors
xvt_mem_free
xvt_win_get_ctl_colors
xvt_win_set_ctl_colors
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

xvt_ctl_get_font

Get the Logical Font from a Single Control

Summary

```
XVT_FNTID xvt_ctl_get_font(WINDOW ctl_win)

WINDOW ctl_win
```

`WINDOW` ID of control.

Description

This function returns a copy of the logical font used in a single control. The logical font returned reflects the latest control font setting for the specified control. This font can either be the control font set in the `WIN_DEF` structure during the control's creation, or a control font set by `xvt_ctl_set_font`.

Return Value

`NULL_FNTID` if an error occurs; a copy of the logical font rendering the control if no error occurs. The application owns this logical font and when finished with it, it must be destroyed with `xvt_font_destroy`.

Parameter and Validity Conditions

`ctl_win` must be a valid control.

See Also

```
ATTR_APP_CTL_FONT_RID
WIN_DEF
WINDOW
XVT_FNTID
xvt_ctl_set_font
xvt_font_destroy
xvt_win_get_ctl_font
xvt_win_set_ctl_font
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

xvt_ctl_get_id

Get a Control ID

Summary

```
int xvt_ctl_get_id(WINDOW ctl_win)

WINDOW ctrl_win
```

`WINDOW` ID of control.

Description

This function returns a control ID, given the control's `WINDOW`. This function is the reverse of `xvt_win_get_ctl` (which returns a `WINDOW` ID given a control ID and the control parent window).

Return Value

Control ID of the window; zero for controls created without a control ID (for example, `xvt_ctl_create` called with a control ID of zero); zero if an error is detected.

Note: This function does not return any information about the container of the control. Use `xvt_vobj_get_parent` to obtain this information.

Parameter and Validity Conditions

`ctl_win` must be a valid control.

See Also

```
WINDOW
xvt_ctl_create
xvt_ctl_create_def
xvt_vobj_get_parent
xvt_win_get_ctl
```

xvt_ctl_get_native_color_component

Get the Native Control Color for a Control Type From a Single Control Component

Summary

COLOR xvt_ctl_get_native_color_component(WIN_TYPE type, XVT_COLOR_TYPE ctype)

WIN_TYPE type

WINDOW type of a control.

XVT_COLOR_TYPE ctype

Control component to get the color for.

Description

This function gets the native control color used in a single control for a single control component. This color is the control color set by the native window system when no other control color settings are used.

Return Value

INVALID_COLOR if no color is defined for the requested component of this control or if an error occurs, otherwise the native COLOR value of the control component.

Parameter and Validity Conditions

XVT issues an error if type is not a valid control type (WC_*) or ctype is not a valid XVT_COLOR_TYPE.

See Also

WIN_DEF
XVT_COLOR_TYPE
XVT_COLOR_COMPONENT
xvt_ctl_set_colors
xvt_ctl_set_color_component
xvt_ctl_unset_color_component
xvt_ctl_get_colors
xvt_win_set_ctl_colors
xvt_win_get_ctl_colors
ATTR_APP_CTL_COLORS
COLOR

xvt_ctl_get_native_colors

Get the Native Default Colors for a Control Type

Summary

```
XVT_COLOR_COMPONENT *xvt_ctl_get_native_colors(WIN_TYPE
    type)

WIN_TYPE type
```

The WC_* control type.

Description

This function provides the application with a copy of the native colors used when creating a control of the type specified in type. These are the colors that would be used when creating a control when no other colors have been specified in the WIN_DEF structure or assigned to the window with xvt_ctl_set_colors or through the attribute ATTR_APP_CTL_COLORS.

Return Value

NULL if an invalid type is passed or if an error occurs; a pointer to an array of XVT_COLOR_COMPONENT structures otherwise. The application owns this array: when finished with it, use xvt_mem_free to destroy it.

Parameter and Validity Conditions

WIN_TYPE must be a WC_* control type.

See Also

WIN_DEF
WIN_TYPE
XVT_COLOR_COMPONENT
xvt_ctl_set_colors
xvt_ctl_get_colors
xvt_mem_free
xvt_win_get_ctl_colors
xvt_win_set_ctl_colors

xvt_ctl_get_text_sel

Get Text Selection in Edit Control

Summary

```
void xvt_ctl_get_text_sel(WINDOW win, int *first,
    int *last)
```

`WINDOW win`

`WC_EDIT` or `WC_LISTEDIT` control to be queried.

`int *first`

Position of the left-most selected character.

`int *last`

Position of the last selected character plus one.

Description

This function returns information about the selection or insertion point in the `WC_EDIT` or `WC_LISTEDIT` control specified by `win`.

`first` returns the position of the left-most selected character (numbered starting at zero). `last` returns the position of the last selected character plus one. For example, if characters one through five (numbered from zero) are selected, `first` is set to one and `last` to six.

If nothing is selected, `xvt_ctl_get_text_sel` returns the position of the insertion point. In this case, `first` and `last` return the same value.

If your application is attempting to validate characters as they are typed in and the user types an invalid character, your application should retrieve the insertion position so that it can restore it (with `xvt_ctl_set_text_sel`) after calling `xvt_vobj_set_title` to remove the erroneous character.

Return Value

None.

Parameter Validity and Error Conditions

XVT issues an error if your application does not meet the following conditons for parameters passed to `xvt_ctl_get_text_sel`:

- `win` window must be of type `WC_EDIT` or `WC_LISTEDIT`

- `first` and `last` must be non-`NULL` pointers to integers

See Also

`WINDOW`
`W_*, WC_*, WD_*, Values for WIN_TYPE`
`xvt_ctl_set_text_sel`
`xvt_vobj_set_title`

The "Windows" and the "Controls" chapters in the *XVT Portability Toolkit Guide*

xvt_ctl_is_checked

Get Checked State of Control

Summary

`BOOLEAN xvt_ctl_is_checked(WINDOW win)WINDOW win`

`WC_RADIOBUTTON` or `WC_CHECKBOX` control to be queried.

Description

Calling `xvt_ctl_is_checked` will tell your application the state of a radio button or check box control. Recall that radio buttons are controls of type `WC_RADIOBUTTON` and check boxes are controls of type `WC_CHECKBOX`. Both of these control types have a visual indication of whether they are "checked," meaning that the user has selected them in some fashion.

Upon creation, radio buttons and check boxes can be initially checked using the`CTL_FLAG_CHECKED` control flag. You can change the "checked" state of a radio button by calling `xvt_ctl_check_radio_button`. You can change the "checked" state of a check box by calling `xvt_ctl_set_checked`.

Return Value

`TRUE` if the `WC_CHECKBOX` or `WC_RADIOBUTTON` control specified by `win` is checked; `FALSE` if the control is not checked.

Parameter Validity and Error Conditions

XVT issues an error if `win` is not of type `WC_CHECKBOX` or `WC_RADIOBUTTON`.

See Also

CTL_FLAG_* Options_(CTL_FLAG_CHECKED)
W_*, WC_*, WD_*, Values for WIN_TYPE (WC_CHECKBOX, WC_RADIOBUTTON)
WINDOW
xvt_ctl_check_radio_button
xvt_ctl_create
xvt_ctl_create_def
xvt_ctl_is_checked
xvt_ctl_set_checked
xvt_dlg_create_def
xvt_dlg_create_res
xvt_win_create
xvt_win_create_def
xvt_win_create_res

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for xvt_ctl_set_checked.

xvt_ctl_set_checked

Check a Check Box Control

Summary

```
void xvt_ctl_set_checked(WINDOW win, BOOLEAN check)
```

WINDOW win

Check box control to be set or reset.

BOOLEAN check

Set or reset a check mark.

Description

This function sets or resets a check mark in a check box control. There are two common situations when you would do this. First, when a window or dialog containing checked boxes is first initialized, you set all of the check box controls to their initial states. Second, when the user operates a check box control, your application receives an E_CONTROL for that check box, in which case your application normally toggles the "checked" state of that control.

When toggling the "checked" state of the control, `xvt_ctl_is_checked` can be used to get the current state. Do not use `xvt_ctl_set_checked` for radio buttons. Instead, use `xvt_ctl_check_radio_button`.

Parameter Validity and Error Conditions

XVT issues an error if the control is not of type `WC_CHECKBOX`.

See Also

```
CTL_FLAG_* Options_(CTL_FLAG_CHECKED)
E_CONTROL
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
xvt_ctl_create
xvt_ctl_create_def
xvt_ctl_check_radio_button
xvt_ctl_is_checked
xvt_dlg_create_def
xvt_dlg_create_res
xvt_win_create
xvt_win_create_def
xvt_win_create_res
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

Example

```
void flip_checkbox (WINDOW win)
{
    xvt_ctl_set_checked(win, !xvt_ctl_is_checked(win));
}
```

xvt_ctl_set_color_component

Sets the Control Color for a Color Type Used by a Single Control

Summary

void xvt_ctl_set_color_component(WINDOW ctl_win, XVT_COLOR_TYPE ctype, COLOR
color)

WINDOW ctl_win

WINDOW ID of the control.

XVT_COLOR_TYPE ctype

Control component to set the color for.

COLOR color

Color to assign.

Description

This function sets the control color used in a single control for a single control component. A call to this function overrides any previous control color setting only for the specified XVT_COLOR_TYPE. This includes the default colors set in the WIN_DEF structure during the control's creation, any previous call to xvt_ctl_set_color_component, xvt_ctl_set_colors, or xvt_win_set_ctl_colors for this component. All other colors used by the designated control not specified by a call to this function are unaffected.

Parameter and Validity Conditions

XVT issues an error if win is not a valid control WINDOW or ctype is not a valid XVT_COLOR_TYPE.

See Also

```
WIN_DEF
XVT_COLOR_TYPE
XVT_COLOR_COMPONENT
xvt_ctl_set_colors
xvt_ctl_set_color_component
xvt_ctl_unset_color_component
xvt_ctl_get_colors
xvt_win_set_ctl_colors
xvt_win_get_ctl_colors
ATTR_APP_CTL_COLORS
COLOR
```

xvt_ctl_set_colors

Changes Control Colors Used by a Single Control

Summary

```
void xvt_ctl_set_colors(WINDOW ctl_win,
    XVT_COLOR_COMPONENT *colors,
    XVT_COLOR_ACTION action)
```

`WINDOW ctl_win`

WINDOW ID of control.

`XVT_COLOR_COMPONENT* colors`

Colors to set or unset.

`XVT_COLOR_ACTION action`

Either sets or unsets the colors.

Description

This function sets (or unsets) the control colors used in a single control. A call to this function overrides any previous control color settings *only* for the specified `XVT_COLOR_COMPONENT`s in the `colors` array. This includes the default colors set in the `WIN_DEF` structure during the control's creation, any previous call to `xvt_ctl_set_colors`, and any previous call to `xvt_win_set_ctl_colors`. All colors used by the designated control *not* specified in a call to this function are unaffected.

If `NULL` is passed to this function as the value of colors, the control reverts to the default colors for its container: i.e., the ACTION parameter is ignored; no colors are set. If the action parameter is `XVT_COLOR_ACTION_SET`, the control colors for the specified color components are set to the values in the colors parameter. If the action parameter is `XVT_COLOR_ACTION_UNSET`, the control colors for the specified color components are either inherited from the control's container, or the application owned `colors`, or the system default.

Parameter and Validity Conditions

XVT issues and error if `win` is not a valid control `WINDOW`.

See Also

`ATTR_APP_CTL_COLORS`
`WINDOW`
`XVT_COLOR_ACTION`
`XVT_COLOR_COMPONENT`
`xvt_ctl_get_colors`
`xvt_win_get_ctl_colors`
`xvt_win_set_ctl_colors`

The "Controls" chapter in the *XVT Portability Toolkit Guide*

xvt_ctl_set_font

Changes a Single Control's Logical Font

Summary

```
void xvt_ctl_set_font(WINDOW ctl_win,
    XVT_FNTID font_id)
```

`WINDOW ctl_win`

`WINDOW` ID of control.

`XVT_FNTID font_id`

Logical font.

Description

This function sets a single control's logical font. A call to this function overrides any previous font setting for the specified control (including the logical font the control received during creation, any previous call to `xvt_ctl_set_font`, and any previous call to `xvt_win_set_ctl_font`).

If `NULL_FNTID` is passed to this function, the control reverts to using either the default font of its container, or the application owned `font_id`, or the default control font.

Parameter and Validity Conditions

XVT issues an error if:

- `win` is not a valid control `WINDOW`
- `font_id` is neither `NULL_FNTID` nor a valid logical font

See Also

```
ATTR_APP_CTL_FONT_RID
NULL_FNTID
WINDOW
XVT_FNTID
xvt_ctl_get_font
xvt_win_get_ctl_font
xvt_win_set_ctl_font
```

The "Controls" chapter in the *XVT Portability Toolkit Guide*

xvt_ctl_set_text_sel

Select Text in Edit Control

Summary

```
void xvt_ctl_set_text_sel(WINDOW win, int first,
    int last)
```

`WINDOW win`

`WC_EDIT` or `WC_LISTEDIT` control.

`int first`

Position of the left-most character to be selected (numbered starting at zero).

`int last`

Position of the last character to be selected plus one.

Description

This function selects characters from `first` to `last` in the `WC_EDIT` or `WC_LISTEDIT` control specified by `win`. If the keyboard focus is not already in that control, this function forces it there.

`first` should be set to the position of the left-most character to be selected (numbered starting at zero). `last` should be set to the position of the last character to be selected plus one. For example, to select characters one through five (numbered from zero), set `first` to one and `last` to six.

To select everything, set `first` to zero and `last` to `SHRT_MAX`. To select nothing--that is, to position the insertion point--set `first` and `last` to the same value.

If your event handler detects an error in entered text when the user clicks OK, select the erroneous part. That way, the user can just type in the correction without having to first select it.

Parameter Validity and Error Conditions

XVT issues an error if any of the following parameter conditions are not met:

- `win` must be of type `WC_EDIT` or `WC_LISTEDIT`.
- `last` must be greater than or equal to the value of `first`. If `last` is less than first, XVT issues a warning, sets `last` equal to `first`, and then proceeds.

See Also

WINDOW
W_*, WC_*, WD_*, Values for WIN_TYPE_(WC_EDIT, WC_LISTEDIT)
xvt_ctl_get_text_sel

The "Windows" and the "Controls" chapters in the *XVT Portability Toolkit Guide*

xvt_ctl_unset_color_component

Unsets the Control Color for a Color Type Used by a Single Control

Summary

void xvt_ctl_set_color_component(WINDOW ctl_win, XVT_COLOR_TYPE ctype)

WINDOW ctl_win

WINDOW ID of the control.

XVT_COLOR_TYPE ctype

Control component to set the color for.

Description

This function unsets the control color used in a single control for a single control component. A call to this function overrides any previous control color setting only for the specified XVT_COLOR_TYPE. This includes the default colors set in the WIN_DEF structure during the control's creation, any previous call to xvt_ctl_set_color_component, or xvt_ctl_set_colors for this component. All other colors used by the designated control not specified by a call to this function are unaffected.

The control reverts to the default colors for its container. The control colors for the specified color component is either inherited from the control's container, or the application owned colors, or the system default.

Parameter and Validity Conditions

XVT issues an error if win is not a valid control WINDOW or ctype is not a valid XVT_COLOR_TYPE.

See Also

```
WIN_DEF
XVT_COLOR_TYPE
XVT_COLOR_COMPONENT
xvt_ctl_set_colors
xvt_ctl_set_color_component
xvt_ctl_unset_color_component
xvt_ctl_get_colors
xvt_win_set_ctl_colors
xvt_win_get_ctl_colors
ATTR_APP_CTL_COLORS
COLOR
```

xvt_cxo_*

Container Extension Object Functions

```
xvt_cxo_call_next
xvt_cxo_create
xvt_cxo_destroy
xvt_cxo_dispatch_msg
xvt_cxo_get_class_name
xvt_cxo_get_data
xvt_cxo_get_event_handler
xvt_cxo_get_event_mask
xvt_cxo_get_win
xvt_cxo_is_valid
xvt_cxo_set_data
xvt_cxo_set_event_handler
xvt_cxo_set_event_mask
```

xvt_cxo_call_next

Call the Next CXO in the Chain

Summary

```
long xvt_cxo_call_next( XVT_CXO cxo, EVENT * event_p )
```

`XVT_CXO cxo`

Specifies the CXO currently being called.

`EVENT * event_p`

Event to be passed on.

Description

This function is used to call the next CXO in the call chain, or to call the base event handler for a container if there are no CXO's left in the call chain. A CXO event handler must call this function if it wishes other CXO's and its container to receive events. If the CXO wants to consume an event, it can return immediately. Events can be pre-processed or post-processed by strategic placement of this function within a CXO's event handler. If a CXO wishes to post-process events, then `xvt_cxo_call_next` should be used at the top of the CXO's event handler. Otherwise this function should be called at the end for pre-processing.

Return Value

The return value is that of following CXO's or the base event handler.

Parameter and Validity Conditions

XVT issues an error if any of the following conditions are not met:

- The specified CXO is not valid.
- The event structure is not valid.

See Also

`EVENT`
`E_CXO`
`XVT_CXO`
`xvt_cxo_create`

Example

This example shows how to call the next CXO in the call chain from a CXO event handler.

```
long my_cxo_eh( XVT_CXO cxo, EVENT * ep )
{
    /* Pre-processing events */

    switch( ep -> type )
    {
    case E_CXO:
    ...
    return( OL );
    default:

    break;
    }
    /*
    Allow the rest of the CXO's and the base event handler
    to process event
    */
    xvt_cxo_call_next( cxo, ep );
}
```

xvt_cxo_create

Create a CXO

Summary

```
XVT_CXO xvt_cxo_create( WINDOW win, const long
    state_data, XVT_CXO_INSERTION where,
    XVT_CXO_EVENT_HANDLER cxo_eh, EVENT_MASK mask,
    char * class_name, long cxo_id )
```

`WINDOW win`

The container that the CXO will be attached to.

`const long state_data`

The initial state data for the CXO.

`XVT_CXO_INSERTION where`

Specifies where to add the CXO to the CXO chain for the container specified. A container can have more than one CXO attached to it. The new CXO can be added at the head or tail of a CXO chain

`XVT_CXO_EVENT_HANDLER cxo_eh`

The CXO event handler function.

`EVENT_MASK mask`

Specifies which events should be sent to the CXO handler. This is usually an OR'ed combination of any of the EM_* constants or EM_ALL.

`char * class_name`

The class name for the CXO. CXO's are uniquely identified by class_name and cxo_id. A CXO must have a class name but does not have to have an id value if the CXO class does not care if it is uniquely identified.

`long cxo_id`

The id value for the CXO. This can be zero if the class is all that is needed to identify a CXO.

Description

This function creates a CXO for the container specified. The container can be any window of type W_* except W_PIXMAP and W_PRINT. The container can also be SCREEN_WIN or TASK_WIN. If the container specified is SCREEN_WIN, then the CXO is a system level CXO. It will see all XVT events that are generated.

Return Value

An XVT_CXO if successful; NULL if unsuccessful (on error).

Parameter and Validity Conditions

XVT issues an error if any of the following conditions are not met:

- win must be a valid container.
- cxo_eh must be a valid pointer to a CXO event handler.
- class_name must point to a valid class name string.

See Also

EVENT_MASK
WINDOW
XVT_CXO
XVT_CXO_EVENT_HANDLER
XVT_CXO_POS_* Values for XVT_CXO_INSERTION

Example

This example shows how to create a CXO and to handle setting the state data for the XVT_CXO_CREATE_MSG.

```
{

    XVT_CXO new_cxo = (XVT_CXO)NULL;
    ...

    /* Creating a system level CXO */

    new_cxo = xvt_cxo_create( SCREEN_WIN, 0L,
    XVT_CXO_POS_FIRST, mñEM_CHAR,
    "MySystemCxo", 0L );

    ...

}

    /* The event handler for MySystemCxo */
    long my_cxo_eh( XVT_CXO cxo, EVENT * ep )
    {

    switch( ep -> type )
    {
    case E_CXO:
    switch( ep -> v.cxo.msg_id )
    {
    case XVT_CXO_CREATE_MSG:
    xvt_cxo_set_data( cxo, (long)ep -> v.cxo.ptr );
    break;
    ...
    }
    return( 0L );
    ...
    }

    /* Calling here because we are pre-processing events
    */ xvt_cxo_call_next( cxo, ep );

}
```

xvt_cxo_destroy

Destroy a CXO

Summary

```
void xvt_cxo_destroy( XVT_CXO cxo )
```

`XVT_CXO cxo`

CXO to destroy.

Description

This function destroys the CXO specified by `cxo`. The CXO's handler will receive the `XVT_CXO_DESTROY_MSG` message before the CXO is destroyed. A CXO that is still on the call stack will not receive this message until its recursion level has reached zero. CXO's that remain at application shutdown will be sent the `XVT_CXO_DESTROY_MSG` but can be destroyed by the user at any time before then.

Parameter and Validity Conditions

XVT issues an error if the following condition is not met:

`cxo` must be a valid CXO.

See Also

`XVT_CXO`
`XVT_CXO_*_MSG`
`xvt_win_get_cxo`
`xvt_cxo_get_data`

Example

This example shows how to destroy a CXO in a window.

```
XVT_CXO cxo = (XVT_CXO)NULL;
char* data;

switch( ep -> type )
{
case E_DESTROY:

    cxo = xvt_win_get_cxo( win, "MySystemCxo", OL );

    if( cxo )
    {
    data = (char *)xvt_cxo get_data( cxo );

    if( data )
    xvt_mem_free( data );
    xvt_cxo_destroy( cxo );
    }
    break;
    ...

}
```

xvt_cxo_dispatch_msg

Send a Message to a CXO

Summary

```
long xvt_cxo_dispatch_msg( XVT_CXO cxo, long msg_id,
    void * data )
```

`XVT_CXO cxo`

The CXO to which a message is being sent.

`long msg_id`

The message being sent to the CXO. The message will in most cases be defined by the implementor of a CXO and be unique to that CXO. Message id's can be any positive long value. Negative values are reserved by XVT.

`void * data`

The data associated with the message.

Description

Applications use this function to send a message to a CXO. The message sent will in most cases be unique to that CXO. The data is associated with the message being sent. The implementor of a CXO will have to specify his or her own messages and what, if any, data is associated with them. The message and the data are similar to those used in `E_USER` messages. This application function will package up the message and data into an `E_CXO` message that will be sent to the CXO's handler. It is important to note that your application cannot use `xvt_win_dispatch_event` to send an `E_CXO` message.

Return Value

The return from this function is specified by the implementor of the CXO on a per-message basis.

Parameter and Validity Conditions

XVT issues an error if the following condition is not met:

`cxo` must be a valid CXO.

See Also

E_CXO
E_USER
XVT_CXO
XVT_CXO_*_MSG
xvt_win_dispatch_event

Example

This example shows how to dispatch a message to a CXO and how it might be handled.

```
#define NO_MORE_CHARS 1 L
{

    ...
    xvt_cxo_dispatch_msg( cxo, NO_MORE_CHARS, (void
    *)NULL );
    ...

}

/* Event handler for MySystemCxo */
long my_cxo_handler( XVT_CXO cxo, EVENT * ep )
{

    /* Post process the events */
    xvt_cxo_call_next( cxo, ep );

    switch( ep -> type )
    {
    ...
    case E_CXO:
    switch( ep -> v.cxo.msg_id )
    {
    case NO_MORE_CHARS:
    {
    EVENT_MASK mask;

    mask = xvt_cxo_get_event_mask( cxo );

    mask &= ~(EM_CHAR);
    xvt_cxo_set_event_mask( cxo, mask );
    break;
    }
    ...
    }
    return( 0L );
    ...
    }

}
```

xvt_cxo_get_class_name

Get the Class Name of a CXO

Summary

```
char * xvt_cxo_get_class_name( XVT_CXO cxo, char *
    class_name, int sz_class_name )
```

`XVT_CXO cxo,`

CXO from which to get the class name.

`char * class_name,`

Buffer in which to return the class name.

`int sz_class_name`

The size of the class name buffer.

Description

This function retrieves the class name for a CXO that was specified during `xvt_cxo_create`. The class name will be returned in the `class_name` buffer and up to `sz_class_name` characters will be returned.

Return Value

A pointer to the `class_name` buffer; `NULL` if an error occurs.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are not met:

- The CXO specified must be valid.
- The `class_name` buffer must not be `NULL`.
- `sz_class_name` must be greater than 0.

See Also

`SZ_CLASS_NAME`
`XVT_CXO`
`xvt_cxo_create`

Example

This example shows how to get the class name of a CXO.

```
char buffer[SZ_CLASS_NAME];

xvt_cxo_get_class_name( cxo, buffer, SZ_CLASS_NAME );
```

xvt_cxo_get_data

Get State Data Associated With a CXO

Summary

```
long xvt_cxo_get_data( XVT_CXO cxo )

XVT_CXO cxo
```

CXO from which to get the state data.

Description

This function retrieves the state data associated with a CXO -- similar to retrieving the application data stored in a `WINDOW`. The state data can be set by using `xvt_cxo_set_data`.

Return Value

The CXO state data, or zero if there is an error or if no data asif sociated with the CXO.

Parameter Validity and Error Conditions

XVT issues an error if the following condition is not met:

The CXO specified must be valid.

See Also

```
XVT_CXO
xvt_cxo_set_data
```

Example

See example for `xvt_cxo_destroy`.

xvt_cxo_get_event_handler

Retrieve Event Handling Function for a CXO

Summary

```
XVT_CXO_EVENT_HANDLER xvt_cxo_get_event_handler( XVT_CXO
    cxo )

XVT_CXO cxo
```

CXO whose event handler is to be retrieved.

Description

This function gets the current event handler for a CXO. The current handler can be changed by calling `xvt_cxo_set_event_handler`.

Return Value

A valid CXO event handler; `NULL` if unsuccessful (on error).

Parameter Validity and Error Conditions

XVT issues an error if the following condition is not met:

The CXO specified must be valid.

See Also

```
XVT_CXO
xvt_cxo_set_event_handler
```

Example

This example shows how to swap CXO event handlers.

```
long my_new_cxo_eh( XVT_CXO cxo, EVENT * ep );

XVT_CXO_EVENT_HANDLER old_cxo_eh;

old_cxo_eh = xvt_cxo_get_event_handler( cxo );
xvt_cxo_set_event_handler( cxo, my_new_cxo_eh );
```

xvt_cxo_get_event_mask

Get Event Mask for a CXO

Summary

```
EVENT_MASK xvt_cxo_get_event_mask( XVT_CXO )

XVT_CXO cxo
```

CXO whose event mask is being retrieved.

Description

This function is used to retrieve the event mask for a CXO. The event mask is set when `xvt_cxo_create` is called and can be changed by using `xvt_cxo_set_event_mask`.

Return Value

The `EVENT_MASK` for the CXO.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are not met:

- The CXO specified must be valid.

See Also

```
EVENT_MASK
XVT_CXO
xvt_cxo_create
xvt_cxo_set_event_mask
```

Example

See example for `xvt_cxo_dispatch_msg`.

xvt_cxo_get_win

Get Window Associated With a CXO

Summary

```
WINDOW xvt_cxo_get_win( XVT_CXO cxo )
```

`XVT_CXO cxo`

CXO whose container is being retrieved.

Description

This function gets the container WINDOW associated with the CXO. The container is set when the CXO is created with `xvt_cxo_create`.

Return Value

The `WINDOW` that contains the CXO.

Parameter Validity and Error Conditions

XVT issues an error if the following condition is not met:

The CXO specified must be valid.

See Also

```
WINDOW
XVT_CXO
xvt_cxo_create
```

xvt_cxo_is_valid

Check the Validity of a CXO

Summary

```
BOOLEAN xvt_cxo_is_valid( cxo )
```

`XVT_CXO cxo`

CXO whose validity is being checked.

Description

This function is used to determine the validity of a CXO.

Return Value

`TRUE` if the CXO is valid: `FALSE` if not.

See Also

```
XVT_CXO
```

Example

See example for `xvt_cxo_destroy`.

xvt_cxo_set_data

Associate State Data With a CXO

Summary

```
void xvt_cxo_set_data( XVT_CXO cxo, long state_data )
```

`XVT_CXO cxo,`

CXO whose state data is to be set.

`long state_data`

The state data to set.

Description

This function sest the state data for a CXO, which is similar to setting application data for a `WINDOWf0`.

Parameter Validity and Error Conditions

XVT issues an error if the following condition is not met:

The CXO specified must be valid.

See Also

```
XVT_CXO
xvt_cxo_get_data
```

Example

See example for `xvt_cxo_create`.

xvt_cxo_set_event_handler

Set CXO Event Handler

Summary

```
void xvt_cxo_set_event_handler(XVT_CXO cxo,
    XVT_CXO_EVENT_HANDLER cxo_eh)
```

`XVT_CXO cxo,`

CXO whose event handler is being set.

`XVT_CXO_EVENT_HANDLER cxo_eh`

CXO handler function.

Description

This function is used to change the event handler for a CXO.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are not met:

- The CXO specified must be valid.
- The `cxo_eh` must point to a valid CXO handler.

See Also

```
XVT_CXO
XVT_CXO_EVENT_HANDLER
xvt_cxo_get_event_handler
```

Example

See example for `xvt_cxo_get_event_handler`.

xvt_cxo_set_event_mask

Specify Event Restrictions For a CXO

Summary

```
void xvt_cxo_set_event_mask( XVT_CXO cxo, EVENT_MASK
    mask )
```

`XVT_CXO cxo,`

CXO whose event mask is being set.

`EVENT_MASK mask`

Specified events that the CXO handler can receive.

Description

This function is used to change the `EVENT_MASK` for the CXO. The event mask affects what events can be sent to a CXO's handler. If the mask is set to `EM_ALL` then all events will be sent to the CXO's handler.

Implementation Note

There is no mask for `E_CXO` events, thus if `EM_NONE` is set for the `EVENT_MASK`, a CXO's handler will only receive `E_CXO` messages. Also note that masked out events will bypass the CXO.

Parameter Validity and Error Conditions

XVT issues an error if the following condition is not met:

The CXO specified must be valid.

See Also

```
E_CXO
EM_* Constants
EVENT_MASK
XVT_CXO
```

Example

See example for `xvt_cxo_dispatch_msg`.

xvt_debug_*

Debug Functions

```
xvt_debug
xvt_debug_printf
```

xvt_debug

Conditionally Append Debugging Information to File

Summary

`void xvt_debug((char *fmt [, arg]...))char *fmt [, arg]...`

Debugging information in `sprintf`-style format.

Description

This macro is a conditional form of `xvt_debug_printf`. It writes debugging information to the debug output file if both of the following conditions are met:

- The preprocessor symbol `DEBUG` is defined when the file containing the `xvt_debug call` is compiled
- The debug output file is already present in the current directory at runtime

Thus, you can control whether any code is generated as a result of calling this macro, and, if code is generated, you can still control whether debugging information is written at run time.

The arguments are identical to those for `xvt_debug_printf` except that you must have an *extra set* of parentheses. That is, `xvt_debug` has only one argument that consists of the arguments to `xvt_debug_printf`, which it calls if debugging output is to be written.

Implementation Note

If the preprocessor symbol `DEBUG` is defined in the application source code, it must precede `#include "xvt.h"`.

The debug output file is called "`debug`" by default, but you can alter it by changing the `ATTR_DEBUG_FILENAME` attribute.

The THINK C compiler on the Mac doesn't allow you to define the symbol `DEBUG` from the "command line." You have to define it at the top of the file that contains calls to `xvt_debug` or in one of your own headers.

The reason for using the debug output file as a flag to enable debugging rather than an environmental variable is that the Mac doesn't support an environment in which variables can be set.

See Also

`ATTR_DEBUG_FILENAME`
`xvt_debug_printf`

The "Diagnostics and Debugging" chapter in the *XVT Portability Toolkit Guide*

Example

Note the use of an extra set of parentheses in each call:

```
BOOLEAN save_ok;
xvt_debug(("About to call do_save..."));
save_ok = do_save();
xvt_debug(("do_save returned %s",
     save_ok? "TRUE" :"FALSE"));
```

xvt_debug_printf

Append Debugging Information to File

Summary

```
void xvt_debug_printf(char *fmt [, arg]...)char *fmt
    [, arg]...
```

Debugging information in `sprintf`-style format.

Description

This function writes debugging information to the debug output file in the current directory; subsequent calls use the same file. Arguments are the same as for `sprintf`.

The debug output file is called "`debug`" by default, but you can alter it by changing the `ATTR_DEBUG_FILENAME` attribute.

If the debug output file exists, this function writes into it. If it does *not* exist, no debug output will occur.

Note that `xvt_debug_printf` cannot be called before `xvt_app_create`, as this is where it is initialized. Calling `xvt_debug_printf` before `xvt_app_create` causes XVT to issue an error.

See Also

`ATTR_DEBUG_FILENAME`
`xvt_debug`

The "Diagnostics and Debugging" chapter in the *XVT Portability Toolkit Guide*

Example

In this example, the first message is guaranteed to appear in the debugging file even if `do_save` causes the application to crash.

```
BOOLEAN save_ok;
xvt_debug_printf("About to call do_save...");
save)ok = do_save();
xvt_debug_printf("do_save returned %s",
     save_ok? "TRUE":"FALSE");
```

xvt_dlg_*

Dialog Functions

`xvt_dlg_create_def`
`xvt_dlg_create_res`

xvt_dlg_create_def

Create a Dialog and Controls from a Data Structure

Summary

```
WINDOW xvt_dlg_create_def(WIN_DEF *win_def_p,
    EVENT_MASK mask, EVENT_HANDLER eh, long app_data)
```

`WIN_DEF *win_def_p`

Pointer to an array of data structures. The first element in the array defines the dialog itself. Subsequent elements of the array define the controls contained within the dialog. The last element of the array is a terminator whose `wtype` field is set to `W_NONE`.

`EVENT_MASK mask`

Bitwise-OR'd combination of the `EM_*` constants. These constants restrict the events that are sent to the dialog's event handler. You normally set this to `EM_ALL` (no restriction).

`EVENT_HANDLER eh`

The event handling function. It receives all of the events for the dialog.

`long app_data`

Contains any application data you wish to attach to the dialog. Often this is a pointer to a data structure cast into a `long`.

Description

This function creates a dialog and its controls based on a description contained in an array of `WIN_DEF` data structures.

For modal dialogs (`WD_MODAL`) this function does not return until the user or the application dismisses (i.e., destroys) the dialog. When you invoke a window or modeless dialog from a modal dialog, you should immediately dismiss the modal dialog so that the newly created window or modeless dialog can receive events and be useful.

When filling in the `WIN_DEF` structures that define the dialog, keep in mind the following:

- `win_def_p[n + 1].wtype = W_NONE` where `n` is the number of controls to be created in the dialog.The array of `win_def_p` is terminated with a `wtype` field set to `W_NONE`.
- `win_def_p[0].wtype` should be set to `WD_MODAL` or `WD_MODELESS` to indicate the type of dialog you want to create.
- `win_def_p[0].rct` should be set to the bounding rectangle for the dialog in the coordinate system of `SCREEN_WIN`.
- `win_def_p[0].text` should point to a string containing the title of the dialog.
- `win_def_p[0].units` should be set to `U_PIXELS`, `U_CHARS`, or `U_SEMICHARS` to indicate which type of coordinate system XVT uses to place the dialog.
- `win_def_p[0].ctlcolors` points to the array of `XVT_COLOR_COMPONENT` structures that define the default colors of the controls in the dialog. If `NULL`, the controls in the dialog use the default application control colors. The last element of the `XVT_COLOR_COMPONENT` array must have an

XVT_COLOR_TYPE of XVT_COLOR_NULL to indicate the end of the array.

- win_def_p[0].v.dlg.flags can contain an OR'd combination of DLG_FLAG_INVISIBLE and DLG_FLAG_DISABLED dialog attributes. These options should only be used with modeless dialogs, since they don't make any sense for modal dialogs and have an undefined effect.
- win_def_p[0].v.dlg.ctl_font_id is the XVT_FNTID that defines the default font used for all the controls in the dialog.

For each control to be created, set its corresponding WIN_DEF element, win_def_p[i], such that:

- win_def_p[i].wtype determines the type of control to be created. The field should contain one of the WC_* constants. For the listing of possible types, see W_*, WC_*, WD_*, Values for WIN_TYPE.
- win_def_p[i].rct defines the bounding rectangle for the control relative to the dialog's client area. If you are creating drop-down list controls of type WC_LISTEDIT or WC_LISTBUTTON, then set this rectangle to include the area that will be occupied by the dropped-down list when the user activates it. XVT will ignore the height of the rectangle when the list is not activated by the user. Instead, XVT will display the edit field or button using the default height for the platform. Because the user activates the drop-down list only when he wants to see the list, it doesn't matter that other controls might overlap with the bounding rectangle.
- win_def_p[i].text is used to set the text of the control in the manner appropriate for the control. For Edit controls, listedit controls, push buttons, radio buttons, check boxes, group boxes, and static text, it sets the label of the control. It has no effect for other controls.
- win_def_p[i].units is filled with one of the type U_PIXELS, U_CHARS or U_SEMICHARS, and specifies the units used to measure the bounding rectangle defined by win_def_p[i].rct.
- win_def_p[i]ctlcolors points to the array of XVT_COLOR_COMPONENT structures that define the control's colors. If it is NULL, the control uses the default colors for the container control colors. The last element of the XVT_COLOR_COMPONENT array should have an XVT_COLOR_TYPE of XVT_COLOR_NULL to indicate the end of the array.

- `win_def_p[i].v.ctl.ctrl_id` is an ID number for the control. When XVT sends an `E_CONTROL` event to the dialog's event handler, it will set the `v.ctl.id` field of the `EVENT` structure to the ID of the control that was activated. This is a way of uniquely identifying the control independently of its window handle. Keep in mind that it is not necessary to use control IDs, but if you choose to use them, all of the IDs for the controls in the dialog must be unique. To retrieve the `WINDOW` for a control, you can also call the `xvt_win_get_ctl` function with an ID and the dialog's window handle.
- `win_def_p[i].v.ctl.icon_id` contains the resource ID for an icon resource to be used for controls of type `WC_ICON`. The icon resource ID refers to an icon resource that is declared non-portably in your resource file. For more information, see the *XVT Platform-Specific Books*.
- `win_def_p[i].v.ctl.flags` controls the `CTL_FLAG_*` options and the initial state of a control. The applicable control flags vary among controls. For a complete table listing the valid control flags for each control, see `CTL_FLAG_* Options` and `Window/Dialog/Control Creation Function Parameters`.
- `win_def_p[i].v.ctl.font_id` is the `XVT_FNTID` that defines the font used in a control. It overrides any font set in `win_def_p[0].v.dlg.ctl.fond_id`. If it is set to `NULL_FNTID`, the control uses the default font for the contrainer control font.

After you have set the controls for your dialog, you must set the `wtype` field of the last `WIN_DEF` structure in the array, `win_def_p[n+1]`, to `W_NONE` to tell XVT that there are no more controls in the dialog.

Return Value

A `WINDOW` if successful; `NULL_WIN` if unsuccessful (on error).

Note: If you are creating a modal dialog, the `WINDOW` returned from `xvt_dlg_create_def` will be returned after the dialog has been destroyed. Therefore, if you are creating a modal dialog, make sure not to use this return value for anything other than to check a successful creation of the dialog.

Parameter Validity and Error Conditions

XVT issues an error if any of the parameters passed to `xvt_dlg_create_def` do not meet the following requirements:

- `win_def_p[0].wtype` must be `WD_MODAL` or `WD_MODELESS`.
- `win_def_p[0].text` must point to a valid string.
- `win_def_p[0].units` must be one of the following: `U_PIXELS`, `U_CHARS`, or `U_SEMICHARS`.
- `win_def_p[0].v.dlg.flags` must be valid.
- `win_def_p[i].wtype` must be one of the `WC_*` control types for each control to be created in the dialog.
- `win_def_p[i].text` must be non-`NULL` for controls that take a title, such as push buttons, radio buttons, check boxes, group boxes, or static text. If you do not want your control to have a title, the empty string ("") is valid.
- `win_def_p[i].v.ctl.flags` must be appropriate for the control you want to create, as defined in `Window/Dialog/ Control Creation Function Parameters`.
- `win_def_p[n+1]`, the `wtype` field of the last `WIN_DEF` structure in the array, must be set to `W_NONE` as a termination.
- `eh` must be set to a valid event handler function.
- `win_def_p[i].ctlcolors` must be either `NULL` or a valid array of `XVT_COLOR_COMPONENT` structures.
- `win_def_p[0].v.dlg.ctl_font_id` must be either `NULL_FNTID` or a valid logical font.
- `win_def_p_[i].v.ctl.font_id` must be either `NULL_FNTID` or a valid logical font.

See Also

```
CTL_FLAG_* Options
EM_* Constants
EVENT_HANDLER
EVENT_MASK
NULL_FNTID
PTR_LONG
W_*, WC_*, WD_*, Values for WIN_TYPE
WIN_DEF
WIN_TYPE
XVT_COLOR_COMPONENT
XVT_COLOR_TYPE
xvt_ctl_create_def
xvt_dlg_create_res
xvt_font_*
xvt_res_get_dlg_def
xvt_vobj_get_data
xvt_vobj_set_data
xvt_win_create_def
xvt_win_get_ctl
Window/Dialog/Control Creation Function Parameters
```

The "Defining and Creating Dialogs" section of the "Dialogs" chapter in the *XVT Portability Toolkit Guide*

xvt_dlg_create_res

Creates a Dialog from a Resource Definition

Summary

```
WINDOW xvt_dlg_create_res(WIN_TYPE wtype, int rid,
    EVENT_MASK event_mask, EVENT_HANDLER eh,
    long app_data)
```

`WIN_TYPE wtype`

`WD_MODAL` or `WD_MODELESS` dialog. This parameter must match the type of dialog specified in the resource file.

`int rid`

Specifies the dialog resource to be used. This resource ID must match the dialog's ID in the resource file.

`EVENT_MASK event_mask`

Indicates which events should be sent to your dialog's event handler. Normally, this is set to `EM_ALL`, indicating that all events should be sent.

`EVENT_HANDLER eh`

The dialog's event handling function.

`long app_data`

Data that your application can attach to the dialog when it is created. Often this is set to a pointer to a data structure cast into a `long`.

Description

This function creates a dialog according to a description specified in a resource file. Typically, the dialog resource is specified in URL, but it can also be specified in the native resource language.

For modal dialogs (`WD_MODAL`), this function does not return until the user or the application dismisses (i.e., destroys) the dialog. When you invoke a window or modeless dialog from a modal dialog, you should immediately dismiss the modal dialog so that the newly created window or modeless dialog can receive events and be useful.

Return Value

A `WINDOW` if successful; `NULL_WIN` if unsuccessful (on error).

Note: If you are creating a modal dialog, the `WINDOW` returned from `xvt_dlg_create_res` is returned after the dialog has been destroyed. Therefore, if you are creating a modal dialog, make sure *not* to use this return value for anything other than to check a successful creation of the dialog.

Parameter Validity and Error Conditions

XVT issues an error if any of the following parameter conditions are not met:

- `wtype` must be one of `WD_MODAL` or `WD_MODELESS` and must match the type of dialog you specified in your resource file
- `rid` must be a valid dialog resource specified in the URL file or by some non-portable means
- `eh` must point to a valid event handler

See Also

```
EM_* Constants
EVENT_HANDLER
EVENT_MASK
NULL_WIN
W_*, WC_*, WD_*, Values for WIN_TYPE
WINDOW
WIN_TYPE
xvt_dlg_create_def
xvt_res_get_dlg_def
xvt_vobj_get_data
xvt_vobj_set_data
xvt_win_create_res
dialog URL statement
```

The "Defining and Creating Dialogs" section of the "Dialogs" chapter in the *XVT Portability Toolkit Guide*

xvt_dm_*

Dialog Management Functions

```
xvt_dm_post_about_box
xvt_dm_post_ask
xvt_dm_post_color_sel
xvt_dm_post_ctools_sel
xvt_dm_post_dir_sel
xvt_dm_post_error
xvt_dm_post_fatal_exit
xvt_dm_post_file_open
xvt_dm_post_file_save
xvt_dm_post_font_sel
xvt_dm_post_message
xvt_dm_post_note
xvt_dm_post_page_setup
xvt_dm_post_string_prompt
xvt_dm_post_warning
```

xvt_dm_post_about_box

Display About Box

Summary

```
void xvt_dm_post_about_box(void)
```

Description

Each XVT Portability Toolkit provides a menu item that allows the user to view an About box. Selection of the standard menu item invokes `xvt_dm_post_about_box` rather than delivering an `E_COMMAND` event to the application.

You can also implement your own calls to `xvt_dm_post_about_box` from other places in your application. For example, you might want to call `xvt_dm_post_about_box` when the application starts up.

`xvt_dm_post_about_box` invokes the About box specified by an application resource ID defined in the `about_box_ID` data structure member of `XVT_CONFIG`. If `about_box_ID` is set to zero, XVT uses the resource defined as `DB_ABOUT` provided in **url_plat.h**. Otherwise you must define resources for an About box in your URL file. The About box must be modal.

XVT provides all event handling for the About box dialog, whether it is XVT's default About box dialog or the one you defined in a resource file. The event handler provided by XVT processes `E_CONTROL` events from buttons with IDs of `DLG_OK` and `DLG_CANCEL`. If the user presses a button with an ID equal to `DLG_CANCEL`, XVT event handling dismisses the About box. If the user presses a button with an ID equal to `DLG_OK`, XVT sends an `E_HELP` event to invoke the help viewer. The `DLG_OK` button is optional.

See Also

```
DLG_* Control IDs
NO_STD_ABOUT_BOX
XVT_CONFIG
xvt_app_create
```

The "Dialogs" chapter in the *XVT Portability Toolkit Guide*

xvt_dm_post_ask

Ask User a Question

Summary

```
ASK_RESPONSE xvt_dm_post_ask(char *lbl_dflt,
    char *lbl2, char *lbl3, char *fmt [, arg] ...)
```

`char *lbl_dflt`

Text of default button.

`char *lbl2`

Text of second button.

`char *lbl3`

Text of third button.

`char *fmt [, arg] ...`

Question in `sprintf-style` format.

Description

This function displays a dialog box that asks the user a question with *two* (typically "Yes" and "No") or *three* (typically "Yes," "No," and "Cancel") buttons. Names for the buttons are specified by supplying appropriate values for the first three arguments. The first argument should be the *default* button, which is highlighted and chosen if the user presses the Return key. If only two buttons are wanted, argument `lbl3` should be `NULL`.

The arguments beginning with `fmt` are a format followed by optional values, in the style of `sprintf`. These arguments form the question to be asked. The total length of the question (`fmt` with all arguments converted to strings according to the corresponding format specifiers) must be less than 200 bytes.

Tip: When determining the *default* button, keep in mind that it should either be the one *most often chosen* or the *safest choice*. To make responding convenient for the user, the default button should be the one that's most often chosen (typically "Yes"). However, if a "Yes" choice could result in a potentially undesirable action, it should not be the default button. For example, if the question is "Delete file XYZ?" then the default should be "Cancel," because an accidental "Yes" could be disastrous.

For additional clarity, it's a good idea to label a button with a particular action, rather than with a literal answer to the question. In that case, the user can safely click a button even without reading the question and determining the meaning of "Yes" and "No." For example, if the question is "Discard changes?" then the buttons should be labeled "Discard" and "Save," rather than "Yes" or "No." Regardless of the question or the user's interpretation of it, the meaning of "Discard" and "Save" is clear.

Note: If you want a dialog box with only *one* button, use `xvt_dm_post_error` or `xvt_dm_post_note` instead of `xvt_dm_post_ask`.

Return Value

`RESP_DEFAULT`, `RESP_2`, or `RESP_3`, depending on which button the user clicked. Closing the window (using the window system menu "Close" item or the "Escape" key) returns `RESP_3`, even if no third button is present.

See Also

```
xvt_dm_post_error
xvt_dm_post_note
```

The "Dialogs" chapter in the *XVT Portability Toolkit Guide*

Example

```
switch (xvt_dm_post_ask("Save", "Discard", "Cancel",
    "Save changes to "%s" before closing?",filename)) {
case RESP_DEFAULT:
    save_drawing();/* save and close window */
    discard_window();
    break;
case RESP_2:
    discard_window();/* discard window */
    break;
case RESP_3:
    break;/* don't close the window */
default:
    xvt_dm_post_error("Invalid return from ask.");
}
```

xvt_dm_post_color_sel

Post Standard Dialog to Select a Color

Summary

```
BOOLEAN xvt_dm_post_color_sel(
    COLOR *color)

COLOR *color
```

Points to a color value that the dialog is initialized with and will return the color selected.

Description

This function puts up a standard dialog box that allows the user to select a color. The dialog box is consistent with the native look-and-feel.

Return Value

TRUE if a new color was selected, otherwise FALSE.

xvt_dm_post_ctools_sel

Set a DRAW_TOOLS struct in a Standard Modal Window

Summary

```
BOOLEAN xvt_dm_post_ctools_sel (DRAW_CTOOLS *ctoolsp,
long attr)
```

DRAW_CTOOLS *ctoolsp
It must point to a valid DRAW_CTOOLS.

long attr
Sets which DRAW_CTOOLS members will be visible to the user for selection.

The attr parameter takes ORed values of XVT_CTOOL_* constants.

Description

This function puts up a standard dialog, which is consistent with the native look-and-feel, that allows the user to choose colors, styles, modes, etc. for drawing and text.

Your application must supply to xvt_dm_post_ctools_sel a valid DRAW_CTOOLS pointer.

Return Value

TRUE if the user clicked OK (xvt_dm_post_ctools_sel does not check whether or not any values were actually changed by the user); FALSE otherwise.

Parameter Validity and Error Conditions

Calling this function with a pointer that is not a valid DRAW_CTOOLS pointer results in undefined behavior. Calling this function with a NULL DRAW_CTOOLS pointer results in a runtime error and message and a return value of FALSE. Calls with illegal values or NULL for the attr parameter generate a runtime error and message and a return value of FALSE. This function cannot be called during an E_UPDATE event.

Implementation Note

No error checking of the DRAW_CTOOLS parameter's members is done.

`xvt_dm_post_ctools_sel` uses resources at run time. Therefore, be sure the file **url.h** is included during compilation of resources.

See Also

`DRAW_CTOOLS`
`XVT_CTOOLS_*` Values for attr
The "Dialogs" chapter in the *XVT Portability Toolkit Guide*

Example

```
/*NOTE: The current window is 'win' */
DRAW_CTOOLS ctool;
xvt_dwin_draw_text ( win, 20, 50, "Old color." );
xvt_dwin_get_draw_ctools ( win, &ctool );
if ( xvt_dm_post_ctools_sel ( &ctool, ( XVT_CTOOL_PEN )) {
    xvt_dwin_set_draw_ctools ( win, &ctool);
    xvt_dwin_draw_text ( win, 20, 100, "Returned TRUE.");
} else
xvt_dwin_draw_text ( win, 20, 100, "Returned FALSE.");
```

xvt_dm_post_dir_sel

Post Standard Dialog to Select a Directory

Summary

```
FL_STATUS xvt_dm_post_color_sel(FILE_SPEC *fsp,
    char *msg)
```

`FILE_SPEC *fsp`

File specification. It must point to a `FILE_SPEC`.

`fsp->type`
Set to the empty string ("").

`fsp->dir`
Set to the directory initially presented to the user. If the initial directory doesn't matter, set `fsp->dir` to a directory retrieved via `xvt_fsys_get_default_dir`.

`fsp->name`
Should be initialized to an empty string.

`char *msg`
String to display. `msg` points to a `NULL`-terminated character string prompting the user for a response.

Description

This function puts up a standard dialog box, which is consistent with the native look-and-feel, that allows the user to select a directory. The dialog supplies the user with a list of directories from which to choose, and may allow the user to change the base directory.

Return Value

`FL_OK` if the user clicked OK (meaning you should use the directory); `FL_CANCEL` if the user clicked Cancel (meaning you should abort the directory); `FL_BAD` if an error occurred, in which case an alert dialog has already been displayed by the function. Upon return, the `FILE_SPEC->dir` contains the selected directory.

See Also

```
DIRECTORY
FILE_SPEC
FL_* Values
xvt_dm_post_file_open
xvt_dm_post_file_save
xvt_fsys_get_default_dir
```

xvt_dm_post_error

Display Alert Box with Error Icon

Summary

```
void xvt_dm_post_error(char *fmt [, arg] ...)char *fmt
    [, arg] ...
```

Message in `sprintf`-style format.

Description

This function puts up an alert box containing a message, an OK button, and an error icon appropriate to the native look-and-feel for each supported platform. When the user clicks the button, the alert box is removed and `xvt_dm_post_error` returns.

The message is specified by a format and arbitrary arguments, similar to the standard C function `sprintf`.

The total length of the message (`fmt` with all arguments converted to strings) must be less than 200 bytes.

See Also

```
xvt_dm_post_fatal_exit
xvt_dm_post_note
xvt_dm_post_warning
```

Example

See the example for `xvt_dm_post_file_open`.

xvt_dm_post_fatal_exit

Display Error Message and Terminate

Summary

```
void xvt_dm_post_fatal_exit(char *fmt [, arg] ...)char
    *fmt [, arg] ...
```

Message in `sprintf-style` format.

Description

This function displays a message which is specified using a format and arbitrary arguments, similar to the standard C function `sprintf`. When the user clicks the mouse, the alert box is removed and `xvt_app_destroy` is called, ending execution. Naturally, `xvt_dm_post_fatal_exit` should be called only in dire emergencies from which your application can't recover.

The total length of the message (`fmt` with all arguments converted to strings according to the corresponding format specifiers) must be less than 200 bytes.

Note: If your application calls `xvt_dm_post_fatal_exit` to put up an alert box, the message box doesn't require resources or additional memory. It can be seen even when memory is exhausted or when ordinary dialog boxes can't be created by calling `xvt_dm_post_error` or `xvt_dm_post_note`.

See Also

```
xvt_dm_post_error
xvt_dm_post_message
xvt_dm_post_note
```

The "Dialogs" chapter in the *XVT Portability Toolkit Guide*

Example

See the example for `xvt_dm_post_ask`.

xvt_dm_post_file_open

Get File to Open with Standard Dialog

Summary

```
FL_STATUS xvt_dm_post_file_open(FILE_SPEC *fsp,
    char *msg)
```

`FILE_SPEC *fsp`

File specification. It must point to a `FILE_SPEC`.

`fsp->type`
Set to the type of files the user will be allowed to select. If `fsp->type` is the empty string (""), all types are allowed.

`fsp->dir`
Set to the directory initially presented to the user. The user can change to other directories. If the initial directory doesn't matter, set `fsp->dir` to a `DIRECTORY` retrieved via `xvt_fsys_get_default_dir`.

`fsp->name`
Should be initialized to an empty string.

`char *msg`
String to display. `msg` points to a `NULL`-terminated character string prompting the user for a response. For example, the prompt might be "Select drawing file...". On some platforms, the length of the message string is limited.

Description

This function puts up a standard dialog box, which is consistent with the native look-and-feel, that allows the user to supply the name of a file to be opened for *reading*. The dialog normally supplies a list of files for the user to choose from, and may allow the user to change directories.

Your application must supply to `xvt_dm_post_file_open` the starting directory and the type of files to display.

`xvt_dm_post_file_open` doesn't open the file--only its name is retrieved. Upon return, your application must change to the proper

directory, check that the file exists, check that the file is readable, and then finally open it.

Depending on the implementation, the user may or may not be able to specify a file type different from that specified in `fsp->type` when `xvt_dm_post_file_open` was called. Normally, even if this were done, you should assume that the user knew what he or she was doing, and read the file as though it were of the correct type.

After `xvt_dm_post_file_open` returns, the following is true:

- `fsp->dir` and `fsp->name` is set to the user's selections. Before opening the file for reading, you should change directories to `fsp->dir` by calling `xvt_fsys_set_dir(&fsp>dir)`. The `fsp>name` argument is a `NULL`-terminated character string that can be used directly in calls to the standard C functions `fopen` and `open`.
- `xvt_dm_post_file_open` might change the current directory. To compensate for this fact, you can use `xvt_fsys_save_dir` to remember the current directory before calling this function, and `xvt_fsys_restore_dir` to put it back later.

Return Value

`FL_OK` if the user clicked OK (you should open the file); `FL_CANCEL` if the user clicked Cancel (the command that tried to open the file should be aborted); `FL_BAD` if an error occurred, in which case an alert has already been displayed by `xvt_dm_post_file_open`. Upon return, the `FILE_SPEC` contains valid data only if `FL_OK` is the return value.

Implementation Note

With XVT/Win32 and XVT/XM, but not with XVT/Mac, the user can use the dialog box to view files of types other than the one specified in the call. Also, on these systems the user can actually enter a name from the keyboard (which may or may not exist), but with XVT/Mac the user can only choose from the names listed.

With XVT/Mac, the application can additionally filter the filename patterns by setting the `fsp->name` field to an XVT filename pattern. Also, directory selection can be done by setting the `fsp->type` field to "`Fldr`." The message string is limited to 255 bytes (325 pixels wide), but with overrun if the dialog is too long.

See Also

DIRECTORY
FILE_SPEC
FL_* Values for FL_STATUS
xvt_dm_post_file_save
xvt_fsys_get_default_dir
xvt_fsys_get_file_attr
xvt_fsys_restore_dir
xvt_fsys_save_dir
xvt_fsys_set_dir
xvt_fsys_set_file_attr

The "Dialogs" chapter in the *XVT Portability Toolkit Guide*

Example

```
FILE_SPEC fs_in;fs_in.type[0] = '0'; /* want all types */
fs_in.name[0] = '0';
xvt_fsys_get_default_dir(&fs_in.dir);
switch (xvt_dm_post_file_open(&fs_in,
    "Select input file...")) {
case FL_BAD:
    xvt_dm_post_error("Error getting file name.");
    return;
case FL_CANCEL:
    return;
case FL_OK:
    break;
}
xvt_fsys_set_dir(&fs_in.dir);
if ((in = fopen(fs_in.name, "r")) == NULL) {
    xvt_dm_post_error("Can't open file "%s".",
        fs_in.name);
    return;
}
```

xvt_dm_post_file_save

Post Standard Dialog to get Filename for Saving

Summary

```
FL_STATUS xvt_dm_post_file_save(FILE_SPEC *fsp,
    char *msg)
```

`FILE_SPEC *fsp`

File specification. It must point to a `FILE_SPEC`.

`fsp->type`

Set to the type of files the user will be allowed to select. If `fsp->type` is the empty string (""), all types are allowed.

`fsp->dir`

Set to the directory initially presented to the user. The user may be allowed to change to other directories. If the initial directory doesn't matter, set `fsp->dir` to a `DIRECTORY` retrieved via `xvt_fsys_get_default_dir`.

`fsp->name`

Set to a suggested filename.

`char *msg`

String to display. Points to a `NULL`-terminated character string prompting the user for a response. For example, the prompt might be "Enter drawing file...". On some platforms, the length of the message string is limited.

Description

This function puts up a standard dialog box, which is consistent with the native look-and-feel, that allows the user to supply the name of a file to be *created* or *opened for writing*. The dialog supplies a list of files for the user to choose from, and may allow the user to change directories.

Your application must supply to `xvt_dm_post_file_save` the starting directory and the type of files to display.

`xvt_dm_post_file_save` doesn't open or create the file--only its name is retrieved. Upon return, your application must change to the proper directory, check that the file is writable or can be created, and then finally open or create it.

After `xvt_dm_post_file_save` returns, the following is true:

- `fsp->dir` and `fsp->name` is set to the user's selections. Before opening the file for writing, you should change directories to `fsp->dir` by calling `xvt_fsys_set_dir(&fsp>dir)`. The `fsp>name` argument is a `NULL`-terminated character string that can be used directly in calls to the standard C functions `fopen`, `open`, and `create`.
- `xvt_dm_post_file_save` might change the current directory. To compensate for this fact, you can use `xvt_fsys_save_dir` to remember the current directory before calling this function, and `xvt_fsys_restore_dir` to put it back later.

Return Value

FL_OK if the user clicked OK (meaning you should write the file); FL_CANCEL if the user clicked Cancel (meaning you should abort the write command); FL_BAD if an error occurred, in which case an alert dialog has already been displayed by this function. Upon return, the FILE_SPEC contains valid data only if FL_OK is the return value.

See Also

DIRECTORY
FILE_SPEC
FL_* Values for FL_STATUS
xvt_dm_post_file_open
xvt_fsys_get_default_dir
xvt_fsys_get_file_attr
xvt_fsys_set_dir
xvt_fsys_set_file_attr

The "Dialogs" chapter in the *XVT Portability Toolkit Guide*

Example

```
FILE_SPEC *get_save_file(char *default_name,
char *file_type)
{
static FILE_SPEC fs;
    xvt_fsys_set_file_attr(&fs, XVT_FILE_ATTR_FILE_STR,
        (long) default_name);
    xvt_fsys_set_file_attr(&fs, XVT_FILE_ATTR_TYPESTR,
         long) file_type);
    xvt_fsys_get_default_dir(&fs.dir);
    switch (xvt_dm_post_file_save(&fs, "Save as:")) {
    case FL_OK:
        return(&fs);
    case FL_BAD:
    case FL_CANCEL:
        return(NULL);
    }
}
```

xvt_dm_post_font_sel

Call the XVT Native Font Selection Dialog

Summary

```
BOOLEAN xvt_dm_post_font_sel(WINDOW win,
    XVT_FNTID font_id, PRINT_RCD *precp,
    unsigned long reserved)
```

`WINDOW win`

`NULL_WIN` or window to which to send the `E_FONT` event.

`XVT_FNTID font_id`

Handle to the logical font used to preset font selection and retrieve the result.

`PRINT_RCD *precp`

Printer record to be used for providing access to physical printer fonts. If this parameter is `NULL`, no printer fonts are shown. If this parameter is a valid `PRINT_RCD` pointer, then physical fonts are shown for both the screen and the given printer.

`unsigned long reserved`

Reserved for future use; it must be 0.

Description

This function calls the XVT native Font Selection dialog. If the application has set the `ATTR_FONT_DIALOG` attribute, the application-customized font dialog is invoked instead of the XVT-supplied one.

By default, the Font Selection dialog operates in synchronous mode. In this mode, the application waits for the user to make a selection. If `win` is *not* `NULL_WIN`, an `E_FONT` event is dispatched to that window.

If you want your application to have a Font Selection dialog that operates in asynchronous mode, you must create a modeless dialog using `ATTR_FONT_DIALOG`. In either mode, the selected font is returned.

If the user makes a selection and exits normally, the dialog returns a reasonable set of portable XVT logical font attributes in the `XVT_FNTID` in the generated `EVENT` (if the calling function requests it), and in the `font_id`. The native font descriptor is also set to match

the user's selection. A "reasonable set" of portable XVT logical font attributes consists of attributes that correspond closely to the native font that was selected: family, style, size, and native descriptor.

The native descriptor is set for the resulting logical font and is honored in any subsequent call to `xvt_font_map` or `xvt_font_map_using_default`.

Sometimes an exact match cannot be made between the user's selection and the set of XVT portable logical font attributes. This occurs if the selection contains some attributes that don't correspond to any XVT portable logical font attributes, or if it has style settings that don't exist in XVT. In these cases, the dialog attempts to specify a "best fit" with the XVT portable logical font attributes.

Return Value

`TRUE` if the logical font passed in as the preset for selection has been changed; `FALSE` otherwise.

Parameter Validity and Error Conditions

XVT issues an error if any of the following conditions are true:

- `xvt_dm_post_font_sel` is called during `E_UPDATE`
- `font_id` is `NULL` or invalid
- `win` is invalid or non-drawable
- `precp` is invalid

See Also

```
ATTR_FONT_DIALOG
E_FONT
PRINT_RCD
XVT_FNTID
xvt_font_map
xvt_font_map_using_default
```

The "Customized Font Selection Dialogs" section of the "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*

Example

```
void change_font(WINDOW window)
{
    XVT_FNTID font_id;
    font_id = xvt_dwin_get_font(window);
    if (xvt_dm_post_font_sel(window, font_id,
        (PRINT_RCD *) NULL, 0L))
    xvt_dwin_set_font(window, font_id);
    xvt_font_destroy(font_id);
}
```

xvt_dm_post_message

Output Emergency Message

Summary

```
void xvt_dm_post_message(char *fmt [, arg] ...)char *fmt
    [, arg] ...
```

Message in `sprintf`-style format.

Description

This function puts up a modal dialog containing a message. The message is specified using a format and arbitrary arguments, similar to the standard C function `sprintf`. The total length of the message (`fmt` with all arguments converted to strings according to the corresponding format specifiers) must be less than 200 bytes.

Note: The dialog put up by `xvt_dm_post_message` doesn't require resources or additional memory, so it is very likely to be seen even when memory is exhausted or when ordinary dialog boxes can't be displayed.

You should call this function only in emergencies; in most cases, it's better to call `xvt_dm_post_error` or `xvt_dm_post_note` instead.

See Also

```
xvt_dm_post_error
xvt_dm_post_fatal_exit
xvt_dm_post_note
```

xvt_dm_post_note

Display Alert Box with Note Icon

Summary

```
void xvt_dm_post_note(char *fmt [, arg] ...)char *fmt
    [, arg] ...
```

Message in `sprintf`-style format.

Description

This function puts up an alert box with a message, an OK button, and a note icon appropriate to the native look-and-feel for each

supported platform. The message is specified using a format and arbitrary arguments, similar to the standard C function `sprintf`. When the user clicks the button, the alert box is removed and `xvt_dm_post_note` returns.

The total length of the message (`fmt` with all arguments converted to strings according to the corresponding format specifiers) must be less than 200 bytes.

See Also

```
xvt_dm_post_error
xvt_dm_post_fatal_exit
xvt_dm_post_message
```

Example

```
xvt_dm_post_note("Thank you for using %s",
     product_name);
```

xvt_dm_post_page_setup

Display Standard Page Setup Dialog

Summary

```
BOOLEAN xvt_dm_post_page_setup(PRINT_RCD *precp)
```

`PRINT_RCD *precp`

Pointer to page setup to be displayed.

Description

This function puts up a dialog box allowing the user to adjust the page setup stored in the `PRINT_RCD` pointed to by `precp`. The page setup dialog is platform-specific look-and-feel, but it generally consists of page size (e.g., US Letter, US Legal, etc.), output scale, device resolution, and output orientation (e.g., Landscape, etc.).

`xvt_dm_post_page_setup` should be called in response to the user choosing the Page Setup command on the File menu (signified by an `E_COMMAND` event with `v.cmd.tag` set to `M_FILE_PG_SETUP`).

If the user modifies the `PRINT_RCD`, then `xvt_dm_post_page_setup` returns `TRUE` to indicate that the altered `PRINT_RCD` should be stored in memory with the document. The document should be marked as "unsaved."

If xvt_dm_post_page_setup returns FALSE, then no change was made to the PRINT_RCD.

If your application has just read a PRINT_RCD from a document file, then it must call xvt_print_is_valid to check its validity before passing the PRINT_RCD to this function. If xvt_print_is_valid returns FALSE, your application must create a valid print record (with xvt_print_create) to use this function.

Once the user clicks OK in the dialog box, your application can get the page size metrics using the xvt_app_escape function. This escape gives you the printer information from the passed print record. Any of the return parameter pointers can be NULL. The call signature is:

```
xvt_app_escape(XVT_ESC_GET_PRINTER_INFO,
    PRINT_RCD* print_rcd, long* heightp,
    long* widthp, long* vresp, long* hresp);
```

`PRINT_RCD* print_rcd`

Current print record.

`long* heightp`

Height in dots.

`long* widthp`

Width in dots.

`long* vresp`

Vertical resolution in dots per inch.

`long* hresp`

Horizontal resolution in dots per inch.

Return Value

TRUE if the user modified the PRINT_RCD; FALSE if the user left it alone.

Implementation Note

On XVT/Mac, xvt_dm_post_page_setup brings up the standard Page Setup dialog box for the current printer.

On XVT/Win32, this function allows the user to change both the target printer and the page setup for that printer from the platform- and driver-specific native dialogs. On these platforms, both the current printer and its page setup are stored in a PRINT_RCD.

On XVT/XM, if the user has a Level 2 PostScript printer with multiple paper trays, the proper paper size is automatically selected. Otherwise, it is the user's responsibility to be sure that the printer is set up correctly.

See Also

```
PRINT_RCD
XVT_ESC_*
xvt_app_escape
xvt_print_create
xvt_print_is_valid
```

The "Printing" chapter in the *XVT Portability Toolkit Guide*

Example

```
DOC *doc;
switch (xdEvent->v.cmd.tag)
{
    ...
case M_FILE_PG_SETUP:
    doc = (DOC *) xvt_vobj_get_data(xdWindow);
    if (doc->print_rcd != NULL)
        if (xvt_dm_post_page_setup(doc->print_rcd))
            change_page_setup(doc);
    break;
    ...
}
```

xvt_dm_post_string_prompt

Put Up a Text-response Dialog

Summary

```
char *xvt_dm_post_string_prompt(char *msg, char *resp,
    int sz_resp)
```

`char *msg`

Message to be shown.

`char *resp`

User response buffer.

`int sz_resp`

Buffer size of `resp` in bytes.

Description

This function puts up a dialog that contains the message specified by `msg`, an edit control, and OK and Cancel buttons. When the dialog appears, it allows the user to type a response into its edit field.

If the user clicks OK, the response is stored into the string specified by `resp`, whose maximum capacity (including the terminating `NULL`) is `sz_resp`; the response is truncated as necessary to fit in `resp`.

You must initialize `resp` with a default response that will be placed in the edit box when the dialog appears. If you have no default response, set `resp` to the *empty string*, not to `NULL` (see the example below for the proper way to do this).

At most 255 bytes can be typed in the edit box--your `resp` buffer does not need to be larger. Approximately 100 bytes of `msg` text can be displayed.

This function is intended for use when testing your application or as a temporary fill-in while completing your user interface. For most applications, custom-designed dialogs will look better.

Return Value

Value of `resp` argument if OK is clicked; `NULL` if Cancel is clicked.

Parameter Validity and Error Conditions

XVT issues an error if `msg` or `resp` is `NULL`.

See Also

The "Dialogs" chapter in the *XVT Portability Toolkit Guide*

Example

```
char s[100];s[0] = '0';
if (xvt_dm_post_string_prompt(
    "Type something...", s, sizeof(s)) == NULL)
    xvt_dm_post_warning("You canceled.");
else
    xvt_dm_post_note("You typed '%s'.", s);
```

xvt_dm_post_warning

Display Alert Box with Warning Icon

Summary

```
void xvt_dm_post_warning(char *fmt [, arg] ...)char *fmt
    [, arg] ...
```

Message in `sprintf`-style format.

Description

This function puts up an warning box containing a message, an OK button, and a warning icon appropriate to the native look-and-feel for each supported platform. When the user clicks the button, the warning box is removed and `xvt_dm_post_warning` returns.

The message is specified by a format and arbitrary arguments, similar to the standard C function `sprintf`.

The total length of the message (`fmt` with all arguments converted to strings) must be less than 200 bytes.

See Also

```
xvt_dm_post_error
xvt_dm_post_fatal_exit
xvt_dm_post_note
```

The "Dialogs" chapter in the*XVT Portability Toolkit Guide*

Example

See the example for `xvt_dm_post_string_prompt`.