`wmgr_completechangerect()`

Calls `XConfigureWindow` with a new `rect`.

```
void
wmgr_completechangerect(
  window, rectnew, rectoriginal,
  parentprleft, parentprtop)
        Xv_window window;
          Rect   *rectnew, *rectoriginal;
          int     parentprleft, parentprtop;
```

`wmgr_open()`

Sets `wmhints.initial_state = NORMALSTATE`

```
void
wmgr_open(frame)
      Frame frame;
```

`wmgr_refreshwindow()`

Calls `XConfigureWindow` repeatedly.

```
void
wmgr_refreshwindow(window)
      Xv_window window;
```

`wmgr_top()`

Sets stackmode to `Above` in `XConfigureWindow`.

```
void
wmgr_top(frame)
      Frame frame;
```

`window_read_event()`

Reads the next input event for `window`. In case of error, sets the global variable `errno` and returns `-1`.

```
int
window_read_event(window, event)
     Xv_window  window;
     Event      *event;
```

2.15 Window Manager Procedures and Macros

`wmgr_bottom()`

Sets stackmode to `Below` in `XConfigureWindow`.

```
void
wmgr_bottom(frame)
    Frame     frame;
```

`wmgr_changelevel()`

Sets stackmode in `XConfigureWindow`.

```
void
wmgr_changelevel(window,parent,top)
    Xv_Window  window;
    int        parent;
    int        top;
```

`wmgr_close()`

Sets `wmhints.initial_state = ICONICSTATE`

```
void
wmgr_close(frame)
    Frame     frame;
```

`ttysw_output()`

Appends `len` number of characters from `buf` onto `tty`'s output queue; it sends them through the terminal emulator to the `TTY`. It returns the number of characters accepted.

```
int
ttysw_output(tty, buf, len)
     Tty   tty;
     char *buf;
     int   len;
```

2.14 Window Procedures and Macros

`window_done()`

Destroys the entire hierarchy to which `win` belongs.

```
int
window_done(win)
     Xv_Window win;
```

`window_fit()`

Causes `win` to fit its contents in the dimensions specified with `WIN_FIT_HEIGHT` and `WIN_FIT_WIDTH`. A macro, defined as:

```
int
xv_set(
   win, WIN_FIT_HEIGHT,
   WIN_FIT_WIDTH, 0, 0)
window_fit(win)
     Xv_Window win;
```

`window_main_loop()`

Displays `base_frame` on the screen and begins the processing of events by passing control to the Notifier.

```
void
window_main_loop(base_frame)
     Frame base_frame;
```

`textsw_scroll_lines()`

Moves the text up or down by `count` lines. If `count` is positive, then the text is scrolled up on the screen. If negative, the text is scrolled down (backward in the file).

```
void
textsw_scroll_lines(textsw, count)
     Textsw  textsw;
     int     count;
```

`textsw_set_selection()`

Sets the selection to begin at `first` and includes all characters up to `last_plus_one`. A `type` value of `1` indicates primary selection, `2` indicates secondary selection.

```
void
textsw_set_selection(textsw, first,
last_plus_one, type)
     Textsw        textsw;
     Textsw_index  first, last_plus_one;
     unsigned      type;
```

`textsw_store_file()`

Stores the contents of `textsw` to the file named by `filename`. If needed, a message box will be displayed at `x, y`.

```
unsigned
textsw_store_file(textsw, filename, x, y)
     Textsw  textsw;
     char   *filename;
     int     x, y;
```

2.13 Tty Procedures and Macros

`ttysw_input()`

Appends `len` number of characters from `buf` onto `tty`'s input queue. It returns the number of characters accepted.

```
int
ttysw_input(tty, buf, len)
     Tty   tty;
     char *buf;
     int   len;
```

`textsw_replace_bytes()`

Replaces the character span from `first` to `last_plus_one` with the characters in `buf`.

The return value is the number of bytes inserted or deleted. The number is positive if bytes are inserted, negative if bytes are deleted. (The number is also negative if the original string is longer than the one that replaces it.) If this operation fails, it will return a value of `0`.

```
Textsw_index
textsw_replace_bytes(
  textsw, first, last_plus_one, buf, buf_len)
     Textsw        textsw;
     Textsw_index  first;
     Textsw_index  last_plus_one;
     char         *buf;
     unsigned      buf_len;
```

`textsw_reset()`

Discards edits performed on the contents of `textsw`. If needed, a message box will be displayed at `x, y.`

```
void
textsw_reset(textsw, x, y)
     Textsw textsw;
     int    x, y;
```

`textsw_save()`

Saves any edits made to the file currently loaded into `textsw`. If needed, a message box will be displayed at `x, y.`

```
unsigned
textsw_save(textsw, x, y)
     Textsw textsw;
     int    x, y;
```

`textsw_screen_line_count()`

Returns the number of screen lines in `textsw`.

```
int
textsw_screen_line_count(textsw)
     Textsw textsw;
```

`textsw_next()`

Returns the next view in the set of `textsw` views.

```
Textsw
textsw_next(textsw)
     Textsw textsw;
```

`textsw_normalize_view()`

Repositions the text so that the character at `position` is visible and at the top of the subwindow.

```
void
textsw_normalize_view(textsw, position)
     Textsw       textsw;
     Textsw_index position;
```

`textsw_possibly_normalize()`

If the character at `position` is already visible, this function does nothing. If it is not visible, it repositions the text so that it is visible and at the top of the subwindow

```
void
textsw_possibly_normalize(textsw, position)
     Textsw       textsw;
     Textsw_index position;
```

`textsw_remove_mark()`

Removes an existing mark from `textsw.`

```
void
textsw_remove_mark(textsw, mark)
     Textsw      textsw;
     Textsw_mark mark;
```

`textsw_first()`

Returns the first `textsw` view.

```
Textsw
textsw_first(textsw)
     Textsw textsw;
```

`textsw_index_for_file_line()`

Returns the character index for the first character in the line given by `line`. If this operation fails, it will return `TEXTSW_CANNOT_SET`.

```
Textsw_index
textsw_index_for_file_line(textsw, line)
     Textsw textsw;
     int    line;
```

`textsw_insert()`

Inserts characters in `buf` into `textsw` at the current insertion point. The number of characters actually inserted is returned. This will equal `buf_len` unless there was a memory allocation failure. If there was a failure, it will return `0`.

```
Textsw_index
textsw_insert(textsw, buf, buf_len)
     Textsw  textsw;
     char    *buf;
     int     buf_len;
```

`textsw_match_bytes()`

Searches for a block of text in the `textsw`'s contents; ends with characters matching `end_sym`. This function places the starting index of the matching block in `first` and its ending index in `last`.

```
int
textsw_match_bytes(
  textsw, first, last_plus_one, start_sym,
  start_sym_len, end_sym, end_sym_len,
  field_flag)
     Textsw        textsw;
     Textsw_index  *first, *last_plus_one;
     char          *start_sym, *end_sym;
     int           start_sym_len,end_sym_len;
     unsigned      field_flag;
```

`textsw_erase()`

Returns `0` if the operation fails. Equivalent to `textsw_delete()`, but does not affect the global shelf.

```
Textsw_index
textsw_erase(textsw, first, last_plus_one)
     Textsw       textsw;
     Textsw_index first, last_plus_one;
```

`textsw_file_lines_visible()`

Fills in `top` and `bottom` with the file line indices of the first and last file lines being displayed in `textsw.`

```
void
textsw_file_lines_visible(textsw, top,
bottom)
     Textsw textsw;
     int    *top, *bottom;
```

`textsw_find_bytes()`

Beginning at the position addressed by `first`, searches for the pattern specified by `buf` of length `buf_len`. Searches forwards if `flags` is `0`, else searches backwards. Returns `-1` if no match, else matching span placed in indices addressed by `first` and `last_plus_one`.

```
int
textsw_find_bytes(textsw, first,
last_plus_one, buf, buf_len, flags)
     Textsw        textsw;
     Textsw_index *first, *last_plus_one;
     char         *buf;
     unsigned      buf_len, flags;
```

`textsw_find_mark()`

Returns the current position of `mark.` If this operation fails, it will return `TEXTSW_INFINITY`.

```
Textsw_index
textsw_find_mark(textsw, mark)
     Textsw      textsw;
     Textsw_mark mark;
```

2.12 Textsw Procedures and Macros

`textsw_add_mark()`

Adds a new mark at `position`. `flags` can be either `TEXTSW_MARK_DEFAULTS` or `TEXTSW_MARK_MOVE_AT_INSERT`.

```
Textsw_mark
textsw_add_mark(textsw, position, flags)
     Textsw         textsw;
     Textsw_index   position;
     unsigned       flags;
```

`textsw_append_file_name()`

Returns `0` if `textsw` is editing a file, and if so appends the name of the file at the end of `name`.

```
int
textsw_append_file_name(textsw, name)
      Textsw  textsw;
      char    *name;
```

`textsw_delete()`

Returns `0` if the operation fails. Removes the span of characters beginning with `first`, and ending one before `last_plus_one`.

```
Textsw_index
textsw_delete(textsw, first, last_plus_one)
     Textsw textsw;
     Textsw_index first, last_plus_one;
```

`textsw_edit()`

Returns `0` if the operation fails. Erases a character, word or line, depending on whether `unit` is `SELN_LEVEL_FIRST`, or `SELN_LEVEL_LINE`. If `direction` is `0`, characters after the insertion point are affected, otherwise characters before the insertion point are affected. The operation will be done `count` times.

```
Textsw_index
textsw_edit(textsw, unit, count, direction)
     Textsw   textsw;
     unsigned unit, count, direction;
```

`selection_request()`

The low-level (policy-independent) mechanism for retrieving information about a selection from the server. Most clients will access it only indirectly, through `selection_ask` or `selection_query`.

`selection_request` takes a pointer to a holder (as returned by `seln_inquire`), and a request constructed in `*buffer`. The request is transmitted to the indicated selection holder, and the buffer rewritten with its response. Failures in the RPC mechanism will cause a `SELN_FAILED` return; if the process of the addressed holder is no longer active, the return value will be `SELN_NON_EXIST`.

Clients which call `selection_request` directly will find it most convenient to initialize the buffer by a call to `selection_init_request`.

Request attributes which are not recognized by the selection holder will be returned as the value of the attribute `SELN_UNRECOGNIZED`. Responses should be provided in the order requests were encountered.

```
Seln_result
selection_request(server, holder, buffer)
     Xv_Server             server
     Seln_holder          *holder;
     Seln_request         *buffer;
```

`selection_yield_all()`

This procedure queries the holders of all selections, and for each which is held by a client in the calling process, sends a yield request to that client and a `Done` to the server. It should be called by applications which are about to exit or to undertake lengthy computations during which they will be unable to respond to requests concerning selections they hold.

```
void
selection_yield_all()
```

`selection_query()`

A request is transmitted to the selection holder indicated by the `holder` argument. `consume` and `context` are used to interpret the response, and are described below. The remainder of the arguments to `selection_query` constitute an attribute value list which is the request. (The last argument should be a `0` to terminate the list.) The procedure pointed to by `reader` will be called repeatedly with a pointer to each buffer of the reply. The value of the `context` argument will be available in `buffer->requester_data.context` for each buffer. This item is not used by the selection library; it is provided for the convenience of the client. When the reply has been completely processed (or when the `consume proc` returns something other than `SELN_SUCCESS`), `selection_query` returns.

```
Selection_result
selection_query(server, holder,
   reader, context, <attributes>, ... 0)
     Xv_Server          server;
     Seln_holder       *holder;
     Seln_result      (*reader)();
     char *            *context;
     A-V list           attributes;
```

`selection_report_event()`

This is a high-level procedure for informing the server of a function key transition which may affect the selection. It incorporates some of the policy of the standard user interface, and provides a more convenient interface to `selection_inform`.

`Seln_client_node` is the client handle returned from `selection_create`; it may be `0` if the client guarantees it will not need to respond to the function transition.

`Event` is a pointer to the `struct inputevent` which reports the transition `seln_report_event`.

`selection_report_event` generates a corresponding call to `seln_inform` and, if the returned `struct` is not `null`, passes it to the client's function_proc callback procedure.

```
void
selection_report_event(server, client, event)
     Xv_Server              server;
     Seln_client_node      *client;
     Event                 *event;
```

`selection_init_request()`

This procedure is used to initialize a buffer before calling `selection_request`. (It is also called internally by `selection_ask` and `seln_query`.) It takes a pointer to a request buffer, a pointer to a `struct` referring to the selection holder to which the request is to be addressed, and a list of attributes which constitute the request to be sent. The attributes are copied into `buffer->data`, and the corresponding size is stored into `buffer->buf_size`. Both elements of `requester_data` are zeroed; if the caller wants to handle long requests, consumer-proc and context pointers must be entered in these elements after `selection_init_request` returns.

```
void
selection_init_request(server, buffer,
holder, <attributes>, ... 0)
     Xv_Server              server;
     Selection_request     *buffer;
     Seln_holder           *holder;
     char                  *attributes;
```

`selection_inquire()`

Returns a `Seln_holder struct` containing information which enables the holder of the indicated selection to be contacted. If the `rank` argument is `SELN_UNSPECIFIED`, the server will return access information for either the primary or secondary selection holder, as warranted by the state of the function keys it knows about. The `rank` element in the returned struct will indicate which is being returned.

This procedure may be called without `selection_create()` having been called first.

```
Seln_holder
selection_inquire(server, rank)
     Xv_Server  server;
     Seln_rank  rank;
```

`selection_inquire_all()`

A `Seln_holders_all struct` is returned from the Selection Service; it consists of a `Seln_holder struct` for each of the four ranks.

```
Seln_holders_all
selection_inquire_all()
```

`selection_hold_file()`

The server is requested to act as the holder of the specified rank, whose ASCII contents have been written to the file indicated by path. This allows a selection to persist longer than the application which made it can maintain it. Most commonly, this will be done by a process which holds the shelf when it is about to terminate.

```
Seln_result
selection_hold_file(server, rank, path)
     Xv_Server        server;
     Seln_rank        rank;
     char            *path;
```

`selection_inform()`

This is the low-level, policy-independent procedure for informing the server that a function key has changed state. Most clients will prefer to use the higher-level procedure `seln_report_event`, which handles much of the standard interpretation required.

```
Seln_function_buffer
selection_inform(server, client, which, down)
     Xv_Server        server;
     Seln_client      client;
     Seln_function    which;
     int              down;
```

`selection_destroy()`

A client created by `selection_create` is destroyed: any selection it may hold is released and various pieces of data associated with the selection mechanism are freed. If this is the last client in this process using the Selection Service, the RPC socket is closed and its notification removed.

```
void
selection_destroy(server, client)
     Xv_Server       server;
     Seln_client     client;
```

`selection_done()`

Client indicates it is no longer the holder of the selection of the indicated rank. The only cause of failure is absence of the server. It is not necessary for a client to call this procedure when it has been asked by the server to yield a selection.

```
Seln_result
selection_done(server, client, rank)
     Xv_Server    server;
     Seln_client  client;
     Seln_rank    rank;
```

`selection_figure_response()`

A procedure to determine the correct response according to the standard user interface when `seln_inform()` returns `*buffer` , or the client's `function_procs` called with it. The `addressee_rank` field in `Seln_function_buffer` will be modified to indicate the selection which should be affected by this client; `holder` will be set to point to the element of `*buffer` which should be contacted in the ensuing action, and the return value indicates what that action should be. Possible return values are `SELN_IGNORE`, `SELN_REQUEST`, `SELN_FIND`, `SELN_SHELVE`, and `SELN_DELETE`.

```
Seln_response
selection_figure_response(
  server, buffer, holder)
     Xv_Server              server;
     Seln_function_buffer  *buffer;
     Seln_holder           *holder;
```

`selection_ask()`

`selection_ask()` is a simplified form of `selection_request()` that looks and acts very much like `seln_query()`. The only difference is that it does not use a callback `proc`, and so cannot handle replies that require more than a single buffer. If it receives a reply consisting of more than one buffer, it returns the first buffer and discards the rest. The return value is a pointer to a static buffer; in case of error, this will be a valid pointer to a null buffer, `buffer->status = SELN_FAILED`.

```
Seln_request *
selection_ask(server, holder, <attributes>,
... 0)
     Xv_Server        server;
     Seln_holder     *holder;
     Attr_union       attributes;
```

`selection_clear_functions()`

The server is told to forget about any function keys it thinks are down, resetting its state to all-up. If it knows of a current secondary selection, the server will tell its holder to yield.

```
void
selection_clear_functions()
```

`selection_create()`

The server is initialized for this client. `Client_data` is a 32-bit opaque client value which the server will pass back in callback procedures, as described above. The first two arguments are addresses of client procedures which will be called from the selection functions when client processing is required. These occasions occur when the server sees a function-key transition which may interest this client, and when another process wishes to make a request concerning the selection this client holds.

```
Seln_client
selection_create(
     server, function_proc, request_proc,
     client_data)
          Xv_Server,   server;
          void        (*function_proc) ();
          Seln_result (*request_proc) ();
          Xv_opaque    client_data;
```

2.10 Scrollbar Procedures and Macros

`scrollbar_default_compute_scroll_proc()`

Computes default scrolling based on scroll position. `pos` is the position from the start of the cable event. `length` is the length of the cable. `offset` and `object_length` are output parameters. The offset is given in client units to scroll by. The object length is in client units.

```
void
scrollbar_default_compute_scroll_proc(
  scroll_public, pos, available_cable,
  motion, offset, object_length)
     Scrollbar        scroll_public
     Scrollbar        scroll_public
     int              pos;
     int              available_cable;
     Scroll_motion    motion;
     unsigned long    *offset;
     unsigned long    *object_length;
```

`scrollbar_paint()`

Repaints all portions of the scrollbar.

```
void
scrollbar_paint(scrollbar)
     Scrollbar scrollbar;
```

2.11 Selection Procedures and Macros

`selection_acquire()`

`client` is the opaque handle returned from `selection_create()`. The client uses this call to become the new holder of the selection of rank

`asked`. `asked` should be one of `SELN_CARET, SELN_PRIMARY, SELN_SECONDARY, SELN_SHELF`, or `SELN_UNSPECIFIED`. If successful, the rank actually acquired is returned.

If `asked` is `SELN_UNSPECIFIED`, the client indicates it wants whichever of the primary or secondary selections is appropriate given the current state of the function keys; the one acquired can be determined from the return value.

```
Seln_rank
selection_acquire(server, client, asked)
     Xv_Server    server;
     Seln_client  client;
     Seln_rank    asked;
```

`panel_paint()`

Paints an item or an entire panel. `paint_behavior` can be either `PANEL_CLEAR`, which causes the area occupied by the panel or item to be cleared prior to painting, or `PANEL_NO_CLEAR`.

```
int
panel_paint(panel_object, paint_behavior)
     <Panel_item or Panel> panel_object;
     Panel_setting paint_behavior;
```

`panel_text_notify()`

Default notify procedure for panel text items. Causes caret to advance on `<Return>` or `<Tab>`, caret to back up on `<Shift-Return>` or `<Shift-Tab>`, printable characters to be inserted into `item's` value, and all other characters to be discarded.

```
Panel_setting
panel_text_notify(item, event)
     Panel_item item
     Event      *event
```

`PANEL_TOGGLE()`

A macro for `"PANEL_CHOICE, PANEL_CHOOSE_ONE, FALSE"`. Used to create non-exclusive choice item(s) similar to SunView toggle items.

```
xv_create(panel, PANEL_TOGGLE, 0);
```

`panel_default_handle_event()`

The default event procedure for panel items `PANEL_EVENT_PROC` and for the panel's background `(PANEL_BACKGROUND_PROC)`. Implements the standard event-to-action mapping for the item types.

```
panel_default_handle_event(object, event)
     <Panel or Panel_item> object;
     Event *event;
```

`PANEL_CHECK_BOX()`

Macro for "`PANEL_TOGGLE, PANEL_FEEDBACK, PANEL_MARK`". Creates non-exclusive choice item(s) with `check_boxes` instead of boxes.

```
xv_create(panel, PANEL_CHECK_BOX, 0);
```

`PANEL_CHOICE_STACK()`

Macro for "`PANEL_CHOICE, PANEL_DISPLAY_LEVEL, PANEL_CURRENT`". Creates an `OPEN LOOK` abbreviated menu button.

```
xv_create(panel, PANEL_CHOICE_STACK, 0);
```

`PANEL_EACH_ITEM()`

Macro to iterate over each item in a panel. The corresponding macro `panel_end_each` closes the loop opened by `PANEL_EACH_ITEM()`

```
PANEL_EACH_ITEM(panel, item)
     Panel panel;
     Panel_item item;
```

- NOTICE_YES if a confirm button or trigger was pushed
- NOTICE_NO if a cancel button or trigger was pushed
- NOTICE_FAILED if the notice failed to pop up.

```
int
notice_prompt(window, event, attributes)
     XV_Window  window;
     Event      *event;
     <attribute-list> attributes;
```

2.8 Openwin Procedures and Macros

OPENWIN_EACH_VIEW()

A macro providing built-in support for iteration across all the views contained in a given openwin. It allows you to perform operations on multiple views for which there are no openwin attributes.

OPENWIN_END_EACH()

Closes the loop started by OPENWIN_EACH_VIEW(). These macros are meant to be used together, as in the following example:

```
OPENWIN_EACH_VIEW(openwin, view)
        Openwin       openwin;
        Openwin_item  item;
        xv_set(openwin, attributes, 0);
OPENWIN_END_EACH();
```

2.9 Panel Procedures and Macros

panel_advance_caret()

Advance the caret to the next text item. If on the last text item, rotate back to the first. Returns the new caret item, or NULL if there are no text items.

```
Panel_item
panel_advance_caret(panel)
     Panel panel;
```

panel_backup_caret()

Back the caret up to the previous text item. If already on the first text item, rotate back to the last. Returns the new caret item, or NULL if there are no text items.

```
Panel_item
panel_backup_caret(panel)
     Panel panel;
```

`menu_return_value()`

Predefined notify procedure which, if given as the value for `MENU_NOTIFY_PROC`, causes the `menu_done_proc()` (if any) to return the value of the selected item rather than the item itself.

```
void
menu_return_value(menu, menu_item)
     Menu menu;
     Menu_item menu_item;
```

`menu_show()`

Displays `menu` and gets a selection from the user. However, unlike SunView, it does not return a value; neither does `menu_show` returning mean that the menu has been dismissed.

`window` is the handle of the window over which the menu is displayed.

`event` is the event which causes the menu to come up. The final `0` is required even though the final argument is currently ignored.

```
void
menu_show(menu, window, event, 0)
    Menu menu;
    Xv_Window window;
    Event *event;
```

2.7 Notice Procedures and Macros

`notice_prompt()`

Displays notice and does not return until the user pushes a button, or until its trigger or the default has been seen. It returns a value of `NOTICE_FAILED` if `notice_prompt()` fails for any reason; otherwise, it is equivalent to the ordinal value of the button which caused the return (i.e., button actually selected, or default button if default action triggered return). The `client_window` should be the window for which the notice has been generated. (This is important for fonts and positioning information. If a window object is not available, `NULL` may be passed and defaults used. For the correct positioning of the notice, clients should specify `NOTICE_FOCUS_XY`.) The event will be completely filled in at the time the `notice_prompt()` returns. The possible status values that may be returned from this function are:

- the `int()value` passed with every `NOTICE_BUTTON` attribute

2.5 Frame Procedures and Macros

`frame_get_rect()`

Gets the `rect` of the frame. `X,y` is the upper left corner of the window coordinate space. Width and height include the window manager decoration.

```
void
frame_get_rect(frame, rect)
    Frame        frame;
    Rect        *rect;
```

`frame_set_rect()`

Sets the `rect` of the frame. `X,y` is the upper left corner of the window coordinate space. Width and height include the window manager decoration.

```
void
frame_set_rect(frame, rect)
    Frame        frame;
    Rect        *rect;
```

2.6 Menu Procedures and Macros

`menu_action_proc()`

Procedure called when a menu item is selected.

```
menu_item
menu_action_proc(menu, menu_item)
```

`menu_return_item()`

Predefined notify procedure which, if given as the value for `MENU_NOTIFY_PROC`, causes the `menu_done_proc()` (if any) to return the selected item rather than its value.

```
void
menu_return_item(menu, menu_item)
    Menu menu;
    Menu_item menu_item;
```

If CANVAS_X_PAINT_WINDOW has been set to TRUE, (a requirement if you want to use direct X graphics calls to draw in an XView canvas) the repaint callback is called with the arguments below.

display is the display handle to the X11 server connection. xid is the X11 window identifier for the canvas paint window. The Xv_xrectlist structure returns an array of XRectangles and a count that specifies the repaint area for the canvas paint window.

```
void
canvas_reapint_proc(
 canvas, paint_window, display,
 xid, xrectlist)
       Canvas       canvas;
       Xv_Window    paint_window;
       Display      *display;
       Window       xid;
       Xv_xrectlist *xrects;
```

canvas_resize_proc()

Procedure called when canvas width or height changes.

```
void
canvas_resize_proc(canvas, width, height)
    Canvas  canvas;
    int     width;
    int     height
```

2.4 Cursor Procedures and Macros

cursor_copy()

Creates and returns a copy of src_cursor.

```
Xv_Cursor
cursor_copy(src_cursor)
    Xv_Cursor src_cursor;
```

`xv_send_message()`

Lets two separate processes communicate with each other. You can specify the `addressee` field either with one of the constants `XV_POINTER_WINDOW` or `XV_FOCUS_WINDOW`, or with the window's `XID`, if known.

```
void
xv_send_message(
 window, addressee, msg_type, format, data, len)
      Xv_object  window;
      Xv_opaque  addressee;
      char       *msg_type;
      int        format;
      Xv_opaque  *data;
      int        len;
```

`XV_SERVER_FROM_WINDOW()`

Macro to get at the handle to the server object from a window object.

```
Xv_opaque
XV_SERVER_FROM_WINDOW(window)
    Xv_Window window;
```

2.3 Canvas Procedures and Macros

`canvas_paint_window()`

Returns the window to use when drawing into the canvas paint window.

```
Xv_window
canvas_paint_window(canvas)
    Canvas canvas;
```

`canvas_repaint_proc()`

Client-supplied callback procedure. Canvas package calls it when getting a `WIN_PAINT` event.

```
void
canvas_repaint_proc(
 canvas, paint_windowrepaint_area)
       Canvas      canvas;
       Xv_Window   paint_window;
       Rectlist    *repaint_area;
```

`xv_col()`

Returns integers representing the number of pixels, excluding the left margin of the window.

```
int
xv_col (window, column)
       Xv_Window    window
       int          column;
```

`xv_cols()`

Returns integers representing the number of pixels, including the left margin of the window.

```
int
xv_cols (window, columns)
       Xv_Window    window
       int          columns;
```

`xv_row()`

Returns integers representing the number of pixels, excluding the top margin of the window.

```
int
xv_row (window, row)
       Xv_Window    window
       int          row;
```

`xv_rows()`

Returns integers representing the number of pixels, including the top margin of the window.

```
int
xv_rows (window, rows)
       Xv_Window    window
       int          rows;
```

`XV_SCREEN_FROM_WINDOW()`

Macro to get at the handle to the screen object from a window object.

```
XV_SCREEN_FROM_WINDOW(window)
   Xv_Window window;
```

`xv_get()`

Retrieves the value of the specified attribute for the specified XView object.

```
Xv_opaque
xv_get(object, attribute)
      Xv_object object;
      <attribute-list> attributes;
```

`xv_init()`

Initializes the notifier, reads any passed attributes, initializes Defaults/Resource-Manager Database, loads Server Resource-Manager Database, and reads the `~/.Xdefaults` Database.

```
int
main(argc, argv)
      int   argc;
      char  **argv:
{
      <initialization/declarations>
   (void)xv_init(XV_INIT_ARGC_PTR_ARGV,
            &argc, argv, 0);
      frame = xv_create(NULL, FRAME,\
            FRAME_LABEL, "foo", 0);
}
```

`xv_set()`

Sets the value of one or more of the specified object's attributes. Here `attributes` is a null-terminated list of attributes and their associated values.

```
Xv_object
Xv_set(owner, type, attributes)
      Xv_object owner;
      <vu type> *type:
      <attribute-list> attributes;
```

2.2 Common Procedures and Macros

`XV_DISPLAY_FROM_WINDOW()`

Macro to get at the handle to the display object from a window object.

```
Xv_opaque
XV_DISPLAY_FROM_WINDOW(window)
   Xv_Window window;
```

2

XView Procedures and Macros

This chapter presents the procedures and macros belonging to each XView object. The lists of procedures are arranged in alphabetical order by object. See the Index for the page number of any particular procedure.

Each entry consists of the procedure's name, a brief description of its purpose, and its declaration.

2.1 Generic Procedures

`xv_create()`

Creates an XView object and returns a handle to that object.

```
Xv_opaque
xv_create(owner, package, attributes)
      Xv_object owner;
      Xv_pkg *package;
      <attribute-list> attributes;
```

`xv_destroy()`

Destroys an XView object and any subframes owned by that object.

```
int
xv_destroy(object)
      Xv_object object;
```

`xv_find()`

Looks for the object and returns its handle. If the object does not exist, it will be created.

```
Xv_opaque
xv_find(owner, package, attributes)
      Xv_object owner;
      Xv_pkg *package;
      <attribute-list> attributes;
```