Guile-GNOME: Libgnomecanvas

version 2.16.2, updated 9 December 2011

Federico Mena Quintero
Raph Levien
and others

Short Contents

1 Overview

`(gnome canvas)` wraps the Gnome-Canvas library for Guile. It is a part of Guile-GNOME.

Libgnomecanvas is a retained-mode canvas: you tell it what should be on it, and it handles redraw operations for you. It is a bit dated, in that it is backed by a library called "libart" and not Cairo, but no clear successor has emerged yet.

The source distribution has a fairly comprehensive example; see `examples/canvas.scm` for more information.

See the documentation for `(gnome gobject)` for more information on Guile-GNOME.

2 GnomeCanvasBpath

Bezier path canvas item

2.1 Overview

A canvas item type for creating a "path" from curve and line segments.

2.2 Usage

`<gnome-canvas-bpath>` [Class]

Derives from `<gnome-canvas-shape>`.

This class defines the following slots:

`bpath`

3 GnomeCanvasClipgroup

Canvas group that implements clipping

3.1 Overview

A canvas group object that clips the view of its children to a shape defined by a `<gnome-canvas-path-def>`.

3.2 Usage

`<gnome-canvas-clipgroup>` [Class]

Derives from `<gnome-canvas-group>`.

This class defines the following slots:

`path`

`wind`

4 GnomeCanvasEllipse

Canvas item for drawing ellipses and circles

4.1 Overview

A canvas item for drawing ellipses and circles. The parameters are defined in the parent classes including `<gnome-canvas-re>` which is shared with `<gnome-canvas-rect>` items as well.

4.2 Usage

`<gnome-canvas-ellipse>` [Class]

Derives from `<gnome-canvas-re>`.

This class defines no direct slots.

5 GnomeCanvasGroup

Canvas item group

5.1 Overview

A canvas item that groups other canvas items. A canvas group is used for organization, determining drawing stacking order, and for applying transforms on all items in the group.

The `<gnome-canvas>` widget contains a toplevel "root" group which can be queried with a call to `gnome-canvas-root`.

5.2 Usage

`<gnome-canvas-group>` [Class]

Derives from `<gnome-canvas-item>`.

This class defines the following slots:

`x` X

`y` Y

6 GnomeCanvasItem

Base class for all canvas items

6.1 Overview

This is the base class for all canvas items. Canvas items are the drawing elements of a `<gnome-canvas>`. Example items include lines, ellipses, polygons, images, text, curves and even arbitrary GTK+ widgets.

Canvas items use the GObject property system to query and set parameters. Properties are inherited so, for example, a `<gnome-canvas-polygon>` has a "fill-color" property that is inherited from its parent class object `<gnome-canvas-shape>`. So be sure to check the parent classes of `<gnome-canvas-item>` objects when looking for item properties. More information on GObject properties can be found in the glib library GObject API reference documentation.

To create a new canvas item call `gnome-canvas-item-new` which takes a parent `<gnome-canvas-group>`, GType of the item to create, and a NULL terminated list of name/value GObject properties to set for the new item.

To change an existing canvas item's properties call `gnome-canvas-item-set`, or `g-object-set` can also be used.

There are several functions to change the drawing stacking order of an item. Call `gnome-canvas-item-raise` to raise an item a specified number of positions or `gnome-canvas-item-lower` to lower it. To raise an item to the top call `gnome-canvas-item-raise-to-top`. The `gnome-canvas-item-lower-to-bottom` function will put it at the bottom.

To show an item call `gnome-canvas-item-show`. Note that canvas item's are shown by default and so do not need to be explicitly shown after creation (contrary to GTK+ widget behavior). Call `gnome-canvas-item-hide` to hide an item.

To move an item relative to its current position (item coordinates) call `gnome-canvas-item-move` or `gnome-canvas-item-affine-relative` for more complex transforms. `gnome-canvas-item-affine-absolute` can be used to set an item's transform to specific values (not offsets).

To convert between world and item coordinate systems call `gnome-canvas-item-w2i`, and to convert in the other direction call `gnome-canvas-item-i2w`. To get the transform for converting from item to world coordinates use `gnome-canvas-item-i2w-affine` or for converting item to canvas coordinates, `gnome-canvas-item-i2c-affine`.

Handling user input for interactive items is accomplished through a few functions and the "event" signal. To grab the mouse cursor call `gnome-canvas-item-grab`, it can be ungrabbed with `gnome-canvas-item-ungrab` (see `gdk-pointer-grab` of the GTK+ library for details). To grab keyboard focus call `gnome-canvas-item-grab-focus`. Received events will be signaled via the "event" signal.

Some other useful functions include a reparenting routine, `gnome-canvas-item-reparent`, and a function to query the bounding box of an item (a minumum rectangular area containing all parts of the item), `gnome-canvas-item-get-bounds`.

6.2 Usage

`<gnome-canvas-item>` [Class]

Derives from `<gtk-object>`.

This class defines the following properties:

`parent`

`event` (*arg0* `<gdk-event>`) ⇒ `<gboolean>` [Signal on `<gnome-canvas-item>`]

Signals mouse button clicks, motion, enter/leave, and key press events on canvas items. Use this signal to create user interactive items. The "x") and converted to canvas world coordinates.

`gnome-canvas-item-move` (*self* `<gnome-canvas-item>`) (*dx* `double`) (*dy* `double`) [Function]

`move` [Method]

Moves a canvas item by creating an affine transformation matrix for translation by using the specified values. This happens in item local coordinate system, so if you have nontrivial transform, it most probably does not do, what you want.

item A canvas item.

dx Horizontal offset.

dy Vertical offset.

`gnome-canvas-item-affine-relative` (*self* `<gnome-canvas-item>`) (*x1* `double`) (*y2* `double`) (*x2* `double`) (*y2* `double`) (*x3* `double`) (*y3* `double`) [Function]

`affine-relative` [Method]

Combines the specified affine transformation matrix with the item's current transformation. NULL affine is not allowed.

item A canvas item.

affine An affine transformation matrix.

`gnome-canvas-item-affine-absolute` (*self* `<gnome-canvas-item>`) (*x1* `double`) (*y2* `double`) (*x2* `double`) (*y2* `double`) (*x3* `double`) (*y3* `double`) [Function]

`affine-absolute` [Method]

Makes the item's affine transformation matrix be equal to the specified matrix. NULL affine is treated as identity.

item A canvas item.

affine An affine transformation matrix.

`gnome-canvas-item-raise` (*self* `<gnome-canvas-item>`) (*positions* `int`) [Function]

`raise` [Method]

Raises the item in its parent's stack by the specified number of positions. If the number of positions is greater than the distance to the top of the stack, then the item is put at the top.

item A canvas item.

positions Number of steps to raise the item.

gnome-canvas-item-lower (*self* <gnome-canvas-item>) (*positions* int) [Function]

lower [Method]

Lowers the item in its parent's stack by the specified number of positions. If the number of positions is greater than the distance to the bottom of the stack, then the item is put at the bottom.

item A canvas item.

positions Number of steps to lower the item.

gnome-canvas-item-raise-to-top (*self* <gnome-canvas-item>) [Function]

raise-to-top [Method]

Raises an item to the top of its parent's stack.

item A canvas item.

gnome-canvas-item-lower-to-bottom (*self* <gnome-canvas-item>) [Function]

lower-to-bottom [Method]

Lowers an item to the bottom of its parent's stack.

item A canvas item.

gnome-canvas-item-show (*self* <gnome-canvas-item>) [Function]

show [Method]

Shows a canvas item. If the item was already shown, then no action is taken.

item A canvas item.

gnome-canvas-item-hide (*self* <gnome-canvas-item>) [Function]

hide [Method]

Hides a canvas item. If the item was already hidden, then no action is taken.

item A canvas item.

gnome-canvas-item-grab (*self* <gnome-canvas-item>) (*event_mask* unsigned-int) (*cursor* <gdk-cursor>) (*etime* unsigned-int32) ⇒ (*ret* int) [Function]

grab [Method]

Specifies that all events that match the specified event mask should be sent to the specified item, and also grabs the mouse by calling `gdk-pointer-grab`. The event mask is also used when grabbing the pointer. If *cursor* is not NULL, then that cursor is used while the grab is active. The *etime* parameter is the timestamp required for grabbing the mouse.

Return value: If an item was already grabbed, it returns '`GDK_GRAB_ALREADY_GRABBED`'. If the specified item was hidden by calling `gnome-canvas-item-hide`, then it

item A canvas item.

event-mask
Mask of events that will be sent to this item.

cursor If non-NULL, the cursor that will be used while the grab is active.

etime The timestamp required for grabbing the mouse, or GDK_CURRENT_TIME.

ret 'GDK_GRAB_NOT_VIEWABLE'. Else, it returns the result of calling `gdk-pointer-grab`.

`gnome-canvas-item-ungrab` (*self* `<gnome-canvas-item>`) (*etime* `unsigned-int32`) [Function]
`ungrab` [Method]

Ungrabs the item, which must have been grabbed in the canvas, and ungrabs the mouse.

item A canvas item that holds a grab.

etime The timestamp for ungrabbing the mouse.

`gnome-canvas-item-w2i` (*self* `<gnome-canvas-item>`) ⇒ (*x* `double`) (*y* `double`) [Function]
`w2i` [Method]

Converts a coordinate pair from world coordinates to item-relative coordinates.

item A canvas item.

x X coordinate to convert (input/output value).

y Y coordinate to convert (input/output value).

`gnome-canvas-item-i2w` (*self* `<gnome-canvas-item>`) ⇒ (*x* `double`) (*y* `double`) [Function]
`i2w` [Method]

Converts a coordinate pair from item-relative coordinates to world coordinates.

item A canvas item.

x X coordinate to convert (input/output value).

y Y coordinate to convert (input/output value).

`gnome-canvas-item-reparent` (*self* `<gnome-canvas-item>`) (*new_group* `<gnome-canvas-group>`) [Function]
`reparent` [Method]

Changes the parent of the specified item to be the new group. The item keeps its group-relative coordinates as for its old parent, so the item may change its absolute position within the canvas.

item A canvas item.

new-group A canvas group.

`gnome-canvas-item-grab-focus` (*self* `<gnome-canvas-item>`) [Function]
`grab-focus` [Method]

Makes the specified item take the keyboard focus, so all keyboard events will be sent to it. If the canvas widget itself did not have the focus, it grabs it as well.

item A canvas item.

`gnome-canvas-item-get-bounds` (*self* `<gnome-canvas-item>`) [Function]
⇒ (*x1* `double`) (*y1* `double`) (*x2* `double`) (*y2* `double`)

`get-bounds` [Method]

Queries the bounding box of a canvas item. The bounds are returned in the coordinate system of the item's parent.

item A canvas item.

x1 Leftmost edge of the bounding box (return value).

y1 Upper edge of the bounding box (return value).

x2 Rightmost edge of the bounding box (return value).

y2 Lower edge of the bounding box (return value).

`gnome-canvas-item-request-update` (*self* `<gnome-canvas-item>`) [Function]

`request-update` [Method]

To be used only by item implementations. Requests that the canvas queue an update for the specified item.

item A canvas item.

7 GnomeCanvasLine

Line canvas item

7.1 Overview

A canvas item for drawing lines. This canvas item uses a `<gnome-canvas-points>` structure so one or multiple joined lined segments can be drawn with a single `<gnome-canvas-line>` item.

7.2 Usage

`<gnome-canvas-line>` [Class]

Derives from `<gnome-canvas-item>`.

This class defines the following properties:

`points`

`fill-color`
`fill-color-gdk`
`fill-color-rgba`
`fill-stipple`
`width-pixels`
`width-units`
`cap-style`
`join-style`
`line-style`
`first-arrowhead`
`last-arrowhead`
`smooth`

`spline-steps`
`arrow-shape-a`
`arrow-shape-b`
`arrow-shape-c`

8 gnome-canvas-path-def

Container and functions for manipulating ArtBpaths

8.1 Overview

Convenient container and functions for manipulating ArtBpaths, which are paths defined by line and curve segments.

8.2 Usage

`<gnome-canvas-path-def>` [Class]

Derives from `<gboxed>`.

`gnome-canvas-path-def-new` ⇒ (*ret* `<gnome-canvas-path-def>`) [Function]

This function creates a new empty `<gnome-canvas-path-def>`.

ret the new canvas path definition.

`gnome-canvas-path-def-new-sized` (*length* `int`) ⇒ (*ret* `<gnome-canvas-path-def>`) [Function]

This function creates a new `<gnome-canvas-path-def>` with *length* number of points allocated. It is useful, if you know the exact number of points in path, so you can avoid automatic point array reallocation.

length number of points to allocate for the path

ret the new canvas path definition

`gnome-canvas-path-def-finish` (*self* `<gnome-canvas-path-def>`) [Function]

Trims dynamic point array to exact length of path.

path a GnomeCanvasPathDef

`gnome-canvas-path-def-ensure-space` (*self* `<gnome-canvas-path-def>`) (*space* `int`) [Function]

This function ensures that enough space for *space* points is allocated at the end of the path.

path a GnomeCanvasPathDef

space number of points to guarantee are allocated at the end of the path.

`gnome-canvas-path-def-duplicate` (*self* `<gnome-canvas-path-def>`) ⇒ (*ret* `<gnome-canvas-path-def>`) [Function]

This function duplicates the passed *path*. The new path is marked as non-static regardless of the state of original.

path a GnomeCanvasPathDef to duplicate

ret a GnomeCanvasPathDef which is a duplicate of *path*.

gnome-canvas-path-def-concat (*list* gslist-of) ⇒ (*ret* <gnome-canvas-path-def>) [Function]

This function concatenates a list of GnomeCanvasPathDefs into one newly created GnomeCanvasPathDef.

list a GSList of GnomeCanvasPathDefs to concatenate into one new path.

ret a new GnomeCanvasPathDef

gnome-canvas-path-def-split (*self* <gnome-canvas-path-def>) ⇒ (*ret* gslist-of) [Function]

This function splits the passed *path* into a list of GnomeCanvasPathDefs which represent each segment of the origional path. The path is split when ever an ART_MOVETO or ART_MOVETO_OPEN is encountered. The closedness of resulting paths is set accordingly to closedness of corresponding segment.

path a GnomeCanvasPathDef

ret a list of GnomeCanvasPathDef(s).

gnome-canvas-path-def-open-parts (*self* <gnome-canvas-path-def>) ⇒ (*ret* <gnome-canvas-path-def>) [Function]

This function creates a new GnomeCanvasPathDef that contains all of the open segments on the passed *path*.

path a GnomeCanvasPathDef

ret a new GnomeCanvasPathDef that contains all of the open segemtns in *path*.

gnome-canvas-path-def-closed-parts (*self* <gnome-canvas-path-def>) ⇒ (*ret* <gnome-canvas-path-def>) [Function]

This function returns a new GnomeCanvasPathDef that contains the all of close parts of passed *path*.

path a GnomeCanvasPathDef

ret a new GnomeCanvasPathDef that contains all of the closed parts of passed *path*.

gnome-canvas-path-def-close-all (*self* <gnome-canvas-path-def>) ⇒ (*ret* <gnome-canvas-path-def>) [Function]

This function closes all of the open segments in the passed path and returns a new GnomeCanvasPathDef.

path a GnomeCanvasPathDef

ret a GnomeCanvasPathDef that contains the contents of *path* but has modified the path is fully closed

gnome-canvas-path-def-reset (*self* <gnome-canvas-path-def>) [Function]

This function clears the contents of the passed *path*.

path a GnomeCanvasPathDef

`gnome-canvas-path-def-moveto` (*self* `<gnome-canvas-path-def>`) (*x* `double`) (*y* `double`) [Function]

This function adds starts new subpath on *path*, and sets its starting point to *x* and *y*. If current subpath is empty, it simply changes its starting coordinates to new values.

path a GnomeCanvasPathDef

x x coordinate

y y coordinate

`gnome-canvas-path-def-lineto` (*self* `<gnome-canvas-path-def>`) (*x* `double`) (*y* `double`) [Function]

This function add a line segment to the passed *path* with the specified *x* and *y* coordinates.

path a GnomeCanvasPathDef

x x coordinate

y y coordinate

`gnome-canvas-path-def-lineto-moving` (*self* `<gnome-canvas-path-def>`) (*x* `double`) (*y* `double`) [Function]

This functions adds a new line segment with loose endpoint to the path, or if endpoint is already loose, changes its coordinates to *x*, *y*. You can change the coordinates of loose endpoint as many times as you want, the last ones set will be fixed, if you continue line. This is useful for handling drawing with mouse.

path a GnomeCanvasPathDef

x x coordinate

y y coordinate

`gnome-canvas-path-def-curveto` (*self* `<gnome-canvas-path-def>`) (*x0* `double`) (*y0* `double`) (*x1* `double`) (*y1* `double`) (*x2* `double`) (*y2* `double`) [Function]

This function adds a bezier curve segment to the path definition.

path a GnomeCanvasPathDef

x0 first control point x coordinate

y0 first control point y coordinate

x1 second control point x coordinate

y1 second control point y coordinate

x2 end of curve x coordinate

y2 end of curve y coordinate

`gnome-canvas-path-def-closepath` (*self* `<gnome-canvas-path-def>`) [Function]

This function closes the last subpath of *path*, adding a ART_LINETO to subpath starting point, if needed and changing starting pathcode to ART_MOVETO

path a GnomeCanvasPathDef

`gnome-canvas-path-def-length` (*self* `<gnome-canvas-path-def>`) ⇒ (*ret* `int`) [Function]

This function returns the length of the path definition. Not Euclidian length of the path but rather the number of points on the path.

path a GnomeCanvasPathDef

ret integer, number of points on the path.

`gnome-canvas-path-def-is-empty` (*self* `<gnome-canvas-path-def>`) ⇒ (*ret* `bool`) [Function]

This function is a boolean test to see if the path is empty, meaning containing no line segments.

path a GnomeCanvasPathDef

ret boolean, indicating if the path is empty.

`gnome-canvas-path-def-any-open` (*self* `<gnome-canvas-path-def>`) ⇒ (*ret* `bool`) [Function]

This function returns a boolean value indicating if the path has any open segments.

path a GnomeCanvasPathDef

ret boolean, indicating if the path has any open segments.

`gnome-canvas-path-def-all-open` (*self* `<gnome-canvas-path-def>`) ⇒ (*ret* `bool`) [Function]

This function returns a boolean value indicating if the path only contains open segments.

path a GnomeCanvasPathDef

ret boolean, indicating if the path has all open segments.

`gnome-canvas-path-def-any-closed` (*self* `<gnome-canvas-path-def>`) ⇒ (*ret* `bool`) [Function]

This function returns a boolean valid indicating if the path has any closed segements.

path a GnomeCanvasPathDef

ret boolean, indicating if the path has any closed segments.

`gnome-canvas-path-def-all-closed` (*self* `<gnome-canvas-path-def>`) ⇒ (*ret* `bool`) [Function]

This function returns a boolean value indicating if the path only contains closed segments.

path a GnomeCanvasPathDef

ret boolean, indicating if the path has all closed segments.

9 GnomeCanvasPixbuf

Pixbuf image canvas item

9.1 Overview

A canvas item for drawing pixbuf images.

9.2 Usage

`<gnome-canvas-pixbuf>` [Class]

Derives from `<gnome-canvas-item>`.

This class defines the following properties:

`pixbuf`

`width`

`width-set`
`width-in-pixels`
`height`

`height-set`
`height-in-pixels`
`x`

`x-in-pixels`
`y`

`y-in-pixels`
`anchor`

10 GnomeCanvasPolygon

Polygon canvas item

10.1 Overview

A canvas item for drawing polygon (multi sided) shapes.

10.2 Usage

`<gnome-canvas-polygon>` [Class]

Derives from `<gnome-canvas-shape>`.

This class defines the following properties:

`points`

11 GnomeCanvasRect

Rectangle canvas item

11.1 Overview

A canvas item for drawing rectangles and squares. The parameters are defined in the parent classes including `<gnome-canvas-re>` which is shared with `<gnome-canvas-ellipse>` items as well.

11.2 Usage

`<gnome-canvas-rect>` [Class]

Derives from `<gnome-canvas-re>`.

This class defines no properties, other than those defined by its superclasses.

12 GnomeCanvasRE

Base class for rectangle and ellipse items

12.1 Overview

This forms a base class for rectangle and ellipse canvas items.

12.2 Usage

`<gnome-canvas-re>` [Class]

Derives from `<gnome-canvas-shape>`.

This class defines the following properties:

`x1`

`y1`

`x2`

`y2`

13 GnomeCanvasRichText

Rich text canvas item

13.1 Overview

A canvas item that displays a GtkTextBuffer which is a flexible text display and editing widget. Consult GtkTextBuffer info in the GTK+ Reference documentation for more details.

13.2 Usage

`<gnome-canvas-rich-text>` [Class]

Derives from `<gnome-canvas-item>`.

This class defines the following properties:

`text` Text to display

`x` X position

`y` Y position

`width` Width for text box

`height` Height for text box

`editable` Is this rich text item editable?

`visible` Is this rich text item visible?

`cursor-visible`
Is the cursor visible in this rich text item?

`cursor-blink`
Does the cursor blink in this rich text item?

`grow-height`
Should the text box height grow if the text does not fit?

`wrap-mode`
Wrap mode for multiline text

`justification`
Justification mode

`direction`
Text direction

`anchor` Anchor point for text

`pixels-above-lines`
Number of pixels to put above lines

`pixels-below-lines`
Number of pixels to put below lines

`pixels-inside-wrap`
Number of pixels to put inside the wrap

`left-margin`
: Number of pixels in the left margin

`right-margin`
: Number of pixels in the right margin

`indent`
: Number of pixels for indentation

tag-changed (*arg0* `<gobject>`) [Signal on `<gnome-canvas-rich-text>`]

gnome-canvas-rich-text-set-buffer [Function]
(*self* `<gnome-canvas-rich-text>`) (*buffer* `<gtk-text-buffer>`)
set-buffer [Method]

Sets the buffer field of the *text* to *buffer*.

text a `<gnome-canvas-rich-text>`.

buffer a `<gtk-text-buffer>`.

gnome-canvas-rich-text-get-buffer [Function]
(*self* `<gnome-canvas-rich-text>`) ⇒ (*ret* `<gtk-text-buffer>`)
get-buffer [Method]

Returns a `<gtk-text-buffer>` associated with the `<gnome-canvas-rich-text>`. This function creates a new `<gtk-text-buffer>` if the text buffer is NULL.

text a `<gnome-canvas-rich-text>`.

ret the `<gtk-text-buffer>`.

14 GnomeCanvasShape

Base class for canvas item shapes

14.1 Overview

Provides a base class for canvas item shapes, including: `<gnome-canvas-bpath>`, `<gnome-canvas-ellipse>`, `<gnome-canvas-rect>`, and `<gnome-canvas-polygon>`.

14.2 Usage

`<gnome-canvas-shape>` [Class]

Derives from `<gnome-canvas-item>`.

This class defines the following properties:

`fill-color`
`fill-color-gdk`
`fill-color-rgba`
`outline-color`
`outline-color-gdk`
`outline-color-rgba`
`fill-stipple`
`outline-stipple`
`width-pixels`
`width-units`
`cap-style`
`join-style`
`wind`

`miterlimit`
`dash`

`gnome-canvas-shape-set-path-def` (*self* `<gnome-canvas-shape>`) (*def* `<gnome-canvas-path-def>`) [Function]

`set-path-def` [Method]

This function sets the the GnomeCanvasPathDef used by the GnomeCanvasShape. Notice, that it does not request updates, as it is meant to be used from item implementations, from inside update queue.

shape a GnomeCanvasShape

def a GnomeCanvasPathDef

`gnome-canvas-shape-get-path-def` (*self* `<gnome-canvas-shape>`) ⇒ (*ret* `<gnome-canvas-path-def>`) [Function]

`get-path-def` [Method]

This function returns the `<gnome-canvas-path-def>` that the shape currently uses. It adds a reference to the `<gnome-canvas-path-def>` and returns it, if there is not a `<gnome-canvas-path-def>` set for the shape it returns NULL.

shape a GnomeCanvasShape

ret a `<gnome-canvas-path-def>` or NULL if none is set for the shape.

15 GnomeCanvasText

Text canvas item

15.1 Overview

A canvas item for displaying text. See `<gnome-canvas-rich-text>` for a more advanced text display and editing canvas item. NB: The `<gnome-canvas-text>` item doesn't scale with the zoom property of the `<gnome-canvas>`. A zoomable implementation could derive from `<gnome-canvas-text>` and check the zoom property for manual adjustments to the font size.

15.2 Usage

`<gnome-canvas-text>` [Class]

Derives from `<gnome-canvas-item>`.

This class defines the following properties:

`text` Text to render

`markup` Marked up text to render

`x`

`y`

`font` Font description as a string

`font-desc`
Font description as a PangoFontDescription struct

`family` Name of the font family, e.g. Sans, Helvetica, Times, Monospace

`family-set`
Whether this tag affects the font family

`attributes`

`style` Font style

`style-set`
Whether this tag affects the font style

`variant` Font variant

`variant-set`
Whether this tag affects the font variant

`weight` Font weight

`weight-set`
Whether this tag affects the font weight

`stretch` Font stretch

`stretch-set`
Whether this tag affects the font stretch

`size` Font size (as a multiple of PANGO_SCALE, eg. 12*PANGO_SCALE for a 12pt font size)

`size-set` Whether this tag affects the font size

`size-points`
Font size in points (eg. 12 for a 12pt font size)

`strikethrough`
Whether to strike through the text

`strikethrough-set`
Whether this tag affects strikethrough

`underline`
Style of underline for this text

`underline-set`
Whether this tag affects underlining

`rise` Offset of text above the baseline (below the baseline if rise is negative)

`rise-set` Whether this tag affects the rise

`scale` Size of font, relative to default size

`scale-set`
Whether this tag affects font scaling

`anchor`

`justification`
`clip-width`
`clip-height`
`clip`

`x-offset`

`y-offset`

`fill-color`
Text color, as string

`fill-color-gdk`
Text color, as a GdkColor

`fill-color-rgba`
Text color, as an R/G/B/A combined integer

`fill-stipple`
`text-width`
Width of the rendered text

`text-height`
Height of the rendered text

16 gnome-canvas-util

Canvas utility functions

16.1 Overview

Some useful canvas utility functions.

The `<gnome-canvas-points>` structure manages an array of points (X and Y coordinates) and is used by `<gnome-canvas-line>` and `<gnome-canvas-polygon>` canvas items.

To create a `<gnome-canvas-points>` structure call `gnome-canvas-points-new` and when finished using it call `gnome-canvas-points-free`.

Of note is that the `<gnome-canvas-points>` structure is actually managed by a reference count, so it won't be freed until this count reaches 0. To increment its reference count call `gnome-canvas-points-ref` and to decrement it call `gnome-canvas-points-unref`.

16.2 Usage

`<gnome-canvas-points>` [Class]

Derives from `<gboxed>`.

`gnome-canvas-get-miter-points` (*x1* `double`) (*y1* `double`) (*x2* `double`) (*y2* `double`) (*x3* `double`) (*y3* `double`) (*width* `double`) ⇒ (*ret* `int`) (*mx1* `double`) (*my1* `double`) (*mx2* `double`) (*my2* `double`) [Function]

Given three points forming an angle, computes the coordinates of the inside and outside points of the mitered corner formed by a line of a given width at that angle.

x1 X coordinate of the first point

y1 Y coordinate of the first point

x2 X coordinate of the second (angle) point

y2 Y coordinate of the second (angle) point

x3 X coordinate of the third point

y3 Y coordinate of the third point

width Width of the line

mx1 The X coordinate of the first miter point is returned here.

my1 The Y coordinate of the first miter point is returned here.

mx2 The X coordinate of the second miter point is returned here.

my2 The Y coordinate of the second miter point is returned here.

ret FALSE if the angle is less than 11 degrees (this is the same threshold as X uses. If this occurs, the return points are not modified. Otherwise, returns TRUE.

gnome-canvas-get-butt-points (*x1* double) (*y1* double) (*x2* double) (*y2* double) (*width* double) (*project* int) ⇒ (*bx1* double) (*by1* double) (*bx2* double) (*by2* double) [Function]

Computes the butt points of a line segment.

x1 X coordinate of first point in the line

y1 Y cooordinate of first point in the line

x2 X coordinate of second point (endpoint) of the line

y2 Y coordinate of second point (endpoint) of the line

width Width of the line

project Whether the butt points should project out by width/2 distance

bx1 X coordinate of first butt point is returned here

by1 Y coordinate of first butt point is returned here

bx2 X coordinate of second butt point is returned here

by2 Y coordinate of second butt point is returned here

gnome-canvas-polygon-to-point (*num_points* int) (*x* double) (*y* double) ⇒ (*ret* double) (*poly* double) [Function]

Computes the distance between a point and a polygon.

poly Vertices of the polygon. X coordinates are in the even indices, and Y coordinates are in the odd indices

num-points Number of points in the polygon

x X coordinate of the point

y Y coordinate of the point

ret The distance from the point to the polygon, or zero if the point is inside the polygon.

gnome-canvas-item-reset-bounds (*self* <gnome-canvas-item>) [Function]
reset-bounds [Method]

Resets the bounding box of a canvas item to an empty rectangle.

item A canvas item

gnome-canvas-update-bbox (*item* <gnome-canvas-item>) (*x1* int) (*y1* int) (*x2* int) (*y2* int) [Function]

Sets the bbox to the new value, requesting full repaint.

item the canvas item needing update

x1 Left coordinate of the new bounding box

y1 Top coordinate of the new bounding box

x2 Right coordinate of the new bounding box

y2 Bottom coordinate of the new bounding box

17 GnomeCanvasWidget

Widget canvas item

17.1 Overview

A canvas item for placing arbitrary GtkWidget objects onto a canvas.

17.2 Usage

`<gnome-canvas-widget>` [Class]

Derives from `<gnome-canvas-item>`.

This class defines the following properties:

`widget`

`x`

`y`

`width`

`height`

`anchor`

`size-pixels`

18 GnomeCanvas

Main canvas widget

18.1 Overview

The `<gnome-canvas>` is an engine for structured graphics that offers a rich imaging model, high performance rendering, and a powerful, high level API. It offers a choice of two rendering back-ends, one based on Xlib for extremely fast display, and another based on Libart, a sophisticated, antialiased, alpha-compositing engine. This widget can be used for flexible display of graphics and for creating interactive user interface elements.

To create a new `<gnome-canvas>` widget call `gnome-canvas-new` or `gnome-canvas-new-aa` for an anti-aliased mode canvas.

A `<gnome-canvas>` widget contains one or more `<gnome-canvas-item>` objects. Items consist of graphing elements like lines, ellipses, polygons, images, text, and curves. These items are organized using `<gnome-canvas-group>` objects, which are themselves derived from `<gnome-canvas-item>`. Since a group is an item it can be contained within other groups, forming a tree of canvas items. Certain operations, like translating and scaling, can be performed on all items in a group.

There is a special root group created by a `<gnome-canvas>`. This is the top level group under which all items in a canvas are contained. To get the root group from a canvas call `gnome-canvas-root`. To clear a canvas you can simply walk through the item_list member of the `<gnome-canvas-group>` and call `gtk-object-destroy` on each one.

There are several different coordinate systems used by `<gnome-canvas>` widgets. The primary system is a logical, abstract coordinate space called world coordinates. World coordinates are expressed as unbounded double floating point numbers. When it comes to rendering to a screen the canvas pixel coordinate system (also referred to as just canvas coordinates) is used. This system uses integers to specify screen pixel positions. A user defined scaling factor and offset are used to convert between world coordinates and canvas coordinates. Each item in a canvas has its own coordinate system called item coordinates. This system is specified in world coordinates but they are relative to an item (0.0, 0.0 would be the top left corner of the item). The final coordinate system of interest is window coordinates. These are like canvas coordinates but are offsets from within a window a canvas is displayed in. This last system is rarely used, but is useful when manually handling GDK events (such as drag and drop) which are specified in window coordinates (the events processed by the canvas are already converted for you).

Along with different coordinate systems comes functions to convert between them. `gnome-canvas-w2c` converts world to canvas pixel coordinates and `gnome-canvas-c2w` converts from canvas to world. `gnome-canvas-w2c-d` is like `gnome-canvas-w2c` but returns the pixel coordinates as doubles which is useful to avoid precision loss from integer rounding. To get the affine transform matrix for converting from world coordinates to canvas coordinates call `gnome-canvas-w2c-affine`. `gnome-canvas-window-to-world` converts from window to world coordinates and `gnome-canvas-world-to-window` converts in the other direction. There are no functions for converting between canvas and window coordinates, since this is just a matter of subtracting the canvas scrolling offset. To convert to/from item coordinates use the functions defined for `<gnome-canvas-item>` objects.

To set the canvas zoom factor (canvas pixels per world unit, the scaling factor) call `gnome-canvas-set-pixels-per-unit`, setting this to 1.0 will cause the two coordinate systems to correspond (e.g., [5, 6] in pixel units would be [5.0, 6.0] in world units).

Defining the scrollable area of a canvas widget is done by calling `gnome-canvas-set-scroll-region` and to get the current region `gnome-canvas-get-scroll-region` can be used. If the window is larger than the canvas scrolling region it can optionally be centered in the window. Use `gnome-canvas-set-center-scroll-region` to enable or disable this behavior. To scroll to a particular canvas pixel coordinate use `gnome-canvas-scroll-to` (typically not used since scrollbars are usually set up to handle the scrolling), and to get the current canvas pixel scroll offset call `gnome-canvas-get-scroll-offsets`.

18.2 Usage

`<gnome-canvas>` [Class]

Derives from `<gtk-layout>`.

This class defines the following properties:

`aa` The antialiasing mode of the canvas.

`focused-item`

`draw-background` (*arg0* `<gdk-drawable>`) (*arg1* `<gint>`) (*arg2* `<gint>`) (*arg3* `<gint>`) (*arg4* `<gint>`) [Signal on `<gnome-canvas>`]

This signal is emitted to draw the background for non-antialiased mode canvas widgets. The default method uses the canvas widget's style to draw the background.

`render-background` (*arg0* `<gpointer>`) [Signal on `<gnome-canvas>`]

This signal is emitted for antialiased mode canvas widgets to render the background. The buf data structure contains both a pointer to a packed 24-bit RGB array and the coordinates.

`gnome-canvas-new` ⇒ (*ret* `<gtk-widget>`) [Function]

Creates a new empty canvas in non-antialiased mode.

ret A newly-created canvas.

`gnome-canvas-new-aa` ⇒ (*ret* `<gtk-widget>`) [Function]

Creates a new empty canvas in antialiased mode.

ret A newly-created antialiased canvas.

`gnome-canvas-root` (*self* `<gnome-canvas>`) ⇒ (*ret* `<gnome-canvas-group>`) [Function]

`root` [Method]

Queries the root group of a canvas.

canvas A canvas.

ret The root group of the specified canvas.

`gnome-canvas-set-scroll-region` (*self* `<gnome-canvas>`) (*x1* `double`) (*y1* `double`) (*x2* `double`) (*y2* `double`) [Function]
`set-scroll-region` [Method]

Sets the scrolling region of a canvas to the specified rectangle. The canvas will then be able to scroll only within this region. The view of the canvas is adjusted as appropriate to display as much of the new region as possible.

canvas A canvas.

x1 Leftmost limit of the scrolling region.

y1 Upper limit of the scrolling region.

x2 Rightmost limit of the scrolling region.

y2 Lower limit of the scrolling region.

`gnome-canvas-get-scroll-region` (*self* `<gnome-canvas>`) ⇒ (*x1* `double`) (*y1* `double`) (*x2* `double`) (*y2* `double`) [Function]
`get-scroll-region` [Method]

Queries the scrolling region of a canvas.

canvas A canvas.

x1 Leftmost limit of the scrolling region (return value).

y1 Upper limit of the scrolling region (return value).

x2 Rightmost limit of the scrolling region (return value).

y2 Lower limit of the scrolling region (return value).

`gnome-canvas-set-pixels-per-unit` (*self* `<gnome-canvas>`) (*n* `double`) [Function]
`set-pixels-per-unit` [Method]

Sets the zooming factor of a canvas by specifying the number of pixels that correspond to one canvas unit.

The anchor point for zooming, i.e. the point that stays fixed and all others zoom inwards or outwards from it, depends on whether the canvas is set to center the scrolling region or not. You can control this using the `gnome-canvas-set-center-scroll-region` function. If the canvas is set to center the scroll region, then the center of the canvas window is used as the anchor point for zooming. Otherwise, the upper-left corner of the canvas window is used as the anchor point.

canvas A canvas.

n The number of pixels that correspond to one canvas unit.

`gnome-canvas-scroll-to` (*self* `<gnome-canvas>`) (*cx* `int`) (*cy* `int`) [Function]
`scroll-to` [Method]

Makes a canvas scroll to the specified offsets, given in canvas pixel units. The canvas will adjust the view so that it is not outside the scrolling region. This function is typically not used, as it is better to hook scrollbars to the canvas layout's scrolling adjusments.

canvas A canvas.

cx Horizontal scrolling offset in canvas pixel units.

cy Vertical scrolling offset in canvas pixel units.

`gnome-canvas-get-scroll-offsets` (*self* `<gnome-canvas>`) [Function]
⇒ (*cx* `int`) (*cy* `int`)
`get-scroll-offsets` [Method]

Queries the scrolling offsets of a canvas. The values are returned in canvas pixel units.

canvas A canvas.

cx Horizontal scrolling offset (return value).

cy Vertical scrolling offset (return value).

`gnome-canvas-update-now` (*self* `<gnome-canvas>`) [Function]
`update-now` [Method]

Forces an immediate update and redraw of a canvas. If the canvas does not have any pending update or redraw requests, then no action is taken. This is typically only used by applications that need explicit control of when the display is updated, like games. It is not needed by normal applications.

canvas A canvas.

`gnome-canvas-get-item-at` (*self* `<gnome-canvas>`) (*x* `double`) [Function]
(*y* `double`) ⇒ (*ret* `<gnome-canvas-item>`)
`get-item-at` [Method]

Looks for the item that is under the specified position, which must be specified in world coordinates.

canvas A canvas.

x X position in world coordinates.

y Y position in world coordinates.

ret The sought item, or NULL if no item is at the specified coordinates.

`gnome-canvas-request-redraw` (*self* `<gnome-canvas>`) (*x1* `int`) [Function]
(*y1* `int`) (*x2* `int`) (*y2* `int`)
`request-redraw` [Method]

Convenience function that informs a canvas that the specified rectangle needs to be repainted. This function converts the rectangle to a microtile array and feeds it to `gnome-canvas-request-redraw-uta`. The rectangle includes *x1* and *y1*, but not *x2* and *y2*. To be used only by item implementations.

canvas A canvas.

x1 Leftmost coordinate of the rectangle to be redrawn.

y1 Upper coordinate of the rectangle to be redrawn.

x2 Rightmost coordinate of the rectangle to be redrawn, plus 1.

y2 Lower coordinate of the rectangle to be redrawn, plus 1.

gnome-canvas-w2c (*self* <gnome-canvas>) (*wx* double) (*wy* double) ⇒ (*cx* int) (*cy* int) [Function]

w2c [Method]

Converts world coordinates into canvas pixel coordinates.

canvas A canvas.

wx World X coordinate.

wy World Y coordinate.

cx X pixel coordinate (return value).

cy Y pixel coordinate (return value).

gnome-canvas-w2c-d (*self* <gnome-canvas>) (*wx* double) (*wy* double) ⇒ (*cx* double) (*cy* double) [Function]

w2c-d [Method]

Converts world coordinates into canvas pixel coordinates. This version returns coordinates in floating point coordinates, for greater precision.

canvas A canvas.

wx World X coordinate.

wy World Y coordinate.

cx X pixel coordinate (return value).

cy Y pixel coordinate (return value).

gnome-canvas-c2w (*self* <gnome-canvas>) (*cx* int) (*cy* int) ⇒ (*wx* double) (*wy* double) [Function]

c2w [Method]

Converts canvas pixel coordinates to world coordinates.

canvas A canvas.

cx Canvas pixel X coordinate.

cy Canvas pixel Y coordinate.

wx X world coordinate (return value).

wy Y world coordinate (return value).

gnome-canvas-window-to-world (*self* <gnome-canvas>) (*winx* double) (*winy* double) ⇒ (*worldx* double) (*worldy* double) [Function]

window-to-world [Method]

Converts window-relative coordinates into world coordinates. You can use this when you need to convert mouse coordinates into world coordinates, for example.

canvas A canvas.

winx Window-relative X coordinate.

winy Window-relative Y coordinate.

worldx X world coordinate (return value).

worldy Y world coordinate (return value).

gnome-canvas-world-to-window (*self* <gnome-canvas>) (*worldx* double) (*worldy* double) ⇒ (*winx* double) (*winy* double) [Function]

world-to-window [Method]

Converts world coordinates into window-relative coordinates.

canvas A canvas.

worldx World X coordinate.

worldy World Y coordinate.

winx X window-relative coordinate.

winy Y window-relative coordinate.

gnome-canvas-get-color (*self* <gnome-canvas>) (*spec* mchars) (*color* <gdk-color>) ⇒ (*ret* int) [Function]

get-color [Method]

Allocates a color based on the specified X color specification. As a convenience to item implementations, it returns TRUE if the color was allocated, or FALSE if the specification was NULL. A NULL color specification is considered as "transparent" by the canvas.

canvas A canvas.

spec X color specification, or NULL for "transparent".

color Returns the allocated color.

ret TRUE if *spec* is non-NULL and the color is allocated. If *spec* is NULL, then returns FALSE.

gnome-canvas-get-color-pixel (*self* <gnome-canvas>) (*rgba* unsigned-int) ⇒ (*ret* unsigned-long) [Function]

get-color-pixel [Method]

Allocates a color from the RGBA value passed into this function. The alpha opacity value is discarded, since normal X colors do not support it.

canvas A canvas.

rgba RGBA color specification.

ret Allocated pixel value corresponding to the specified color.

gnome-canvas-set-stipple-origin (*self* <gnome-canvas>) (*gc* <gdk-gc>) [Function]

set-stipple-origin [Method]

Sets the stipple origin of the specified GC as is appropriate for the canvas, so that it will be aligned with other stipple patterns used by canvas items. This is typically only needed by item implementations.

canvas A canvas.

gc GC on which to set the stipple origin.

`gnome-canvas-set-dither` (*self* `<gnome-canvas>`) (*dither* `<gdk-rgb-dither>`) [Function]

`set-dither` [Method]

Controls dithered rendering for antialiased canvases. The value of dither should be `<gdk-rgb-dither-none>`, `<gdk-rgb-dither-normal>`, or `<gdk-rgb-dither-max>`. The default canvas setting is `<gdk-rgb-dither-normal>`.

canvas A canvas.

dither Type of dithering used to render an antialiased canvas.

`gnome-canvas-get-dither` (*self* `<gnome-canvas>`) ⇒ (*ret* `<gdk-rgb-dither>`) [Function]

`get-dither` [Method]

Returns the type of dithering used to render an antialiased canvas.

canvas A canvas.

ret The dither setting.

19 Undocumented

The following symbols, if any, have not been properly documented.

19.1 (gnome gw canvas)

`gnome-canvas-get-center-scroll-region` [Variable]

`gnome-canvas-path-def-closepath-current` [Variable]

`gnome-canvas-path-def-has-currentpoint` [Variable]

`gnome-canvas-rich-text-copy-clipboard` [Variable]

`gnome-canvas-rich-text-cut-clipboard` [Variable]

`gnome-canvas-rich-text-get-iter-at-location` [Variable]

`gnome-canvas-rich-text-get-iter-location` [Variable]

`gnome-canvas-rich-text-paste-clipboard` [Variable]

`gnome-canvas-set-center-scroll-region` [Variable]

Type Index

Function Index

A

C

D

E

G

H

I

L

M

R

S

T

U

W