Appendix A: Attributes

ARRAY_TILE_ALIGN

Controls the positioning of children within the column and rows.

Argument: Array_tile_align
```
        ARRAY_TILE_ALIGN_NORTHWEST,
        ARRAY_TILE_ALIGN_NORTH,
        ARRAY_TILE_ALIGN_NORTHEAST,
        ARRAY_TILE_ALIGN_WEST,
        ARRAY_TILE_ALIGN_CENTER,
        ARRAY_TILE_ALIGN_EAST,
        ARRAY_TILE_ALIGN_SOUTHWEST,
        ARRAY_TILE_ALIGN_SOUTH,
        ARRAY_TILE_ALIGN_SOUTHEAST
```
Default: ARRAY_TILE_ALIGN_CENTER
Procs: create, get, set

ARRAY_TILE_AUTO_POSITION

If TRUE, children are positioned automatically when they are added and deleted. Children are positioned so that there are no holes in the array_tile. It is useful to set this to FALSE if you have specific positioning needs and are explicitly positioning children with the other array_tile attributes.

Argument: Boolean
Default: TRUE
Procs: create, set, get

ARRAY_TILE_COLUMN

The column index of the object (child) referenced.

Argument: object, int
Default: n/a
Procs: get, set
Usage:
```
xv_set(array_tile,
       ARRAY_TILE_COLUMN, child, 12,
       NULL);

child_column = (int) xv_get(array_tile, ARRAY_TILE_COLUMN, child);
```

ARRAY_TILE_COLUMN_GAP

The horizontal distance in pixels between the columns of the array_tile.

Argument: int
Default: 15
Procs: create, get, set

ARRAY_TILE_COLUMN_WIDTH

The horizontal space in pixels allocated to columns. All columns in an array_tile are of the same size. Adding a child that is wider than this value causes this value to increase to fit the child within the column.

Argument: int
Default: 40
Procs: create, get, set

ARRAY_TILE_HLINES

Display horizontal lines around the children.

Argument:	Boolean
Default:	FALSE
Procs:	create, set, get

ARRAY_TILE_LAYOUT

This controls automatic layout of objects when they are added to an array_tile. It also controls the growth of the array_tile when all positions are taken.

If set to ARRAY_TILE_LAYOUT_COLUMN, new columns will be added when the array_tile is full. If set to ARRAY_TILE_LAYOUT_ROW, new rows will be added when the array_tile is full. When not full, a newly added object will be positioned at the first available spot.

Argument:	Array_tile_layout
	`ARRAY_TILE_LAYOUT_ROW,`
	`ARRAY_TILE_LAYOUT_COLUMN`

Default:	ARRAY_TILE_LAYOUT_ROW
Procs:	create, get, set

ARRAY_TILE_N_COLUMNS

The number of columns in the array_tile.

Argument:	int
Default:	4
Procs:	create, get, set

ARRAY_TILE_N_ROWS

The number of rows in the array_tile.

Argument:	int
Default:	1
Procs:	create, get, set

ARRAY_TILE_POSITION

The row and column of the object (child) referenced.

The array_tile does not allow two children to be placed at the same position. If this happens, the object previously at that spot will be reposititition to (-1, -1) and will not be visible.

Argument:	object, int, int
Default:	n/a
Procs:	get, set

Usage:

```
xv_set(array_tile,
       ARRAY_TILE_POSITION, child, 4, 5,
       NULL);

child = (Rectobj) xv_get(array_tile, ARRAY_TILE_POSITION, 7, 6);
```

ARRAY_TILE_ROW

The row index of the object (child) referenced.

Argument: object, int
Default: n/a
Procs: get, set
Usage:

```
xv_set(array_tile,
       ARRAY_TILE_ROW, child, 7,
       NULL);

child_row = (int) xv_get(array_tile, ARRAY_TILE_ROW, child);
```

ARRAY_TILE_ROW_GAP

The vertical distance in pixels between the rows of the array_tile.

Argument: int
Default: 15
Procs: create, get, set

ARRAY_TILE_ROW_HEIGHT

The vertical space in pixels allocated to rows. All rows in an array_tile are of the same size. Adding a child that is wider than this value causes to increase to fit the child within the column.

Argument: int
Default: 20
Procs: create, get, set

ARRAY_TILE_VLINES

Display vertical lines around the children.

Argument: Boolean
Default: FALSE
Procs: create, set, get

BAG_ANCHORED

If TRUE, the bag's origin (i.e. its upper left edge) will remain fixed when children are added or change geometry. If FALSE, the upper left edge will dynamically change to shrink close to the upper leftmost child.

Argument: Boolean
Default: TRUE
Procs: create, get, set

BAG_AUTO_SHRINK

The bag always gets larger to keep expanding children within the bag's bounds. If this attribute is TRUE, the bag will reduce it's size when children shrink or are deleted.

Argument: Boolean
Default: FALSE
Procs: create, get, set

BOX_GAP

The gap in pixels between the children.

Argument:	int
Default:	1
Procs:	create, set, get

BOX_LAYOUT

The direction that the box lays it's children out.

Argument:	Box_layout
	`BOX_LAYOUT_HORIZONTAL, BOX_LAYOUT_VERTICAL`
Default:	BOX_LAYOUT_HORIZONTAL
Procs:	create, get, set

CANVAS_SHELL_AUTO_DROP_SITE

If this is TRUE, the canvas_shell will create and maintain a XView drop site object. This is necessary if you want RECTOBJ_DROP_PROC callbacks to operate.

Argument:	Boolean
Default:	FALSE
Procs:	create, get, set

CANVAS_SHELL_BATCH_REPAINT

When this is TRUE, an offscreen pixmap is used to buffer repaints. This reduces flashing considerable, at the cost of repaint speed and memory consumption.

If the server doesn't have enough memory, or if the user overrides this by setting the X resource *SlingShot.Batching* to FALSE, no pixmap will be created.

The size of the offscreen pixmap is reset to new dimensions whenever the canvas_shell is resized. This is done by the default CANVAS_RESIZE_PROC, canvas_shell_resize_proc.

Argument:	Boolean
Default:	FALSE
Procs:	create, get, set

CANVAS_SHELL_DELAY_REPAINT

At the end of each xv_set() call, changes to visual appearance of objects are flushed to the screen. In many circumstances, this can cause thrashing during repaint because multiple calls are made to xv_set(). This attribute is used to work around this problem.

Setting this attribute to TRUE will prohibit repainting until it is set to FALSE again. This can be set to TRUE several times, but it must be set to FALSE an equal number of times.

The canvas shell event handler uses this internally, so you usually won't need to use this in event callbacks such as RECTOBJ_DOUBLE_CLICK_PROC.

Argument:	Boolean
Default:	FALSE
Procs:	get, set

CANVAS_SHELL_DROP_SITE

Returns the drop site that is created if CANVAS_SHELL_AUTO_DROP_SITE is TRUE. You can get this attribute from the paint window only.

Default: N/A
Procs: get (on paint window)

CANVAS_SHELL_EVENT_PROC

The Canvas_shell uses the WIN_EVENT_PROC on each of it's paint windows to track events. If you need to get the event before it is processed by the canvas_shell, use this attribute. Return TRUE if the event should not be further passed onto the rectobjs within the canvas_shell. A typical use of this may be to handle ACTION_CUT, ACTION_COPY, and ACTION_PASTE events.

Argument: int (*)()
Default: NULL
Procs: create, get, set
Usage:

```
int miscellaneous_event_proc(
      Xv_window    paint_window,
      Event        *event,
      Canvas_shell canvas_shell)
```

CANVAS_SHELL_FONT

The default font for Drawtext and Drawarea objects for this Canvas_shell. Use in case you need a default font that is different from the font from the frame.

Argument: Xv_font
Default: The XV_FONT of the frame
Procs: create

CLOCKOBJ_HR

The hour to display. Value ranges from 1 to 12.

Argument: int
Default: 12
Procs: create, get, set

CLOCKOBJ_MIN

The minute to display. [0..59]

Argument: int
Default: 0
Procs: create, get, set

CLOCKOBJ_HR_DRAWAREA
CLOCKOBJ_MIN_DRAWAREA

The drawarea uses a separate object to draw the hour and min hand into. These attribute returns the handles of these objects so the colors or callbacks can be set.

Argument: Drawarea
Default: n/a
Procs: get

CLOCKOBJ_MOVABLE

Allows the user to set the time by clicking and dragging on either of the hands of the clock.

Argument: Boolean
Default: FALSE
Procs: create, get, set

CLOCKOBJ_MOVE_PROC

A notification callback for the application. This is called after the user has changed the time by clicking and dragging of the clock hands. The done parameter is TRUE when the mouse has been released.

Argument: Proc_ptr
Default: NULL
Procs: create, get, set
Usage:

```
void clockobj_time_changed(
        Clockobj      clockobj,
        int           new_hour,
        int           new_min,
        int           done)
```

DRAWAREA_LAST_MARK_KEY

Returns the last key that was specified when an event was mapped to a display list entry. It is only useful to get this in an event callback.

Argument: Xv_opaque
Default: n/a
Procs: get

DRAWAREA_LEFT_X
DRAWAREA_RIGHT_X
DRAWAREA_UPPER_Y
DRAWAREA_LOWER_Y

Configures the dimensions of the drawarea.

Argument: double *
Default: (0, 0, 10000, 10000)
Procs:
Usage:

```
double current_value, new_value;

current_value = *(double*) xv_get(drawarea, DRAWAREA_LEFT_X);

new_value = 1.5;
xv_set(drawarea,
        DRAWAREA_UPPER_X, &new_value,
        NULL);
```

DRAWAREA_MAP_EVENTS

Controls how events are mapped to display list entries.

Argument: Drawarea_map_event_mode
Default: DRAWAREA_MAP_ANY
Procs: create, get, set

DRAWAREA_MIN_ALLOC

As drawing commands are issued, the display list grows. The minimum amount of memory used by the display list can be controlled with this attribute, this can improve performance for lengthy display lists. The current length will be used as the minimum if this is set to zero.

Argument:	int
Default:	100
Procs:	create, set, get

DRAWICON_GAP

The amount of space, in pixels, between the image and the text.

Argument:	int
Default:	1
Procs:	create, get, set

DRAWICON_IMAGE

Returns the Drawimage object owned by the Drawicon. This is useful for setting attributes such as color or callback procs on the image of a Drawicon.

Argument:	
Default:	n/a
Procs:	get
Usage:	

```
image = (Drawimage) xv_get(drawimage, DRAWICON_IMAGE);
```

DRAWICON_IMAGE_COLOR

Sets the RECTOBJ_FG of the Drawicon's image.

Argument:	int
Default:	-1
Procs:	create, get, set

DRAWICON_LAYOUT_VERTICAL

Sets the layout of the Drawicon. If TRUE, the text is below the image. If FALSE, the text is to the right of the image.

Argument:	Boolean
Default:	TRUE
Procs:	create, get, set

DRAWICON_TEXT

Returns the Drawtext object owned by the Drawicon. This is useful for setting attributes such as color or callback procs on the text of a Drawicon.

Argument:	
Default:	n/a
Procs:	get
Usage:	

```
text = (Drawtext) xv_get(drawicon, DRAWICON_TEXT);
```

DRAWICON_TEXT_COLOR

Sets the RECTOBJ_FG of the Drawicon's text.

Argument: int
Default: -1
Procs: create, get, set

DRAWIMAGE_IMAGE1 (DRAWIMAGE_IMAGE)

The XView Server image to be displayed when not selected.

Argument: Server_image
Default: NULL
Procs: create, get, set

DRAWIMAGE_IMAGE1_MASK

The mask for DRAWIMAGE_IMAGE1. The image is not painted and events are not recognized where bits are off in the mask.

Argument: Server_image
Default: NULL
Procs: create, get, set

DRAWIMAGE_IMAGE2 (DRAWIMAGE_HIGHLIGHT_IMAGE)

The XView server image to be displayed when selected.

Argument: Server_image
Default: NULL
Procs: create, get, set

DRAWIMAGE_IMAGE2_MASK

The mask for DRAWIMAGE_IMAGE2. The image is not painted and events are not recognized where bits are off in the mask.

Argument: Server_image
Default: NULL
Procs: create, get, set

DRAWLINE_ARROW_ANGLE

The angle of the tip of the arrow, measured in degrees * 64, maximum is 360*64.

Argument: int
Default: 60 * 64
Procs: create, get, set

DRAWLINE_ARROW_INSET_LENGTH

The length from the arrow tip to the "neck" of the arrowhead. Measured in pixel dimensions.

Argument: int
Default: 7
Procs: create, get, set

DRAWLINE_ARROW_LENGTH

The length of the lines along the cutting edge of the arrowhead, in pixels.

Argument:	int
Default:	10
Procs:	create, get, set

DRAWLINE_ARROW_STYLE

The style for the arrowhead.

Argument:	ARROW_FILLED, ARROW_HOLLOW, ARROW_SIMPLE, or ARROW_NONE.
Default:	ARROW_NONE
Procs:	create, get, set

DRAWLINE_X
DRAWLINE_Y

Set and get the endpoints of the line relative to the parent.

Argument:	int pair (endpoint [0,1], value)
Default:	N/A
Procs:	create, get, set
Usage:	When used with xv_get() one arg indicates which endpoint to retrieve.

DRAWLINE_X0
DRAWLINE_X1
DRAWLINE_Y0
DRAWLINE_Y1

These four attributes are functionally equivalent to DRAWLINE_X and DRAWLINE_Y, except that they are positioned relative to the coordinate space of the canvas shell paint window.

Argument:	int
Procs:	create, get, set

DRAWOBJ_FILLED (DRAWOBJ_OPAQUE)

If TRUE, the interior is filled.

Argument:	Boolean
Default:	TRUE
Procs:	create, get, set

DRAWRECT_BORDER1

The distance (in pixels) from the object's XV_RECT to the illustrated border.

Argument:	int
Default:	0
Procs:	create, get, set

DRAWRECT_BORDER2

The size of the illustrated border.

Argument:	int
Default:	1
Procs:	create, get, set

DRAWRECT_BORDER3

The space between the illustrated border and the children.

Argument:	int
Default:	0
Procs:	create, get, set

DRAWTEXT_EDITABLE

If TRUE, the drawtext can be edited.

Argument:	Boolean
Default:	FALSE
Procs:	create, get, set

DRAWTEXT_FONT

The font used to display the object. The minimum size of the Drawtext is determined by the font and the string.

Argument:	Xv_Font
Default:	CANVAS_SHELL_FONT
Procs:	create, get, set

DRAWTEXT_JUSTIFY

When the XV_WIDTH of the drawtext is set to be larger than the drawtext object, this controls how it is positioned within the available space.

Argument:	Drawtext_justify_style
	`DRAWTEXT_JUSTIFY_LEFT`
	`DRAWTEXT_JUSTIFY_CENTER`
	`DRAWTEXT_JUSTIFY_RIGHT`

Default:	DRAWTEXT_JUSTIFY_LEFT
Procs:	create, get, set

DRAWTEXT_LENGTH

Set the width to allow for this number of characters. The string is justified within this space.

Argument:	int
Default:	0
Procs:	create, get, set

DRAWTEXT_SHOW_UNDERLINE

If TRUE, a line indicating that this object is editable will be shown even when the drawtext does not have the focus.

Argument:	Boolean
Default:	FALSE
Procs:	create, get, set

DRAWTEXT_STRING

The string to be displayed. The Drawtext allocates space for, and copies the string.

Argument:	char *
Default:	NULL
Procs:	create, get, set

DRAWTEXT_STRING_PTR

Alternate method of specifing a string for a drawtext. It has the same effect as DRAWTEXT_STRING except that the string is not copied into allocated space.

Argument: char *
Default: NULL
Procs: create, get ,set

GRIP_BTN_DOWN_X
GRIP_BTN_DOWN_Y

The position of the button down event that started the drag. This is legit while the object is being dragged.

Argument: int
Default: n/a
Procs: get

GRIP_DONE_PROC

The function to be called when the user releases the mouse button.

Argument: void (*)()
Default: NULL
Procs: create, get, set
Usage:

```
int done_callback(
        Xv_window     paint_window,
        Event         *event,
        Canvas_shell  canvas_shell,
        Grip          grip,
        short         x,
        short         y)
```

GRIP_EXCEED_PARENT_DIMS

If FALSE, the dimensions of the grip's parent will not exceed that of it's parent while it is being dragged.

Argument: Boolean
Default: FALSE
Procs: create, get, set

GRIP_IMMEDIATE

If TRUE, the grip does not wait for the user to move beyond the X resource value *SlingShot.ClickMoveThreshold* before the drag begins.

Argument: Boolean
Default: FALSE
Procs: create, get, set

GRIP_MAX_X
GRIP_MAX_Y
GRIP_MIN_X
GRIP_MIN_Y

Sets the maximum and minimum dimensions the the object can be positioned at. The defaults are sufficiently large that they will be ignored if GRIP_EXCEED_PARENT_DIMS is FALSE. The position of the grip is compared using the origin of the grip, as a result the max attributes do not include the width and height of the grip.

Argument:	int
Default:	0, 0, 32000, 32000
Procs:	create, get, set

GRIP_MOVE_PROC

The function that gets called for each move event. This function returns TRUE if the object should be moved to the new location (*new_x, *new_y). It may also reset the coordinates of the event by setting new_x and new_y. new_x and new_y are the origin of the object relative to its owner.

Argument:	int (*)();
Default:	grip_default_event_proc
Procs:	create, get, set
Usage:	

```
int move_callback(
        Xv_window     paint_window,
        Event         *event,
        Canvas_shell  canvas_shell,
        Grip          grip,
        short         *new_x,
        short         *new_y)
```

GRIP_RUBBER_STYLE

How the object is animated while the user is moving it. The various styles are bounding boxes, vertical and horizontal lines, and crosshairs.

Argument:	Grip_rubber_style
	`GRIP_RUBBER_NONE`
	`GRIP_RUBBER_RECT,`
	`GRIP_RUBBER_VLINE,`
	`GRIP_RUBBER_VLINE_PAIR,`
	`GRIP_RUBBER_HLINE,`
	`GRIP_RUBBER_HLINE_PAIR,`
	`GRIP_RUBBER_CROSSHAIRS`

Default:	GRIP_RUBBER_NONE
Procs:	create, get, set

GRIP_SLIDE_X
GRIP_SLIDE_Y

If TRUE, permits the grip to be moved in that direction.

Argument:	Boolean
Default:	TRUE
Procs:	create, get, set

RECTOBJ_ACCEPTS_DROP

If FALSE, the object's RECTOBJ_DROP_PROC will not be called when a drop event has occured on it.

Argument: Boolean
Default: TRUE
Procs: create, get, set

RECTOBJ_ADD_CHILD_PROC

The callback function that is called when a child has been created or added to an rectobj (by setting XV_OWNER). Usually this function sets up data structures to manage the child, then returns. The geometry manage proc will be called right after this callback returns.

Argument: Proc_ptr
Default: depends on class
Procs: create, get, set
Usage:

```
void    add_child_proc(
                Rectobj         parent,/*self*/
                Rectobj         child,
                Rect            *rect)
```

RECTOBJ_BG

The background color for a RECTOBJ, an index into the cms. Setting it to -1 sets the default.

Argument: int
Default: -1 (The Canvas_shell's WIN_BACKGROUND_COLOR.)
Procs: create, get, set

RECTOBJ_BORDER

Used by Rectobj subclasses that support borders.

Argument: int
Default: 0
Procs: create, get, set

RECTOBJ_CANVAS

The Canvas_shell that an object is painted on.

Argument: n/a
Default: n/a
Procs: get
Usage:

```
canvas_shell = (Canvas_shell) xv_get(rectobj, RECTOBJ_CANVAS);
```

RECTOBJ_CHILDREN

The list of children for an object. This does not return a copy of the list, so it should be copied if children are to be destroyed while traversing this list.

Argument: Rectobj_list *
Default: NULL
Procs: get

RECTOBJ_DBL_CLICK_PROC

The callback function that is called when an object is double clicked on. The callback is invoked when the mouse button goes up. This callback is only invoked by the rectobj_selection_event_proc event handling routine.

Argument: Proc_ptr
Default: NULL
Procs: create, get, set
Usage:

```
void  double_click_proc(
      Xv_window     paint_window,
      Event         *event,
      Canvas_shell  canvas_shell,
      Rectobj       rectobj);
```

RECTOBJ_DEL_CHILD_PROC

The callback function that is called when a child is destroyed or removed from a rectobj (by setting XV_OWNER). This function is used to clean up data structures associated with the child.

Argument: Proc_ptr
Default: depends on class
Procs: create, get, set
Usage:

```
void  del_child_proc(
             Rectobj       parent,/*self*/
             Rectobj       child,
             Rect          *oldrect);
```

RECTOBJ_DRAGGABLE

If FALSE, the object's RECTOBJ_START_DRAG_PROC will not be called when the mouse pointer is dragged beyond the threshold distance (*SlingShot.ClickMoveThreshold*).

Argument: Boolean
Default: TRUE
Procs: create, get, set

RECTOBJ_DROP_PROC

The callback function that is called when an object receives an ACTION_DRAG_PREVIEW, ACTION_DRAG_COPY, ACTION_DRAG_LOAD, ACTION_DRAG_LOAD event. For ACTION_DRAG_PREVIEW events, the callback function should examine the event_id to determine if the event was a LOC_WINENTER or a LOC_WINEXIT.

```
void  drop_proc(
      Xv_window     paint_window,
      Event         *drag_event,
      Canvas_shell  canvas_shell,
      Rectobj       rectobj);
```

RECTOBJ_EVENT_PROC

The event handling function for an object. When an event occurs on an object, the event is passed to the object via this callback. Applications may change this attribute to one of the predefined callbacks (rectobj_selection_event_proc, rectobj_toggle_event_proc, rectobj_button_event_proc, rectobj_background_event_proc). Applications that need special event handling not provided by the provided callbacks may wish to write their own callback functions, or interpose by calling these functions from their own callback functions.

Argument: Proc_ptr
Default: rectobj_event_proc (also depends on class)
Procs: create, get, set
Usage:

```
void   event_proc(
       Xv_window     paint_window,
       Event         *event,
       Canvas_shell  canvas_shell
       Rectobj       rectobj);
```

RECTOBJ_EXCLUSIVE_SELECT

If TRUE, when this object is selected, all other selected objects will be deselected.

Argument: Boolean
Default: FALSE
Procs: create, get, set

RECTOBJ_FG

The foreground color for a RECTOBJ, an index into the cms. Setting it to -1 sets the default.

Argument: int
Default: -1 (The Canvas_shell's WIN_FOREGROUND_COLOR.)
Procs: create, get, set

RECTOBJ_GEOMETRY_SILENT

If this is TRUE, the parent will not be informed if the geometry of the rectobj has changed. This is useful for classes that explicitly set the positions of children and are not interested in size changes.

Argument: Boolean
Default: FALSE
Procs: create, get, set

RECTOBJ_MAP_EVENT_PROC

The callback function that traverses the rectobj hierarchy to map events to rectobjs. If a match is found, the object that matches should be returned.

Argument: Rectobj (*)()
Default: rectobj_map_event_proc (also depends on class)
Procs: create, get, set
Usage:

```
Rectobj      map_event_proc(
                    Rectobj      rectobj,
                    Event        *event);
```

RECTOBJ_MENU

The XView Menu object associated with an object. The menu appears when the user clicks the right mouse button. There may be several menus associated with a hierarchy of a user interface, the menu that is closest to the leaf object will be displayed.

Argument:	Menu
Default:	XV_NULL
Procs:	create, set, get

RECTOBJ_MENU_CLIENT

This attribute is set on a XView menu and can be retrieved from a menu callback to determine the SlingShot object that was clicked on to bring up the menu.

Argument:	Rectobj
Default:	n/a
Procs:	get

RECTOBJ_MIN_WIDTH
RECTOBJ_MIN_HEIGHT

The minimum dimensions that an object may be. Usually calculated by the object.

Argument:	int
Default:	n/a
Procs:	get

RECTOBJ_NORMAL
RECTOBJ_HIGHLIGHT
RECTOBJ_PREHIGHLIGHT
RECTOBJ_PREDROP_NORMAL
RECTOBJ_PREDROP_HIGHLIGHT

The paint style of the rectobj. These attributes take no arguments and are usually used internally to the SlingShot extensions.

RECTOBJ_NTH_CHILD

The nth child of the parent.

Argument:	int
Default:	n/a
Procs:	get, set

Usage:

```
child = (Rectobj) xv_get(parent, RECTOBJ_NTH_CHILD, 10);
```

RECTOBJ_N_CHILDREN

The number of children a rectobj has.

Argument:	int
Default:	0
Procs:	get

RECTOBJ_OPS

All the callback functions for a rectobj are grouped into one structure, Rectobj_ops. This is an alternate way to manipulate those callbacks. It is also more efficient because it reduces the amount of allocated memory.

Argument: Rectobj_ops*
Default: defined by class
Procs: create, get, set

RECTOBJ_PAINT_PROC

The callback function that paints an object. Generally only used for class implementations, it can be used by applications if a specific appearance is desired. The paint proc should not paint outside its XV_RECT and the xrects argument.

Argument: Proc_ptr
Default: depends on class
Procs: create, get, set
Usage:

```
void paint_proc(
        Rectobj      rectobj,
        Display      *display,
        Window       window,
        Xv_xrectlist *xrects);
```

RECTOBJ_SELECTABLE

If TRUE, the object can be selected with a single click.

Argument: Boolean
Default: True
Procs: create, get, set

RECTOBJ_SELECTED

If TRUE, the object is currently selected.

Argument: Boolean
Procs: get, set

RECTOBJ_SELECTION_OWNER

The XView Selection_owner object used to acquire a selection when an object is selected. This should be set by the application on the highest object in the rectobj hierarchy that has children that will hold a selection.

Argument: Selection_owner
Default: NULL
Procs: create, get, set

RECTOBJ_SELECTION_PROC

The callback function that is called when an object is selected or deselected. The second argument indicates the new state of the object (TRUE if selected). The event parameter is helpful if you need to determine the timestamp of the selection (be warned that sometimes this is NULL). You also need to be careful when the object is being deselected because the object may be being destroyed.

Argument: Proc_ptr
Default: NULL
Procs: create, get, set
Usage:

```
void   selection_proc(
              Rectobj       rectobj,
              int           selected,
              Event         *event);
```

RECTOBJ_SET_GEOMETRY_PROC

The callback function that is called when an object's geometry is changed. The function should propagate the changes to its children.

Argument: Proc_ptr
Default: depends on class
Procs: create, get, set
Usage:

```
void   set_geometry_proc(
              Rectobj       rectobj,
              Rect          *newrect,
              Rect          *oldrect);
```

RECTOBJ_SINGLE_CLICK_PROC

The callback function that is called when an object is clicked on. The callback is invoked when the mouse button goes up. When using rectobj_toggle_click_proc(), the state parameter indicates if the object has been toggled on (highlighted) or off (normal).

Argument: Proc_ptr
Default: NULL
Procs: create, get, set
Usage:

```
void   single_click_proc(
       Xv_window     paint_window,
       Event         *event,
       Canvas_shell  canvas_shell,
       Rectobj       rectobj,
       int           state);
```

RECTOBJ_STACKING_POSITION

The stacking position of an object, relative to siblings of the same parent. Higher numbers will bring the object to the top.

Argument: int
Default: N/A (newly created objects appear on top)
Procs: get, set

RECTOBJ_START_DRAG_PROC

The callback function that is called when the user drags the object more beyond the drag threshold (*SlingShot.ClickMoveThreshold*). This callback is only invoked by the rectobj_selection_event_proc event handling routine.

Argument: Proc_ptr
Default: NULL
Procs: create, get, set
Usage:

```
void   start_drag_proc(
       Xv_window    paint_window,
       Event        *event,
       Canvas_shell canvas_shell,
       Rectobj      rectobj,
       int          button_down_x,
       int          button_down_y,
       int          adjust);
```

RECTOBJ_STYLE_CHANGE_PROC

The notification routine for when an object goes between paint styles.

RECTOBJ_TOGGLE_STATE

When using rectobj_toggle_event_proc(), this returns the state of the object, TRUE if it is highlighted, FALSE otherwise.

Argument: Rectobj
Default: FALSE
Procs: create, set, get

RECTOBJ_X
RECTOBJ_Y

The X and Y position of an object's upper left corner, relative to the coordinates of the paint window. This is measured in pixels.

Argument: int
Default: n/a
Procs: create, get, set

TREE_ADD_LINK

Connects the second argument (a rectobj) as a subtree of the first argument (a rectobj). Both objects should be owned by the same tree. The tree itself acts as the root node.

Argument: Rectobj, Rectobj
Default: n/a
Procs: set
Usage:

```
xv_set(tree,
       TREE_ADD_LINK, object1, object2,
       NULL);
```

TREE_BORDER

Space in pixels that is reserved around each node of a tree.

Argument: int
Default: 4
Procs: create, get, set

TREE_DRAWLINE

Returns the handle to the drawline object that points to a node.

Argument: Rectobj
Default: n/a
Procs: get
Usage:

```
drawline = (Drawline) xv_get(tree, TREE_DRAWLINE, object);
```

TREE_LAYOUT

The orientation of the tree.

Argument: Tree_layout
`TREE_LAYOUT_HORIZONTAL, TREE_LAYOUT_VERTICAL`
Default: TREE_LAYOUT_HORIZONTAL
Procs: create, get, set

TREE_LINK_FROM

Returns the node (Rectobj) that points to the argument (Rectobj).

Argument: Rectobj
Default: n/a
Procs: get
Usage:

```
object1 = (Rectobj) xv_get(tree, TREE_LINK_FROM, object2);
```

TREE_LINK_TO_LIST

Returns the list of Rectobjs that a node points to.

Argument: Rectobj_list*
Default: n/a
Procs: get
Usage:

```
list = (Rectobj_list*) xv_get(tree, TREE_LINK_TO_LIST, object1);
```

TREE_PARENT_DISTANCE

The distance, in pixels, between nodes and their leaves.

Argument: int
Default: 30
Procs: create, get, set

TREE_UNLINK

Detaches a node from the tree hierarchy.

Argument: Rectobj
Default: n/a
Procs: set
Usage:

```
xv_set(tree,
        TREE_UNLINK, object,
        NULL);
```

XV_RECT

A pointer to a XView `Rect` structure.

Argument: Rect *
Default: n/a
Procs: create, get, set

XV_SHOW

If FALSE, the object is not visible. The object remains on the selected list when it is made invisible.

Argument: Boolean
Default: TRUE
Procs: create, get, set

XV_WIDTH
XV_HEIGHT

The width and height of the object, measured in pixels. Some subclasses calculate these values based on other attributes.

Argument: int
Default: 0
Procs: create, get, set

XV_X
XV_Y

The X and Y position of an object's upper left corner. This is measured in pixels and relative to the parent's origin. Some objects explicitly set these on their children.

Argument: int
Default: 0
Procs: create, get, set