PyGTK 2.0 Reference Manual

John Finlay

Version 2.5.2

March 5, 2005

Revision History	
Revision 2.5.2	March 5, 2005
Revision 2.5.1	December 23, 2004
First release covering PyGTK 2.5.1 (GTK+ 2.6)	
Revision 2.5.0	November 15, 2004
First release covering PyGTK 2.5.x	
Revision 2.4.11	October 3, 2004
Revision 2.4.10	August 11, 2004
Revision 2.4.9	August 3, 2004
Revision 2.4.8	July 1, 2004
Revision 2.4.7	May 19, 2004
Revision 2.4.6	May 17, 2004
Revision 2.4.4	May 5, 2004
Revision 2.4.2	April 29, 2004
Revision 2.4.0	April 27, 2004
First release covering PyGTK 2.2 and 2.4	
Revision 1.9	January 28, 2004
Revision 1.8	October 7, 2003
Revision 1.7	August 23, 2003
Revision 1.6	July 20, 2003
Revision 1.5	July 17, 2003
Revision 1.4	July 16, 2003
Revision 1.3	July 15, 2003
Revision 1.2	July 12, 2003
Revision 1.1	July 11, 2003
Revision 1.0	July 2, 2003
Frist release for PyGTK 2.0	

Abstract

This reference describes the classes of the Python PyGTK module.

Table of Contents

List of Examples

gtk.gdk.Atom

gtk.gdk.Atom — an object representing an interned string

Synopsis

```
class gtk.gdk.Atom:
    gtk.gdk.atom_intern(name, only_if_exists=FALSE)
```

Description

`gtk.gdk.Atom` is a `PyGTK` class that wraps the `GTK+` GdkAtom – an unsigned integer representing an interned string. An interned string is a string that has an internal `GTK+` mapping between an atom and a string. `gtk.gdk.Atom` has no methods. A copy of the interned string can be retrieved by using the Python `str()` function. gtk.gdk.atom_intern() will return a `gtk.gdk.Atom` referencing an existing interned string but will intern a string if it isn't already interned.

`gtk.gdk.Atom` objects are used to provide the targets for `gtk.SelectionData` objects that are used by `gtk.Clipboard` and `gtk.TreeView`, objects and for drag and drop (see `gtk.gdk.DragContext`, `gtk.gdk.Window` and `gtk.TreeDragSource` and `gtk.Widget` for more information). The advantage of the interned string is that it is easy to pass between processes and even systems since only an integer value is passed.

`PyGTK` 2.4 has a number of pre–defined atoms that map to builtin `GTK+` interned string atoms. The interned strings and the `PyGTK` 2.4 `gtk.gdk.Atom` objects are:

"PRIMARY"	`gtk.gdk.SELECTION_PRIMARY`
"SECONDARY"	`gtk.gdk.SELECTION_SECONDARY`
"CLIPBOARD"	`gtk.gdk.SELECTION_CLIPBOARD`
"ATOM"	`gtk.gdk.SELECTION_TYPE_ATOM`
"BITMAP"	`gtk.gdk.TARGET_BITMAP` or `gtk.gdk.SELECTION_TYPE_BITMAP`
"COLORMAP"	`gtk.gdk.TARGET_COLORMAP` or `gtk.gdk.SELECTION_TYPE_COLORMAP`

"DRAWABLE"	`gtk.gdk.TARGET_DRAWABLE` or `gtk.gdk.SELECTION_TYPE_DRAWABLE`
"PIXMAP"	`gtk.gdk.TARGET_PIXMAP` or `gtk.gdk.SELECTION_TYPE_PIXMAP`
"STRING"	`gtk.gdk.TARGET_STRING` or `gtk.gdk.SELECTION_TYPE_STRING`
"WINDOW"	`gtk.gdk.SELECTION_TYPE_WINDOW`

Also in `PyGTK` 2.4, comparison between a string and a `gtk.gdk.Atom` is supported.

Constructor

```
    gtk.gdk.atom_intern(name, only_if_exists=FALSE)
```

`name` :	the string to be interned or retrieved
`only_if_exists` :	this value is ignored
Returns :	a new `gtk.gdk.Atom` or `None`

Creates a `gtk.gdk.Atom` referencing the interned string specified by *`name`*. An interned string is a string that has a `GTK+` mapping to an unsigned integer value. This constructor will intern the string and create a `gtk.gdk.Atom` if it does not already exist. *`only_if_exists`* is ignored and essentially is always `FALSE`.

Prev	Up	Next
The gtk.gdk Class Reference	Home	gtk.gdk.Color

gtk.gdk.Color

Prev **The gtk.gdk Class Reference** Next

gtk.gdk.Color

gtk.gdk.Color — an object holding color information

Synopsis

```
class gtk.gdk.Color(gobject.GBoxed):
    gtk.gdk.Color(red=0, green=0, blue=0, pixel=0)
Functions

    def gtk.gdk.color_parse(spec)
```

Attributes

"pixel"	Read–Write	The pixel value of the color
"red"	Read–Write	The value of the red component of the color
"green"	Read–Write	The value of the green component of the color
"blue"	Read–Write	The value of the blue component of the color

Description

A `gtk.gdk.Color` contains the values of a color that may or may not be allocated. The red, green and blue attributes are specified by an unsigned integer in the range 0–65535. The pixel value is an index into the colormap that has allocated the `gtk.gdk.Color`. Typically a color is allocated by using the `gdk.Colormap.alloc_color()` method. Unallocated colors can be used to specify the color attributes of `gtk.Style` objects since these colors will be allocated when an attempt is made to use the `gtk.Style` object.

Constructor

```
gtk.gdk.Color(red=0, green=0, blue=0, pixel=0)
```

red :	The red color component in the range 0–65535
green :	The green color component in the range 0–65535
blue :	The blue color component in the range 0–65535
pixel :	The index of the color when allocated in its colormap
Returns :	a new `gtk.gdk.Color` object

Creates a new `gtk.gdk.Color` object with the color component values specified by *red*, *green* and *blue* (all default to 0) and using the pixel value specified by *pixel*. The value of *pixel* will be overwritten when the color is allocated.

Functions

gtk.gdk.color_parse

```
def gtk.gdk.color_parse(spec)
```

spec :	a string containing a color specification
Returns :	a `gtk.gdk.Color` object

The `gtk.gdk.color_parse()` method returns the `gtk.gdk.Color` specified by *spec*. The format of *spec* is a string containing the specification of the color either as a name (e.g. "navajowhite") as specified in the X11 `rgb.txt` file or as a hexadecimal string (e.g. "#FF0078"). The hexadecimal string must start with '#' and must contain 3 sets of hexadecimal digits of the same length (i.e. 1, 2 ,3 or 4 digits). For example the following specify the same color value: "#F0A", "#FF00AA", "#FFF000AAA" and "#FFFF0000AAAA". The `gtk.gdk.Color` is *not* allocated.

This function raise the ValueError (TypeError prior to PyGTK 2.4) exception if unable to parse the color specification

Prev	Up	Next
gtk.gdk.Atom	Home	gtk.gdk.Colormap

gtk.gdk.Colormap

Prev **The gtk.gdk Class Reference** Next

gtk.gdk.Colormap

gtk.gdk.Colormap — a table of color display component values

Synopsis

```
class gtk.gdk.Colormap(gobject.GObject):
    gtk.gdk.Colormap(visual, allocate)
    def alloc_color(color, writeable=FALSE, best_match=TRUE)
    def alloc_color(spec, writeable=FALSE, best_match=TRUE)
    def alloc_color(red, green, blue, writeable=FALSE, best_match=TRUE)
    def get_visual()
    def get_screen()
    def query_color(pixel)
Functions

    def gtk.gdk.colormap_get_system()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.gdk.Colormap
```

Description

A `gtk.gdk.Colormap` contains a table mapping index values to the RGB color component values used to display the colors. The index value corresponds to the pixel value of a `gtk.gdk.Color` object and the RGB component values correspond to the `gtk.gdk.Color` red, green and blue values. The `gtk.Widget.get_colormap()` method is used to retrieve the colormap used by a widget. The default system colormap is retrieved by using the `gtk.gdk.colormap_get_system()` function. The `alloc_color()` method has three signatures to allow the color specification using an unallocated `gtk.gdk.Color`, a string or a RGB trio.

Constructor

```
    gtk.gdk.Colormap(visual, allocate)
```

`visual` :	the `gtk.gdk.Visual` to use
`allocate` :	if `TRUE`, the newly created colormap will be a private colormap, and all colors in it will be allocated for the applications use.
Returns :	a `gtk.gdk.Colormap` object

Creates a new `gtk.gdk.Colormap` for the `gtk.gdk.Visual` specified by *`visual`*. If *`allocate`* is `TRUE` the colormap will be a private colormap for the sole use of the application.

Methods

gtk.gdk.Colormap.alloc_color

```
    def alloc_color(color, writeable=FALSE, best_match=TRUE)
```

`color` :	an unallocated `gtk.gdk.Color`
`writeable` :	if `TRUE` the colormap entry is writeable.
`best_match` :	if `TRUE` a best match to the requested color can be used if needed.
Returns :	a new `gtk.gdk.Color` object

The `alloc_color()` method allocates the color specified by *`color`* in the colormap. The value of color should be an unallocated `gtk.gdk.Color`. If *`writeable`* is `TRUE` the color is allocated writeable which means it can be changed after allocation but cannot be shared with another application. If *`best_match`* is `TRUE` the closest match to the color will be returned if the request could not be satisfied exactly.

This method raises:

- the ValueError (TypeError prior to PyGTK 2.4) exception if unable to parse the color specification
- the RuntimeError exception if the color could not be allocated

gtk.gdk.Colormap.alloc_color

```
    def alloc_color(spec, writeable=FALSE, best_match=TRUE)
```

`spec` :	a string containing a color specification
`writeable` :	if `TRUE` the colormap entry is writeable.
`best_match` :	if `TRUE` a best match to the requested color can be used if needed.
Returns :	a `gtk.gdk.Color` object

The `alloc_color()` method allocates the color specified by *`spec`* in the colormap. The format of *`spec`* is a string containing the specification of the color either as a name (e.g. "navajowhite") as specified in the X11 `rgb.txt` file or as a hexadecimal string (e.g. "#FF0078"). The hexadecimal string must start with '#' and must contain 3 sets of hexadecimal digits of the same length (i.e. 1, 2 ,3 or 4 digits). For example the following specify the same color value: "#F0A", "#FF00AA", "#FFF000AAA" and "#FFFF0000AAAA". If *`writeable`* is `TRUE` the color is allocated writeable which means it can be changed after allocation but cannot be shared with another application. If *`best_match`* is `TRUE` the closest match to the color will be returned if the request could not be satisfied exactly.

This method raises:

- the ValueError (TypeError prior to PyGTK 2.4) exception if unable to parse the color specification
- the RuntimeError exception if the color could not be allocated

gtk.gdk.Colormap.alloc_color

```
    def alloc_color(red, green, blue, writeable=FALSE, best_match=TRUE)
```

`red` :	the red component of the color in the range 0–65535
`green` :	the green component of the color in the range 0–65535
`blue` :	the blue component of the color in the range 0–65535
`writeable` :	a gboolean
`best_match` :	a gboolean
Returns :	a `gtk.gdk.Color` object

The `alloc_color()` method allocates the color specified by the component values *`red`*, *`green`* and *`blue`*. If *`writeable`* is `TRUE` the color is allocated writeable which means it can be changed after allocation but cannot be shared with another application. If *`best_match`* is `TRUE` the closest match to the color will be returned if the request could not be satisfied exactly.

This method raises:

- the ValueError (TypeError prior to PyGTK 2.4) exception if unable to parse the color specification
- the RuntimeError exception if the color could not be allocated

gtk.gdk.Colormap.get_visual

```
def get_visual()
```

Returns :	a `gtk.gdk.Visual`

The `get_visual()` method returns the visual the colormap was created for. See the gtk.gdk.Colormap constructor.

gtk.gdk.Colormap.get_screen

```
def get_screen()
```

Returns :	a `gtk.gdk.Screen`

The `get_screen()` method returns the `gtk.gdk.Screen` the colormap was created for.

gtk.gdk.Colormap.query_color

```
def query_color()
```

`pixel` :	a pixel value
Returns :	the `gtk.gdk.Color` corresponding to *`pixel`*

Note

This method is available in PyGTK 2.4 and above.

The `query_color()` method returns the `gtk.gdk.Color` corresponding to the hardware pixel value specified by *`pixel`*. *`pixel`* must be a valid pixel in the colormap. It's a programmer error to call this method with a pixel which is not in the colormap. Hardware pixels are normally obtained from the `alloc_color()` method, or from a `gtk.gdk.Image` object. (A `gtk.gdk.Image` contains image data in hardware format while a `gtk.gdk.Pixbuf` contains image data in a canonical 24−bit RGB format.)

Functions

gtk.gdk.colormap_get_system

```
def gtk.gdk.colormap_get_system()
```

Returns :	the system' default colormap

The `gtk.gdk.colormap_get_system()` method returns the default colormap used by the system on the default screen. See the `gtk.gdk.Screen.get_system_colormap()` method for more information.

gtk.gdk.Cursor

gtk.gdk.Cursor — standard and pixmap cursors

Synopsis

```
class gtk.gdk.Cursor(gobject.GBoxed):
    gtk.gdk.Cursor(cursor_type)
    gtk.gdk.Cursor(display, cursor_type)
    gtk.gdk.Cursor(display, pixbuf, x, y)
    gtk.gdk.Cursor(source, mask, fg, bg, x, y)
    def get_display()
```

Description

A `gtk.gdk.Cursor` represents a bitmap image used for the mouse pointer. Each `gtk.gdk.Window` can have its own cursor. By default a `gtk.gdk.Window` uses its parent's cursor. A standard set of cursors is provided in `PyGTK`:

`gtk.gdk.X_CURSOR`	✖
`gtk.gdk.ARROW`	➶
`gtk.gdk.BASED_ARROW_DOWN`	⤓
`gtk.gdk.BASED_ARROW_UP`	⤒
`gtk.gdk.BOAT`	⛵
`gtk.gdk.BOGOSITY`	⌗
`gtk.gdk.BOTTOM_LEFT_CORNER`	⇙
`gtk.gdk.BOTTOM_RIGHT_CORNER`	⇘
`gtk.gdk.BOTTOM_SIDE`	⤓
`gtk.gdk.BOTTOM_TEE`	⊥
`gtk.gdk.BOX_SPIRAL`	▣
`gtk.gdk.CENTER_PTR`	↑
`gtk.gdk.CIRCLE`	○
`gtk.gdk.CLOCK`	⏲
`gtk.gdk.COFFEE_MUG`	☕
`gtk.gdk.CROSS`	╬
`gtk.gdk.CROSS_REVERSE`	✳
`gtk.gdk.CROSSHAIR`	┼
`gtk.gdk.DIAMOND_CROSS`	◈
`gtk.gdk.DOT`	●
`gtk.gdk.DOTBOX`	⊡
`gtk.gdk.DOUBLE_ARROW`	↕
`gtk.gdk.DRAFT_LARGE`	➶
`gtk.gdk.DRAFT_SMALL`	↗
`gtk.gdk.DRAPED_BOX`	⊠
`gtk.gdk.EXCHANGE`	⇄
`gtk.gdk.FLEUR`	✥
`gtk.gdk.GOBBLER`	🦃
`gtk.gdk.GUMBY`	🚶
`gtk.gdk.HAND1`	☛
`gtk.gdk.HAND2`	☞
`gtk.gdk.HEART`	♡

`gtk.gdk.ICON`	▣
`gtk.gdk.IRON_CROSS`	⊠
`gtk.gdk.LEFT_PTR`	↖
`gtk.gdk.LEFT_SIDE`	⇤
`gtk.gdk.LEFT_TEE`	├
`gtk.gdk.LEFTBUTTON`	▣
`gtk.gdk.LL_ANGLE`	└
`gtk.gdk.LR_ANGLE`	┘
`gtk.gdk.MAN`	🕴
`gtk.gdk.MIDDLEBUTTON`	▣
`gtk.gdk.MOUSE`	🐭
`gtk.gdk.PENCIL`	✎
`gtk.gdk.PIRATE`	☠
`gtk.gdk.PLUS`	✚
`gtk.gdk.QUESTION_ARROW`	?
`gtk.gdk.RIGHT_PTR`	↗
`gtk.gdk.RIGHT_SIDE`	⇥
`gtk.gdk.RIGHT_TEE`	┤
`gtk.gdk.RIGHTBUTTON`	▣
`gtk.gdk.RTL_LOGO`	⊞
`gtk.gdk.SAILBOAT`	⛵
`gtk.gdk.SB_DOWN_ARROW`	⇓
`gtk.gdk.SB_H_DOUBLE_ARROW`	↔
`gtk.gdk.SB_LEFT_ARROW`	⇐
`gtk.gdk.SB_RIGHT_ARROW`	⇒
`gtk.gdk.SB_UP_ARROW`	⇑
`gtk.gdk.SB_V_DOUBLE_ARROW`	⇕
`gtk.gdk.SHUTTLE`	🚀
`gtk.gdk.SIZING`	⧉
`gtk.gdk.SPIDER`	✳
`gtk.gdk.SPRAYCAN`	🧯
`gtk.gdk.STAR`	☆
`gtk.gdk.TARGET`	⊙
`gtk.gdk.TCROSS`	+
`gtk.gdk.TOP_LEFT_ARROW`	↖
`gtk.gdk.TOP_LEFT_CORNER`	⇱
`gtk.gdk.TOP_RIGHT_CORNER`	⌝
`gtk.gdk.TOP_SIDE`	⤒
`gtk.gdk.TOP_TEE`	┬
`gtk.gdk.TREK`	♁
`gtk.gdk.UL_ANGLE`	┌
`gtk.gdk.UMBRELLA`	☂
`gtk.gdk.UR_ANGLE`	┐
`gtk.gdk.WATCH`	⌚
`gtk.gdk.XTERM`	⌶

Constructor

```
gtk.gdk.Cursor(cursor_type)
```

`cursor_type` :	the standard cursor to create
Returns :	a new `gtk.gdk.Cursor`

Creates the new `gtk.gdk.Cursor` from a builtin cursor specified by *`cursor_type`*. To make the cursor invisible, see the description of the gtk.gdk.Cursor() constructor that creates a cursor from a pixmap below.

```
gtk.gdk.Cursor(display, cursor_type)
```

`display` :	the `gtk.gdk.Display` to create the cursor for
`cursor_type` :	the standard cursor to create
Returns :	a new `gtk.gdk.Cursor`

Note

This constructor is available in PyGTK 2.4 and above.

Creates the new `gtk.gdk.Cursor` for the `gtk.gdk.Display` specified by *`display`* from a builtin cursor specified by *`cursor_type`*. To make the cursor invisible, see the description of the gtk.gdk.Cursor() constructor that creates a cursor from a pixmap below.

```
gtk.gdk.Cursor(display, pixbuf, x, y)
```

`display` :	the `gtk.gdk.Display` to create the cursor for
`pixbuf` :	the `gtk.gdk.Pixbuf` holding the cursor image
`x` :	the "hot spot" x offset
`y` :	the "hot spot" y offset
Returns :	a new `gtk.gdk.Cursor`

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new `gtk.gdk.Cursor` for the `gtk.gdk.Display` specified by display using the `gtk.gdk.Pixbuf` specified by *`source`* as the icon image. The "hotspot" of the cursor will be located as the position specified by *`x`* and *`y`*. To make the cursor invisible, see the description of the gtk.gdk.Cursor() constructor that creates a cursor from a pixmap below

```
gtk.gdk.Cursor(source, mask, fg, bg, x, y)
```

`source` :	the `gtk.gdk.Pixmap` holding the cursor image
`mask` :	the `gtk.gdk.Pixmap` to use as a mask
`fg` :	the unallocated foreground `gtk.gdk.Color`
`bg` :	the unallocated background `gtk.gdk.Color`
`x` :	the "hot spot" x offset
`y` :	the "hot spot" y offset
Returns :	a new `gtk.gdk.Cursor`

Creates a new `gtk.gdk.Cursor` using:

- the `gtk.gdk.Pixmap` specified by *`source`* as the icon image

- the gtk.gdk.Pixmap specified by *mask* to mask *source* (must be the same size as source)
- the gtk.gdk.Color specified by *fg* as the foreground color of the cursor
- the gtk.gdk.Color specified by *bg* as the background color of the cursor
- the horizontal offset of the cursor "hot spot" specified by *x*
- the vertical offset of the cursor "hot spot" specified by *y*

To make the cursor invisible, create a cursor from an empty gtk.gdk.Pixmap as follows:

```
  pixmap = gtk.gdk.Pixmap(None, 1, 1, 1)
  color = gtk.gdk.Color()
  cursor = gtk.gdk.Cursor(pixmap, pixmap, color, color, 0, 0)
```

Methods

gtk.gdk.Cursor.get_display

```
    def get_display()
```

Returns :	the associated gtk.gdk.Display

Note

This method is available in PyGTK 2.2 and above.

The `get_display()` method returns the gtk.gdk.Display on which the cursor is defined.

gtk.gdk.Device

gtk.gdk.Device — an object for supporting input devices

Synopsis

```
class gtk.gdk.Device(gobject.GObject):
    def set_source(source)
    def set_mode(mode)
    def set_key(index, keyval, modifiers)
    def set_axis_use(index, use)
    def get_state(window)
    def get_history(window, start, stop)
    def get_axis(axes, use)
Functions

    def gtk.gdk.devices_list()
    def gtk.gdk.device_get_core_pointer()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.gdk.Device
```

Attributes

"axes"	Read	a tuple containing axes information. Each axes is described by a tuple containing: use information (one of: `gtk.gdk.AXIS_IGNORE`, `gtk.gdk.AXIS_X`, `gtk.gdk.AXIS_Y`, `gtk.gdk.AXIS_PRESSURE`, `gtk.gdk.AXIS_XTILT`, `gtk.gdk.AXIS_YTILT`, `gtk.gdk.AXIS_WHEEL`, `gtk.gdk.AXIS_LAST`), the minimum and maximum axes values.
"has_cursor"	Read	if `TRUE` the pointer follows device motion.
"keys"	Read	a tuple describing the mapped macro buttons. Each macro button is described by a tuple containing: a key value output when the macro button is pressed and a set of modifiers output with the key value.
"mode"	Read	the mode of this device – one of: `gtk.gdk.MODE_DISABLED`, `gtk.gdk.MODE_SCREEN`, `gtk.gdk.MODE_WINDOW`
"name"	Read	the name of this device.
"num_axes"	Read	the length of the `axes` tuple.
"num_keys"	Read	the length of the `keys` tuple
"source"	Read	the type of this device – one of: `gtk.gdk.SOURCE_MOUSE`, `gtk.gdk.SOURCE_PEN`, `gtk.gdk.SOURCE_ERASER`, `gtk.gdk.SOURCE_CURSOR`

Description

In addition to the normal keyboard and mouse input devices, `PyGTK` also contains support for extended input devices. In particular, this support is targeted at graphics tablets. Graphics tablets typically return sub–pixel positioning information and possibly information about the pressure and tilt of the stylus. Under X, the support for extended devices is done through the XInput extension. Because handling extended input devices may involve considerable overhead, they need to be turned on for each `gtk.gdk.Window` individually using `gtk.gdk.Window.input_set_extension_events()`. (Or, more typically, for `gtk.Widget` objects, using the `gtk.Widget.set_extension_events()` method). As an additional complication, depending on the support from the windowing system, its possible that a normal mouse cursor will not be displayed for a particular extension device. If an application does not want to deal with displaying a cursor itself, it can ask only to get extension events from devices that will display a cursor, by passing the `gtk.gdk.EXTENSION_EVENTS_CURSOR` value to the `gtk.gdk.Window.input_set_extension_events()` method. Otherwise, the application must retrieve the device information using the `gtk.gdk.devices_list()` function, check the has_cursor field, and, if it is `FALSE`, draw a cursor itself when it receives motion events.

Each pointing device is assigned a unique integer ID; events from a particular device can be identified by the *`deviceid`* attribute in the event structure. The events generated by pointer devices have also been extended to contain *pressure*, *xtilt* and *ytilt* attributes which contain the extended information reported as additional valuators from the device. The *pressure* attribute ranges from 0.0 to 1.0, while the tilt attributes range from −1.0 to 1.0. (With −1.0 representing the maximum tilt to the left or up, and 1.0 representing the maximum tilt to the right or down.) One additional attribute in each event is the *source* attribute, which contains an enumeration value describing the type of device; this currently can be one of `gtk.gdk.SOURCE_MOUSE`, `gtk.gdk.SOURCE_PEN`, `gtk.gdk.SOURCE_ERASER`, or `gtk.gdk.SOURCE_CURSOR`. This attribute is present to allow simple applications to (for instance) delete when they detect eraser devices without having to keep track of complicated per−device settings.

Various aspects of each device may be configured. The easiest way of creating a GUI to allow the user to configure such a device is to use the `gtk.InputDialog` widget in `PyGTK`. However, even when using this widget, application writers will need to directly query and set the configuration parameters in order to save the state between invocations of the application. The configuration of devices is queried using the `gtk.gdk.devices_list()` function. Each device must be activated using the `set_mode()` method, which also controls whether the device's range is mapped to the entire screen or to a single window. The mapping of the valuators of the device onto the predefined valuator types is set using the `set_axis_use()` method. And the source type for each device can be set with the `set_source()` method.

Devices may also have associated keys or macro buttons. Such keys can be globally set to map into normal X keyboard events. The mapping is set using the `set_key()` method. The interfaces in this section will most likely be considerably modified in the future to accommodate devices that may have different sets of additional valuators than the pressure xtilt and ytilt.

Methods

gtk.gdk.Device.set_source

```
    def set_source(source)
```

source :	the source type of the device

The `set_source()` method sets the source type for the input device to the value specified by *source*. The value of *source* must be one of:

`gtk.gdk.SOURCE_MOUSE`	the device is a mouse. (This will be reported for the core pointer, even if it is something else, such as a trackball.)
`gtk.gdk.SOURCE_PEN`	the device is a stylus of a graphics tablet or similar device.
`gtk.gdk.SOURCE_ERASER`	the device is an eraser. Typically, this would be the other end of a stylus on a graphics tablet.
`gtk.gdk.SOURCE_CURSOR`	the device is a graphics tablet "puck" or similar device.

gtk.gdk.Device.set_mode

```
    def set_mode(mode)
```

mode :	the input mode
Returns :	`TRUE` if the mode was successfully changed.

The `set_mode()` method sets the input device mode to the value specified by *mode*. The value of *mode* must be one of:

`gtk.gdk.MODE_DISABLED`	the device is disabled and will not report any events.
`gtk.gdk.MODE_SCREEN`	

	the device is enabled. The device's coordinate space maps to the entire screen.
`gtk.gdk.MODE_WINDOW`	the device is enabled. The device's coordinate space is mapped to a single window. The manner in which this window is chosen is undefined, but it will typically be the same way in which the focus window for key events is determined.

gtk.gdk.Device.set_key

```
    def set_key(index, keyval, modifiers)
```

`index` :	the index of the macro button to set.
`keyval` :	the key value to generate.
`modifiers` :	the modifiers to set.

The `set_key`() method sets the key event to generate when a macro button of a device is pressed. The macro button is specified by *`index`*. The key value and modifiers generated are specified by *`keyval`* and *`modifiers`* respectively.

gtk.gdk.Device.set_axis_use

```
    def set_axis_use(index, use)
```

`index` :	the index of the axis.
`use` :	how the axis is used.

The `set_axis_use`() method sets the axis (specified by *`index`*) of the input device to be used in the fashion specified by *`use`*. The value of *`use`* must be one of:

`gtk.gdk.AXIS_IGNORE`	the axis is ignored.
`gtk.gdk.AXIS_X`	the axis is used as the x axis.
`gtk.gdk.AXIS_Y`	the axis is used as the y axis.
`gtk.gdk.AXIS_PRESSURE`	the axis is used for pressure information.
`gtk.gdk.AXIS_XTILT`	the axis is used for x tilt information.
`gtk.gdk.AXIS_YTILT`	the axis is used for y tilt information.
`gtk.gdk.AXIS_WHEEL`	the axis is used for wheel information.
`gtk.gdk.AXIS_LAST`	a constant equal to the numerically highest axis value.

gtk.gdk.Device.get_state

```
    def get_state(window)
```

`window` :	a `gtk.gdk.Window`
Returns :	a tuple containing: a tuple containing the axes data; and, the modifiers in effect.

The `get_state`() method returns a tuple containing:

- a tuple containing the axes data
- the bitmask containing the set of key modifiers in effect

The state information is relative to the `gtk.gdk.Window` specified by *`window`*.

gtk.gdk.Device.get_history

```
    def get_history(window, start, stop)
```

window :	a gtk.gdk.Window
start :	the earliest event time limit
stop :	the latest event time limit
Returns :	a tuple containing event tuples each with axes data and a timestamp

The `get_history()` method returns a tuple containing the motion history for the device with respect to the gtk.gdk.Window specified by window between the time limits specified by *start* and *stop*. The motion history is a tuple containing event motion tuples that each contain a timestamp for the event and a tuple with the axes data for the event.

gtk.gdk.Device.get_axis

```
    def get_axis(axes, use)
```

axes :	a sequence of axes values
use :	the axis use to match
Returns :	the axis value from *axes* that matches the use type or `None` if there is no match.

The `get_axis()` method returns the value in *axes* whose axis matches the specified *use*.

Functions

gtk.gdk.devices_list

```
    def gtk.gdk.devices_list()
```

Returns :	a list containing the gtk.gdk.Device objects for the default display

The `gtk.gdk.devices_list()` function returns a list containing the gtk.gdk.Device objects fro the default display.

gtk.gdk.device_get_core_pointer

```
    def gtk.gdk.device_get_core_pointer()
```

Returns :	the

The `gtk.gdk.device_get_core_pointer()` function returns the device that is used as the core pointer.

Prev	Up	Next
gtk.gdk.Cursor	Home	gtk.gdk.Display

gtk.gdk.Display

Prev	**The gtk.gdk Class Reference**	Next

gtk.gdk.Display

gtk.gdk.Display — controls the keyboard/mouse pointer grabs and a set of gtk.gdk.Screen objects

Synopsis

```
class gtk.gdk.Display(gobject.GObject):
    gtk.gdk.Display(display_name)
    def get_name()
    def get_n_screens()
    def get_screen(screen_num)
    def get_default_screen()
    def pointer_ungrab(time_=0L)
    def keyboard_ungrab(time_=0L)
    def pointer_is_grabbed()
    def beep()
    def sync()
    def close()
    def list_devices()
    def get_event()
    def peek_event()
    def put_event(event)
    def set_double_click_time(msec)
    def get_core_pointer()
    def get_pointer()
    def get_window_at_pointer()
    def flush()
    def set_double_click_distance(distance)
    def supports_cursor_alpha()
    def supports_cursor_color()
    def get_default_cursor_size()
    def get_maximal_cursor_size()
    def get_default_group()
    def supports_selection_notification()
    def supports_clipboard_persistence()
    def request_selection_notification(selection)
    def store_clipboard(clipboard_window, time_, targets)
```

```
Functions

    def gtk.gdk.display_get_default()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.gdk.Display
```

Signal Prototypes

"closed"	`def callback(widget, is_error, user_param1, ...)`

Description

Note

This object is available in PyGTK 2.2 and above.

`gtk.gdk.Display` objects provide two capabilities:

- To grab/ungrab keyboard focus and mouse pointer
- To manage and provide information about the `gtk.gdk.Screen`) objects available for this `gtk.gdk.Display`

gtk.gdk.Display objects are the GDK representation of the X Display which can be described as a workstation consisting of a keyboard, a pointing device (such as a mouse) and one or more screens. It is used to open and keep track of various gtk.gdk.Screen objects currently instantiated by the application. It is also used to grab and release the keyboard and the mouse pointer.

Constructor

```
gtk.gdk.Display(display_name)
```

display_name :	the name of the display to open
Returns :	a gtk.gdk.Display, or None if the display could not be opened.

Note

This constructor is available in PyGTK 2.2 and above.

Opens the display with the name specified by *display_name* and returns a gtk.gdk.Display object wrapping the display..

Methods

gtk.gdk.Display.get_name

```
def get_name()
```

Returns :	a string representing the display name.

Note

This method is available in PyGTK 2.2 and above.

The get_name() method returns the name of the display.

gtk.gdk.Display.get_n_screens

```
def get_n_screens()
```

Returns :	the number of display screens.

Note

This method is available in PyGTK 2.2 and above.

The get_n_screens() method returns the number of screens managed by the display.

gtk.gdk.Display.get_screen

```
def get_screen(screen_num)
```

screen_num :	the screen number
Returns :	the gtk.gdk.Screen object

Note

This method is available in PyGTK 2.2 and above.

The `get_screen()` method returns a `gtk.gdk.Screen` object for one of the screens of the display.

gtk.gdk.Display.get_default_screen

```
    def get_default_screen()
```

Returns :	the default `gtk.gdk.Screen` object for *`display`*

Note

This method is available in PyGTK 2.2 and above.

The `get_default_screen()` method returns the default `gtk.gdk.Screen` for the display

gtk.gdk.Display.pointer_ungrab

```
    def pointer_ungrab(time_=0L)
```

`time_` :	a timestamp or 0L for the current time.

Note

This method is available in PyGTK 2.2 and above.

The `pointer_ungrab()` method releases any pointer grab held by the display.

gtk.gdk.Display.keyboard_ungrab

```
    def keyboard_ungrab(time_=0L)
```

`time_` :	a timestamp or 0L for the current time.

Note

This method is available in PyGTK 2.2 and above.

The `keyboard_ungrab()` method releases any keyboard grab held by the display.

gtk.gdk.Display.pointer_is_grabbed

```
    def pointer_is_grabbed()
```

Returns :	`TRUE` if an active X pointer grab is in effect

Note

This method is available in PyGTK 2.2 and above.

The `pointer_is_grabbed()` method returns `TRUE` if the pointer is grabbed.

gtk.gdk.Display.beep

```
def beep()
```

Note

This method is available in PyGTK 2.2 and above.

The `beep`() method emits a short beep on the display.

gtk.gdk.Display.sync

```
def sync()
```

Note

This method is available in PyGTK 2.2 and above.

The `sync`() method flushes any requests queued for the windowing system and waits until all requests have been handled. This is often used for making sure that the display is synchronized with the current state of the program. This is most useful for X11. On windowing systems where requests are handled synchronously, this method will do nothing.

gtk.gdk.Display.close

```
def close()
```

Note

This method is available in PyGTK 2.2 and above.

The `close`() method closes the connection to the windowing system for the given display, and cleans up associated resources.

gtk.gdk.Display.list_devices

```
def list_devices()
```

Returns :	a list of `gtk.gdk.Device` objects.

Note

This method is available in PyGTK 2.2 and above.

The `list_devices`() method returns the list of available input devices attached to the display.

gtk.gdk.Display.get_event

```
def get_event()
```

Returns :	the next `gtk.gdk.Event` to be processed, or `None` if no events are pending..

Note

This method is available in PyGTK 2.2 and above.

The `get_event`() method returns the next `gtk.gdk.Event` to be processed for the display, fetching events from the windowing system if necessary. The returned `gtk.gdk.Event` should be freed with the `gtk.gdk.Event.free()` method

gtk.gdk.Display.peek_event

```
def peek_event()
```

Returns :	a copy of the first `gtk.gdk.Event` on the event queue, or `None` if no events are in the queue.

Note

This method is available in PyGTK 2.2 and above.

The `peek_event`() method returns a copy of the first `gtk.gdk.Event` in the the display's event queue, without removing the event from the queue. (Note that this method will not get more events from the windowing system. It only checks the events that have already been moved to the `GDK` event queue.) The returned `gtk.gdk.Event` should be freed with `gtk.gdk.Event.free()`.

gtk.gdk.Display.put_event

```
def put_event(event)
```

`event` :	a `gtk.gdk.Event`.

Note

This method is available in PyGTK 2.2 and above.

The `put_event`() method appends a copy of the given event onto the front of the event queue for the display.

gtk.gdk.Display.set_double_click_time

```
def set_double_click_time(msec)
```

`msec` :	the double click time in milliseconds (thousandths of a second)

Note

This method is available in PyGTK 2.2 and above.

The `set_double_click_time`() method sets the double click time (two clicks within this time interval count as a double click and result in a `gtk.gdk._2BUTTON_PRESS` event). Applications should *not* set this, it is a global user–configured setting.

gtk.gdk.Display.get_core_pointer

```
def get_core_pointer()
```

Returns :	the core pointer device

Note

This method is available in PyGTK 2.2 and above.

The `get_core_pointer()` method returns the core pointer device for the given display

gtk.gdk.Display.get_pointer

```
def get_pointer()
```

Returns :	a 4–tuple containing the screen that the cursor is on, the root window X and Y coordinates of the pointer and the current modifier mask

Note

This method is available in PyGTK 2.2 and above.

The `get_pointer()` method returns a 4–tuple containing the `gtk.gdk.Screen` that the pointer is on, the current location coordinates of the pointer and the current modifier mask for the display. The returned modifier mask is a combination of the GDK Modifier Constants.

gtk.gdk.Display.get_window_at_pointer

```
def get_window_at_pointer()
```

Returns :	a 3–tuple containing the `gtk.gdk.Window` under the mouse pointer and the x and y coordinates of the window origin

Note

This method is available in PyGTK 2.2 and above.

The `get_window_at_pointer()` method returns a 3–tuple containing the `gtk.gdk.Window` underneath the mouse pointer and the location of that window's origin. Returns `None` if the window under the mouse pointer is not known to `GDK` (for example, belongs to another application).

gtk.gdk.Display.flush

```
def flush()
```

Note

This method is available in PyGTK 2.4 and above.

The `flush()` method flushes any requests queued for the windowing system; this happens automatically when the main loop blocks waiting for new events, but if your application is drawing without returning control to the main loop, you may need to call this method explicitly. A common case where this method needs to be called is when an application is executing drawing commands from a thread other than the thread where the main loop is running.

This is most useful for X11. On windowing systems where requests are handled synchronously, this method will do nothing.

gtk.gdk.Display.set_double_click_distance

```
    def set_double_click_distance(distance)
```

distance :	the distance in pixels

Note

This method is available in PyGTK 2.4 and above.

The `set_double_click_distance`() method sets the double click distance (two clicks within this distance count as a double click and result in a `gtk.gdk.2BUTTON_PRESS` event). See the `set_double_click_time()` method for more information. Applications should *not* set this, it is a global user−configured setting.

gtk.gdk.Display.supports_cursor_alpha

```
    def supports_cursor_alpha()
```

Returns :	`TRUE` if cursors can have alpha channels.

Note

This method is available in PyGTK 2.4 and above.

The `supports_cursor_alpha`() method returns `TRUE` if cursors can use an 8bit alpha channel on the display. Otherwise, cursors are restricted to bilevel alpha (i.e. a mask).

gtk.gdk.Display.supports_cursor_color

```
    def supports_cursor_color()
```

Returns :	`TRUE` if cursors can have multiple colors.

Note

This method is available in PyGTK 2.4 and above.

The `supports_cursor_color`() method returns `TRUE` if multicolored cursors are supported on the display. Otherwise, cursors have only a foreground and a background color.

gtk.gdk.Display.get_default_cursor_size

```
    def get_default_cursor_size()
```

Returns :	the default cursor size.

Note

This method is available in PyGTK 2.4 and above.

The `get_default_cursor_size`() method returns the default size to use for cursors on the display.

gtk.gdk.Display.get_maximal_cursor_size

```
    def get_maximal_cursor_size()
```

Returns : a 2–tuple containing the maximal cursor width and height

Note

This method is available in PyGTK 2.4 and above.

The `get_maximal_cursor_size()` method returns a 2–tuple containing the maximum width and height to use for cursors on the display.

gtk.gdk.Display.get_default_group

```
    def get_default_group()
```

Returns : The default group leader window for the display

Note

This method is available in PyGTK 2.4 and above.

The `get_default_group()` method returns the default group leader window for all toplevel windows on the display. This window is implicitly created by `GDK`. See the `gtk.gdk.Window.set_group()` method for more information.

gtk.gdk.Display.supports_selection_notification

```
    def get_default_group()
```

Returns : `TRUE` if `gtk.gdk.EventOwnerChange` events will be sent.

Note

This method is available in PyGTK 2.6 and above.

The `supports_selection_notification()` method returns `TRUE` if `gtk.gdk.EventOwnerChange` events will be sent when the owner of a selection changes.

gtk.gdk.Display.supports_clipboard_persistence

```
    def supports_clipboard_persistence()
```

Returns : `TRUE` if the display supports clipboard persistence.

Note

This method is available in PyGTK 2.6 and above.

The `supports_clipboard_persistence()` method Returns whether the specified display supports clipboard persistence; i.e. if it's possible to store the clipboard data after an application has quit. On X11 this checks if a clipboard daemon is running.

gtk.gdk.Display.request_selection_notification

```
def request_selection_notification(selection)
```

selection :	The string (or gtk.gdk.Atom) naming the selection for which ownership change notification is requested

Note

This method is available in PyGTK 2.6 and above.

The `request_selection_notification()` method requests that `gtk.gdk.EventOwnerChange` events will be sent for changes in ownership of the atom specified by *selection*.

gtk.gdk.Display.store_clipboard

```
def store_clipboard(clipboard_window, time, targets)
```

clipboard_window :	a gtk.gdk.Window belonging to the clipboard owner
time :	a timestamp
targets :	a list of targets that should be saved, or `None` if all available targets should be saved.

Note

This method is available in PyGTK 2.6 and above.

The `store_clipboard()` method issues a request to the the clipboard manager to store the clipboard data. On X11, this is a special program that works according to the freedesktop clipboard specification, available at http://www.freedesktop.org/Standards/clipboard–manager–spec.

Functions

gtk.gdk.display_get_default

```
def gtk.gdk.display_get_default()
```

Returns :	a gtk.gdk.Display, or `None` if there is no default display.

Note

This function is available in PyGTK 2.2 and above.

The `gtk.gdk.display_get_default()` function returns the default gtk.gdk.Display. This is a convenience function for:

```
display_manager = gtk.gdk.display_manager_get()
display = display.manager.get_default_display()
```

Signals

The "closed" gtk.gdk.Display Signal

```
  def callback(display, is_error, user_param1, ...)
```

display :	the display that received the signal
is_error :	`TRUE` if the display was closed due to an error
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "closed" signal is emitted when the connection to the windowing system for *display* is closed.

Prev	Up	Next
gtk.gdk.Device	Home	gtk.gdk.DisplayManager

gtk.gdk.DisplayManager

Prev **The gtk.gdk Class Reference** Next

gtk.gdk.DisplayManager

gtk.gdk.DisplayManager — maintains a list of all open `gtk.gdk.Display` objects

Synopsis

```
class gtk.gdk.DisplayManager(gobject.GObject):
    def get_default_display()
    def set_default_display(display)
    def list_displays()
Functions

    def gtk.gdk.display_manager_get()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.gdk.DisplayManager
```

Properties

"default–display"	Read–Write	The default `gtk.gdk.Display`. Available in GTK+ 2.2 and above.

Signal Prototypes

"display–opened"	`def callback(displaymanager, display, user_param1, ...)`

Description

Note

This object is available in PyGTK 2.2 and above.

The `gtk.gdk.DisplayManager` is a singleton object that offers notification when displays appear or disappear or the default display changes. The global `gtk.gdk.DisplayManager` object is returned from the gtk.gdk.display_manager_get() function.

Methods

gtk.gdk.DisplayManager.get_default_display

```
def get_default_display()
```

Returns :	a gtk.gdk.Display, or None if there is no default display.

Note

This method is available in PyGTK 2.2 and above.

The get_default_display() method returns the default gtk.gdk.Display.

gtk.gdk.DisplayManager.set_default_display

```
def set_default_display(display)
```

display :	a gtk.gdk.Display

Note

This method is available in PyGTK 2.2 and above.

The set_default_display() method sets the gtk.gdk.Display specified by *display* as the default display.

gtk.gdk.DisplayManager.list_displays

```
def list_displays()
```

Returns :	a list containing gtk.gdk.Display objects.

Note

This method is available in PyGTK 2.2 and above.

The list_displays() method returns a list containing all currently open displays.

Functions

gtk.gdk.display_manager_get

```
def gtk.gdk.display_manager_get()
```

Returns :	the singleton gtk.gdk.DisplayManager object.

Note

This function is available in PyGTK 2.2 and above.

The `gtk.gdk.display_manager_get()` method returns the global `gtk.gdk.DisplayManager` singleton.

Signals

The "display−opened" gtk.gdk.DisplayManager Signal

```
    def callback(displaymanager, display, user_param1, ...)
```

displaymanager :	the displaymanager that received the signal
display :	the display that was opened
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.2 and above.

The "display−opened" signal is emitted when display is opened.

Prev	Up	Next
gtk.gdk.Display	Home	gtk.gdk.DragContext

gtk.gdk.DragContext

gtk.gdk.DragContext — an object containing the drag and drop context data

Synopsis

```
class gtk.gdk.DragContext(gobject.GObject):
    gtk.gdk.DragContext()
    def drag_status(action, time)
    def drop_reply(ok, time)
    def drop_finish(success, time)
    def drag_get_selection()
    def drag_find_window(drag_window, x_root, y_root)
    def drag_find_window_for_screen(drag_window, screen, x_root, y_root)
    def drag_motion(dest_window, protocol, x_root, y_root, suggested_action, possible_actions,
    def drag_drop(time)
    def drag_abort(time)
    def finish(success, del_, time)
    def get_source_widget()
    def set_icon_widget(widget, hot_x, hot_y)
    def set_icon_pixmap(colormap, pixmap, mask, hot_x, hot_y)
    def set_icon_pixbuf(pixbuf, hot_x, hot_y)
    def set_icon_stock(stock_id, hot_x, hot_y)
    def set_icon_default()
    def drag_drop_succeeded()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.gdk.DragContext
```

Attributes

"action"	Read	the action chosen by the destination. One of: `gtk.gdk.ACTION_COPY`, `gtk.gdk.ACTION_MOVE`, `gtk.gdk.ACTION_LINK`, `gtk.gdk.ACTION_PRIVATE`
"actions"	Read	a bitmask of actions proposed by the source when `suggested_action` is `gtk.gdk.ACTION_ASK`. A combination of: `gtk.gdk.ACTION_COPY`, `gtk.gdk.ACTION_MOVE`, `gtk.gdk.ACTION_LINK`, `gtk.gdk.ACTION_PRIVATE`
"dest_window"	Read	the destination `gtk.gdk.Window` of this drag.
"is_source"	Read	if TRUE the context is used on the source side.
"protocol"	Read	the DND protocol which governs this drag – one of: `gtk.gdk.DRAG_PROTO_MOTIF`, `gtk.gdk.DRAG_PROTO_XDND`, `gtk.gdk.DRAG_PROTO_ROOTWIN`, `gtk.gdk.DRAG_PROTO_NONE`, `gtk.gdk.DRAG_PROTO_WIN32_DROPFILES`, `gtk.gdk.DRAG_PROTO_OLE2`, `gtk.gdk.DRAG_PROTO_LOCAL`
"source_window"	Read	the source `gtk.gdk.Window` of this drag.
"start_time"	Read	a timestamp recording the start time of this drag.
"suggested_action"	Read	the action suggested by the source. One of: `gtk.gdk.ACTION_DEFAULT`, `gtk.gdk.ACTION_COPY`, `gtk.gdk.ACTION_MOVE`, `gtk.gdk.ACTION_LINK`, `gtk.gdk.ACTION_PRIVATE`, `gtk.gdk.ACTION_ASK`
"targets"	Read	a list of targets offered by the source.

Description

A `gtk.gdk.DragContext` holds information about a drag in progress. It is used for both source and destination. A `gtk.gdk.DragContext` can be created using the gtk.gdk.DragContext() constructor but since there is no way to set its attributes the new object is not much use. Useful `gtk.gdk.DragContext` objects are created as a result of the `gtk.gdk.Window.drag_begin()` or the `gtk.Widget.drag_begin()` methods.

Constructor

```
  gtk.gdk.DragContext()
```

Returns :	a new `gtk.gdk.DragContext`

Creates a new `gtk.gdk.DragContext` object.

Methods

gtk.gdk.DragContext.drag_status

```
    def drag_status(action, time)
```

action :	a drag action that will be taken when a drop happens
time :	the timestamp for this action

The `drag_status()` method sets the specified *`action`* to be taken when a drop happens. The value of *`action`* is one of:

`gtk.gdk.ACTION_DEFAULT`	
`gtk.gdk.ACTION_COPY`	Copy the data.
`gtk.gdk.ACTION_MOVE`	Move the data, i.e. first copy it, then delete it from the source using the DELETE target of the X selection protocol.
`gtk.gdk.ACTION_LINK`	Add a link to the data. Note that this is only useful if source and destination agree on what it means.
`gtk.gdk.ACTION_PRIVATE`	Special action which tells the source that the destination will do something that the source doesn't understand.
`gtk.gdk.ACTION_ASK`	Ask the user what to do with the data.

The time the action occurred is specified by *`time`*. This method is called by the drag destination in response to `drag_motion()` called by the drag source.

gtk.gdk.DragContext.drop_reply

```
    def drop_reply(ok, time)
```

ok :	if `TRUE` the drop is accepted by the destination
time :	the timestamp

The `drop_reply()` method determines if the drop is accepted or rejected according to the value specified by *`ok`*. If *`ok`* is `TRUE` the drop is accepted. *`time`* specifies the time that the drop reply occurred. This method is called by the drag destination when a drop is initiated by the drag source.

gtk.gdk.DragContext.drop_finish

```
    def drop_finish(success, time)
```

success :	if TRUE the data was received
time :	the time of the finish

The `drop_finish()` method ends a drag operation and indicates if the drop was successful by setting the value of *`success`*. *`time`* indicates the time of the drop finish. This method is called by the drag destination.

gtk.gdk.DragContext.drag_get_selection

```
    def drag_get_selection()
```

Returns :	the selection atom

The `drag_get_selection()` method returns the selection atom for the current source window.

gtk.gdk.DragContext.drag_find_window

```
    def drag_find_window(drag_window, x_root, y_root)
```

drag_window :	a gtk.gdk.Window
x_root :	the x position of the pointer in root coordinates.
y_root :	the y position of the pointer in root coordinates.
Returns :	a 2–tuple containing the drag destination gtk.gdk.Window and the drag protocol

The `drag_find_window()` method returns a 2–tuple containing the drag destination gtk.gdk.Window and the drag protocol being used. The drag protocol is one of:

`gtk.gdk.DRAG_PROTO_MOTIF`	The Motif DND protocol.
`gtk.gdk.DRAG_PROTO_XDND`	The Xdnd protocol.
`gtk.gdk.DRAG_PROTO_ROOTWIN`	An extension to the Xdnd protocol for unclaimed root window drops.
`gtk.gdk.DRAG_PROTO_NONE`	no protocol.
`gtk.gdk.DRAG_PROTO_WIN32_DROPFILES`	The simple WM_DROPFILES protocol.
`gtk.gdk.DRAG_PROTO_OLE2`	The complex OLE2 DND protocol (not implemented).
`gtk.gdk.DRAG_PROTO_LOCAL`	Intra–application DND.

drag_window is a gtk.gdk.Window and *x_root* and *y_root* specify the location of the mouse pointer in the root window.

gtk.gdk.DragContext.drag_find_window_for_screen

```
    def drag_find_window_for_screen(drag_window, screen, x_root, y_root)
```

drag_window :	a gtk.gdk.Window
screen :	a gtk.gdk.Screen
x_root :	the x position of the pointer in root coordinates.
y_root :	the y position of the pointer in root coordinates.
Returns :	a 2–tuple containing the drag destination gtk.gdk.Window and the drag protocol

Note

This method is available in PyGTK 2.2 and above.

The `drag_find_window_for_screen()` method returns a 2–tuple containing the drag destination gtk.gdk.Window and the drag protocol being used. The drag protocol is one of:

`gtk.gdk.DRAG_PROTO_MOTIF`	The Motif DND protocol.
`gtk.gdk.DRAG_PROTO_XDND`	The Xdnd protocol.
`gtk.gdk.DRAG_PROTO_ROOTWIN`	An extension to the Xdnd protocol for unclaimed root window drops.
`gtk.gdk.DRAG_PROTO_NONE`	no protocol.
`gtk.gdk.DRAG_PROTO_WIN32_DROPFILES`	The simple WM_DROPFILES protocol.
`gtk.gdk.DRAG_PROTO_OLE2`	The complex OLE2 DND protocol (not implemented).
`gtk.gdk.DRAG_PROTO_LOCAL`	Intra–application DND.

drag_window is a gtk.gdk.Window and *x_root* and *y_root* specify the location of the mouse pointer in the root window.

gtk.gdk.DragContext.drag_motion

```
    def drag_motion(dest_window, protocol, x_root, y_root, suggested_action, possible_actions, t
```

dest_window :	the destination gtk.gdk.Window the drag is moving over
protocol :	the drag protocol in use
x_root :	the x root coordinate of the mouse pointer
y_root :	the y root coordinate of the mouse pointer
suggested_action :	the suggest drag action
possible_actions :	the possible drag actions
time :	the timestamp of the drag motion
Returns :	TRUE if there is a drag destination window and the drag has paused or a drop has occurred.

The `drag_motion()` method updates the drag context when the pointer moves or the set of actions changes. This method is called by the drag source. *dest_window* specifies the drag destination `gtk.gdk.Window`; *protocol* specifies the drag protocol being used (see the `drag_find_window()` method for details); *x_root* and *y_root* specify the root window coordinates of the mouse pointer; *suggested_action* specifies the suggested drag action (see the `drag_status()` method for more detail); possible_actions specifies the possible drag actions for the drag (see the `drag_status()` method for more detail); and, *time* specifies the timestamp of the drag motion.

gtk.gdk.DragContext.drag_drop

```
    def drag_drop(time)
```

time :	the timestamp of the drag drop.

The `drag_drop()` method initiates a drop on the current drag destination at the time specified by *time*. This method is called by the drag source.

gtk.gdk.DragContext.drag_abort

```
    def drag_abort(time)
```

time :	the time of the drag abort operation

The `drag_abort()` method aborts the current drag operation at the specified *time*. No drop operation is initiated. This method is called by the drag source.

gtk.gdk.DragContext.finish

```
    def finish(success, del_, time)
```

success :	if TRUE the drop was completed
del_ :	if TRUE the drag source should delete the source data
time :	the time of the drag finish operation.

The `finish()` method informs the drag source that the drop is finished, and that the data of the drag will no longer be required. If *success* is TRUE the drag drop completed successfully; if *del_* is TRUE the source data should be deleted; *time* is the timestamp of the finish operation. This method is called by the drag destination.

gtk.gdk.DragContext.get_source_widget

```
def get_source_widget()
```

Returns :	the source `gtk.Widget` if the drag is within the same application or `None` otherwise.

The `get_source_widget()` method returns the source `gtk.Widget` if the drag is within the application; otherwise `None` is returned.

gtk.gdk.DragContext.set_icon_widget

```
def set_icon_widget(widget, hot_x, hot_y)
```

`widget` :	a toplevel window to use as an icon.
`hot_x` :	the X offset within *`widget`* of the hotspot.
`hot_y` :	the Y offset within *`widget`* of the hotspot.

The `set_icon_widget()` method changes the icon for a drag source to the specified *`widget`* with its hot spot at the offset specified by *`hot_x`* and *`hot_y`*. `PyGTK` will not destroy the icon, so if you don't want it to persist, you should connect to the "drag_end" signal and destroy it yourself.

gtk.gdk.DragContext.set_icon_pixmap

```
def set_icon_pixmap(colormap, pixmap, mask, hot_x, hot_y)
```

`colormap` :	the colormap of the icon
`pixmap` :	the `gtk.gdk.Pixmap` image data for the icon
`mask` :	the `gtk.gdk.Pixmap` transparency mask for the icon
`hot_x` :	the X offset within *`pixmap`* of the hotspot.
`hot_y` :	the Y offset within *`pixmap`* of the hotspot.

The `set_icon_pixmap()` method sets *`pixmap`* as the icon for the drag. *`mask`* is a bitmap mask for *`pixmap`* and *`hot_x`* and *`hot_y`* specify the offset of the hot spot in *`pixmap`*. In general, `gtk.gdk.DragContext.set_icon_pixbuf()` will be more convenient to use.

gtk.gdk.DragContext.set_icon_pixbuf

```
def set_icon_pixbuf(pixbuf, hot_x, hot_y)
```

`pixbuf` :	the `gtk.gdk.Pixbuf` to use as the drag icon.
`hot_x` :	the X offset within *`pixbuf`* of the hotspot.
`hot_y` :	the Y offset within *`pixbuf`* of the hotspot.

The `set_icon_pixbuf()` method sets *`pixbuf`* as the icon for the drag. *`hot_x`* and *`hot_y`* specify the offset of the hot spot within *`pixbuf`*.

gtk.gdk.DragContext.set_icon_stock

```
def set_icon_stock(stock_id, hot_x, hot_y)
```

`stock_id` :	the ID of the stock icon to use for the drag.
`hot_x` :	the X offset within the icon of the hotspot.
`hot_y` :	the Y offset within the icon of the hotspot.

The `set_icon_stock()` method sets the the icon for a given drag from a stock ID specified by *`stock_id`*. *`hot_x`* and *`hot_y`* specify the offset of the hot spot within the stock icon.

gtk.gdk.DragContext.set_icon_default

```
    def set_icon_default()
```

The `set_icon_default()` method sets the icon for the drag to the default icon.

gtk.gdk.DragContext.drag_drop_succeeded

```
    def drag_drop_succeeded()
```

Returns :	`TRUE` if the drop was successful.

The `drag_drop_succeeded()` method returns `TRUE` if the dropped data has been successfully transferred. This method is intended to be used while handling a `gtk.gdk.DROP_FINISHED` event, its return value is meaningless at other times.

gtk.gdk.Drawable

gtk.gdk.Drawable a base class for drawing methods

Synopsis

```
class gtk.gdk.Drawable(gobject.GObject):
    def get_size()
    def set_colormap(colormap)
    def get_colormap()
    def get_visual()
    def get_depth()
    def get_screen()
    def get_display()
    def draw_point(gc, x, y)
    def draw_line(gc, x1, y1, x2, y2)
    def draw_rectangle(gc, filled, x, y, width, height)
    def draw_arc(gc, filled, x, y, width, height, angle1, angle2)
    def draw_polygon(gc, filled, points)
    def draw_drawable(gc, src, xsrc, ysrc, xdest, ydest, width, height)
    def draw_image(gc, image, xsrc, ysrc, xdest, ydest, width, height)
    def draw_points(gc, points)
    def draw_segments(gc, segs)
    def draw_lines(gc, points)
    def draw_pixbuf(gc, pixbuf, src_x, src_y, dest_x, dest_y, width=-1, height=-1, dither=gtk.g
    def draw_glyphs(gc, font, x, y, glyphs)
    def draw_layout(gc, x, y, layout, foreground=None, background=None)
    def get_image(x, y, width, height)
    def new_gc(foreground, background, font, function, fill, tile, stipple, clip_mask, subwindo
    def draw_rgb_image(gc, x, y, width, height, dith, rgb_buf, rowstride=-1, xdith=0, ydith=0)
    def draw_rgb_32_image(gc, x, y, width, height, dith, rgb_buf, rowstride=-1, xdith=0, ydith=
    def draw_gray_image(gc, x, y, width, height, dith, buf, rowstride=-1)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.gdk.Drawable
```

Attributes

"handle"	Read	The handle of the MS Windows window associated with the drawable. Not supported on X11.
"xid"	Read	The id of the X window available with the drawable. Not supported on MS Windows

Description

A `gtk.gdk.Drawable` is a base class providing drawing primitives for its subclasses: `gtk.gdk.Pixmap` and `gtk.gdk.Window`.

These methods provide support for drawing points, lines, arcs and text onto what are called 'drawables'. Drawables, as the name suggests, are things which support drawing onto them, and are either `gtk.gdk.Window` or `gtk.gdk.Pixmap` objects.

Many of the drawing operations take a `gtk.gdk.GC` argument, which represents a graphics context. This `gtk.gdk.GC` contains a number of drawing attributes such as foreground color, background color and line width, and is used to reduce the number of arguments needed for each drawing operation. See the `Graphics Contexts` section for more information.

Some of the drawing operations take Pango objects like `pango.Context` or `pango.Layout` as arguments. Use the `gtk.Widget.create_pango_context()` or `gtk.Widget.create_pango_layout()` methods to obtain these objects.

Methods

gtk.gdk.Drawable.get_size

```
    def get_size()
```

Returns : a tuple containing the drawable's width and height

The `get_size`() method returns a tuple containing the width and height of the drawable.

On the X11 platform, if the drawable is a `gtk.gdk.Window`, the returned size is the size reported in the most−recently−processed configure event, rather than the current size on the X server.

gtk.gdk.Drawable.set_colormap

```
    def set_colormap(colormap)
```

`colormap` : a `gtk.gdk.Colormap`

The `set_colormap`() method sets the `gtk.gdk.Colormap` associated with the drawable to the value specified by *`colormap`*. Normally this will happen automatically when the drawable is created; you only need to use this function if the drawable−creating function did not have a way to determine the colormap, and you then use drawable operations that require a colormap. The colormap for all drawables and graphics contexts you intend to use together should match. i.e. when using a `gtk.gdk.GC` to draw to a drawable, or copying one drawable to another, the colormaps should match.

gtk.gdk.Drawable.get_colormap

```
def get_colormap()
```

Returns :	the colormap, or `None`

The `get_colormap()` method returns the `gtk.gdk.Colormap` for the drawable or `None` if no colormap is set.

gtk.gdk.Drawable.get_visual

```
def get_visual()
```

Returns :	a `gtk.gdk.Visual`

The `get_visual()` method returns the `gtk.gdk.Visual` describing the pixel format of the drawable.

gtk.gdk.Drawable.get_depth

```
def get_depth()
```

Returns :	the number of bits per pixel

The `get_depth()` method returns the bit depth of the drawable, that is, the number of bits that make up a pixel in the drawable's visual. Examples are 8 bits per pixel, 24 bits per pixel, etc.

gtk.gdk.Drawable.get_screen

```
def get_screen()
```

Returns :	the `gtk.gdk.Screen` associated with the drawable

Note

This method is available in PyGTK 2.2 and above.

The `get_screen()` method returns the `gtk.gdk.Screen` associated with the drawable.

gtk.gdk.Drawable.get_display

```
def get_display()
```

Returns :	the `gtk.gdk.Display` associated with the drawable

Note

This method is available in PyGTK 2.2 and above.

The `get_display()` method returns the `gtk.gdk.Display` associated with the drawable.

gtk.gdk.Drawable.draw_point

```
def draw_point(gc, x, y)
```

gc :	a graphics context
x :	the X coordinate of the point in drawable coordinates
y :	the Y coordinate of the point in drawable coordinates

The `draw_point`() method draws a point at the location specified by *x* and *y* in the drawable using the `gtk.gdk.GC` graphics context specified by *gc*.

gtk.gdk.Drawable.draw_line

```
def draw_line(gc, x1, y1, x2, y2)
```

gc :	a graphics context
x1 :	the X coordinate of the first point
y1 :	the Y coordinate of the first point
x2 :	the X coordinate of the second point
y2 :	the Y coordinate of the second point

The `draw_line`() method draws a line between the two points specified by (*x1*, *y1*) and (*x2*, *y2*) using the `gtk.gdk.GC` graphics context specified by *gc*.

gtk.gdk.Drawable.draw_rectangle

```
def draw_rectangle(gc, filled, x, y, width, height)
```

gc :	a graphics context
filled :	if `TRUE` the rectangle will be filled with the foreground color
x :	the X coordinate of the top left corner
y :	the Y coordinate of the top left corner
width :	the width of the rectangle
height :	the height of the rectangle

The `draw_rectangle`() method draws a rectangle of the specified *width* and *height* with its top left corner at the location specified by (*x*, *y*) using the `gtk.gdk.GC` graphics context specified by *gc*. If *filled* is `TRUE` the rectangle will be filled with the foreground color.

Note

A rectangle drawn filled is 1 pixel smaller in both dimensions than a rectangle outlined. Calling:

```
window.draw_rectangle(gc, TRUE, 0, 0, 20, 20)
```

results in a filled rectangle 20 pixels wide and 20 pixels high. Calling:

```
window.draw_rectangle(gc, FALSE, 0, 0, 20, 20)
```

results in an outlined rectangle with corners at (0, 0), (0, 20), (20, 20), and (20, 0), which makes it 21 pixels wide and 21 pixels high.

gtk.gdk.Drawable.draw_arc

```
    def draw_arc(gc, filled, x, y, width, height, angle1, angle2)
```

gc :	a graphics context
filled :	if `TRUE` the arc will be filled with the foreground color creating a "pie slice"
x :	the X coordinate of the left edge of the bounding rectangle.
y :	the Y coordinate of the top edge of the bounding rectangle.
width :	the width of the bounding rectangle.
height :	the height of the bounding rectangle.
angle1 :	the start angle of the arc, relative to the 3 o'clock position, counter–clockwise, in 1/64ths of a degree.
angle2 :	the end angle of the arc, relative to angle1, counter–clockwise, in 1/64ths of a degree.

The `draw_arc()` method draws an arc or a filled 'pie slice' if *filled* is `TRUE`. The arc is defined by the bounding rectangle of the entire ellipse (specified by *x*, *y*, *width* and *height*), and the start and end angles of the part of the ellipse to be drawn (specified by *angle1* and *angle2*). The `gtk.gdk.GC` graphics context specified by *gc* is used to determine the drawing attributes.

gtk.gdk.Drawable.draw_polygon

```
    def draw_polygon(gc, filled, points)
```

gc :	a graphics context
filled :	if `TRUE` the polygon will be filled with the foreground color
points :	a sequence of 2–tuples

The `draw_polygon()` method draws an outlined or filled polygon using the points specified by *points*. *points* is a sequence of 2–tuples that each contain an x and y coordinate of a point. The points are connected in the order that they are specified and the last point is automatically connected to the first point. The `gtk.gdk.GC` graphics context specified by *gc* is used to determine the drawing attributes.

gtk.gdk.Drawable.draw_drawable

```
    def draw_drawable(gc, src, xsrc, ysrc, xdest, ydest, width, height)
```

gc :	a `gtk.gdk.GC` sharing the drawable's visual and colormap
src :	another `gtk.gdk.Drawable`
xsrc :	the X position in *src* of rectangle to draw
ysrc :	the Y position in *src* of rectangle to draw
xdest :	the X position in the drawable where the rectangle should be drawn
ydest :	the Y position in the drawable where the rectangle should be drawn
width :	the width of rectangle to draw, or –1 for entire *src* width
height :	the height of rectangle to draw, or –1 for entire *src* height

The draw_drawable() method copies the specified *width* x *height* area of the drawable specified by *src* at the specified coordinates (*xsrc*, *ysrc*) to the specified coordinates (*xdest*, *ydest*) in the drawable. *width* and *height* may be given as –1, to copy the entire *src* drawable. Most fields in the `gtk.gdk.GC` specified by *gc* are not used for this operation, but the clip mask or clip region will be honored.

The source and destination drawables must have the same visual and colormap, or errors will result. (On X11, failure to match visual and colormap results in a `BadMatch` error from the X server.) A common cause of this problem is an attempt to draw a bitmap to a color drawable. The way to draw a bitmap is to set the bitmap as a clip mask on your `gtk.gdk.GC`, then use the `draw_rectangle()` method to draw a rectangle clipped to the bitmap.

gtk.gdk.Drawable.draw_image

```
    def draw_image(gc, image, xsrc, ysrc, xdest, ydest, width, height)
```

gc :	a graphics context
image :	a gtk.gdk.Image
xsrc :	the left edge of the source rectangle within *image*.
ysrc :	the top edge of the source rectangle within *image*.
xdest :	the left edge of the destination within drawable.
ydest :	the top edge of the destination within drawable.
width :	the width of the area to be copied, or −1 to make the area extend to the right edge of *image*.
height :	the height of the area to be copied, or −1 to make the area extend to the bottom edge of *image*.

The `draw_image()` method draws the portion of the gtk.gdk.Image specified by the rectangle (*xsrc*, *ysrc*, *width* and *height*) onto the drawable at the location specified by *xdest* and *ydest*. The depth of the gtk.gdk.Image must match the depth of the gtk.gdk.Drawable. The gtk.gdk.GC graphics context specified by *gc* is used to determine the drawing attributes.

gtk.gdk.Drawable.draw_points

```
    def draw_points(gc, points)
```

gc :	a graphics context
points :	a sequence of 2−tuples

The `draw_points()` method draws the set of points specified by *points* on the drawable using the gtk.gdk.GC graphics context specified by *gc*. *points* is a sequence of 2−tuples each containing a pair of x and y coordinates of a point location in the drawable.

gtk.gdk.Drawable.draw_segments

```
    def draw_segments(gc, segs)
```

gc :	a graphics context
segs :	a sequence of 4−tuples

The `draw_segments()` method draws a set of line segments specified by *segs* on the drawable using the gtk.gdk.GC graphics context specified by *gc* to specify the drawing attributes. *segs* is a sequence of 4−tuples each containing the coordinates of the start and end points of the line segment in the format (x1, y1, x2, y2).

gtk.gdk.Drawable.draw_lines

```
    def draw_lines(gc, points)
```

gc :	a graphics context
points :	a sequence of 2−tuples

The `draw_lines()` method draws a series of lines connecting the points specified by *points*. *points* is a sequence of 2−tuples each containing the x and y coordinates of a point location. The gtk.gdk.GC graphics context specified by *gc* is used to determine the drawing attributes.The style of joins between lines is determined by the cap style attribute in the gtk.gdk.GC. This can be set with the gtk.gdk.GC.set_line_attributes() method.

gtk.gdk.Drawable.draw_pixbuf

```
    def draw_pixbuf(gc, pixbuf, src_x, src_y, dest_x, dest_y, width=-1, height=-1, dither=gtk.gc
```

gc :	a gtk.gdk.GC, used for clipping, or None
pixbuf :	a gtk.gdk.Pixbuf
src_x :	Source X coordinate within pixbuf.
src_y :	Source Y coordinate within pixbuf.
dest_x :	Destination X coordinate within drawable.
dest_y :	Destination Y coordinate within drawable.
width :	Width of region to render, in pixels, or −1 to use pixbuf width. Must be specified in PyGTK 2.2.
height :	Height of region to render, in pixels, or −1 to use pixbuf height. Must be specified in PyGTK 2.2
dither :	Dithering mode for GdkRGB.
x_dither :	X offset for dither.
y_dither :	Y offset for dither.

Note

This method is available in PyGTK 2.2 and above.

The draw_pixbuf() method renders a rectangular portion of a gtk.gdk.Pixbuf specified by *pixbuf* to the drawable using the gtk.gdk.GC specified by *gc*. The portion of *pixbuf* that is rendered is specified by the origin point (*src_x src_y*) and the *width* and *height* arguments. *pixbuf* is rendered to the location in the drawable specified by (*dest_x dest_y*). *dither* specifies the dithering mode as one of:

gtk.gdk.RGB_DITHER_NONE	Never use dithering.
gtk.gdk.RGB_DITHER_NORMAL	Use dithering in 8 bits per pixel (and below) only.
gtk.gdk.RGB_DITHER_MAX	Use dithering in 16 bits per pixel and below.

The destination drawable must have a colormap. All windows have a colormap, however, pixmaps only have colormap by default if they were created with a non−None window argument. Otherwise a colormap must be set on them with the gtk.gdk.Drawable.set_colormap() method.

On older X servers, rendering pixbufs with an alpha channel involves round trips to the X server, and may be somewhat slow. The clip mask of *gc* is ignored, but clip rectangles and clip regions work fine.

gtk.gdk.Drawable.draw_glyphs

```
    def draw_glyphs(gc, font, x, y, glyphs)
```

gc :	a gtk.gdk.GC
font :	the font to be used
x :	the X coordinate of baseline origin
y :	the Y coordinate of baseline origin
glyphs :	the glyphs to render

The draw_glyphs() method draws the sequence of glyphs (characters in a font) specified by *glyphs* at the location specified by *x* and *y* using the font specified by *font*. Instead of using this method 99% of text rendering should be done using the draw_layout() method.

gtk.gdk.Drawable.draw_layout

```
    def draw_layout(gc, x, y, layout, foreground=None, background=None)
```

gc :	base graphics context to use
x :	the X position of the left of the layout (in pixels)
y :	the Y position of the top of the layout (in pixels)
layout :	a pango.Layout
foreground :	a gtk.gdk.Color to override the foreground color or None
background :	a gtk.gdk.Color to override the background color or None

The draw_layout() method renders the pango.Layout specified by *layout* onto the drawable at the location specified by *x* and *y*. If *foreground* or *background* has a value other than None it is used to override the corresponding attribute specified by *gc*.

gtk.gdk.Drawable.get_image

```
    def get_image(x, y, width, height)
```

x :	the X coordinate on the drawable
y :	the Y coordinate on the drawable
width :	the width of region to get
height :	the height or region to get
Returns :	a gtk.gdk.Image containing the contents of the drawable

The get_image() method returns a gtk.gdk.Image object containing a copy of the region in the drawable specified by *x*, *y*, *width* and *height*. A gtk.gdk.Image stores client–side image data (pixels). In contrast, a gtk.gdk.Pixmap and gtk.gdk.Window are server–side objects. The get_image() method retrieves the pixels from a server–side drawable as a client–side gtk.gdk.Image. The format of a gtk.gdk.Image depends on the gtk.gdk.Visual of the current display, which makes manipulating gtk.gdk.Image extremely difficult; therefore, in most cases you should use the gtk.gdk.Pixbuf.get_from_drawable() method instead of this lower–level function. A gtk.gdk.Pixbuf contains image data in a canonicalized RGB format, rather than a display–dependent format. Of course, there's a convenience vs. speed tradeoff here, so you'll want to think about what makes sense for your application.

You would usually copy image data to the client side if you intend to examine the values of individual pixels, for example to darken an image or add a red tint. It would be prohibitively slow to make a round–trip request to the windowing system for each pixel, so instead you get all of them at once, modify them, then copy them all back at once. If the X server or other windowing system backend is on the local machine, this function may use shared memory to avoid copying the image data. If the source drawable is a gtk.gdk.Window and partially off screen or obscured, then the obscured portions of the returned image will contain undefined data.

gtk.gdk.Drawable.new_gc

```
    def new_gc(foreground, background, font, function, fill, tile, stipple, clip_mask, subwindow
```

foreground :	the foreground gtk.gdk.Color
background :	the background gtk.gdk.Color
font :	a font (deprecated and ignored)
function :	the bitwise operator used to combine the existing pixel value and a new pixel value – usually one of: gtk.gdk.COPY, gtk.gdk.XOR or gtk.gdk.INVERT.

`fill` :	the fill style – one of: `gtk.gdk.SOLID`, `gtk.gdk.TILED`, `gtk.gdk.STIPPLED`, `gtk.gdk.OPAQUE_STIPPLED`
`tile` :	a `gtk.gdk.Pixmap` used for tiling the background
`stipple` :	a `gtk.gdk.Pixmap` used for stippling the background
`clip_mask` :	a `gtk.gdk.Pixmap` of depth 1 used to mask pixels to be drawn
`subwindow_mode` :	the mode of drawing on subwindows in a `gtk.gdk.Window` – one of: `gtk.gdk.CLIP_BY_CHILDREN` or `gtk.gdk.INCLUDE_INFERIORS`
`ts_x_origin` :	the X coordinate of the origin of *`tile`* or *`stipple`*
`ts_y_origin` :	the Y coordinate of the origin of *`tile`* or *`stipple`*
`clip_x_origin` :	the X coordinate of the origin of *`clip_mask`*
`clip_y_origin` :	the Y coordinate of the origin of *`clip_mask`*
`graphics_exposures` :	if `TRUE` graphics exposures are enabled for calls to the `draw_drawable()` method.
`line_width` :	the line width in pixels
`line_style` :	the line style – one of: `gtk.gdk.LINE_SOLID`, `gtk.gdk.LINE_ON_OFF_DASH`, `gtk.gdk.LINE_DOUBLE_DASH`
`cap_style` :	the style of line ends – one of: `gtk.gdk.CAP_NOT_LAST`, `gtk.gdk.CAP_BUTT`, `gtk.gdk.CAP_ROUND`, `gtk.gdk.CAP_PROJECTING`
`join_style` :	the style of line joins – one of: `gtk.gdk.JOIN_MITER`, `gtk.gdk.JOIN_ROUND`, `gtk.gdk.JOIN_BEVEL`
Returns :	a graphics context

The `new_gc()` method creates a new `gtk.gdk.GC` object with the attributes as specified by the arguments. Since there are a large number of parameters it's probably best to specify the attributes using keywords. Any attributes not specified will use a default value.

gtk.gdk.Drawable.draw_rgb_image

```
    def draw_rgb_image_dithalign(gc, x, y, width, height, dith, rgb_buf, rowstride=-1, xdith=0,
```

`gc` :	a graphics context
`x` :	the X coordinate of the top–left corner in the drawable.
`y` :	the Y coordinate of the top–left corner in the drawable.
`width` :	the width of the image to be drawn.
`height` :	the height of the image to be drawn.
`dith` :	a dither value – one of: `gtk.gdk.RGB_DITHER_NONE`, `gtk.gdk.RGB_DITHER_NORMAL`, `gtk.gdk.RGB_DITHER_MAX`
`rgb_buf` :	the pixel data, represented as packed 24–bit data.
`rowstride` :	the number of bytes from the start of one row in *`rgb_buf`* to the start of the next or –1 to calculate the number of bytes.
`xdith` :	an X offset for dither alignment.
`ydith` :	a Y offset for dither alignment.

The `draw_rgb_image()` method draws an RGB image in the drawable, with an adjustment for dither alignment. This method is useful when drawing dithered images into a window that may be scrolled. Pixel (x, y) will be drawn dithered as if its actual location is (x + xdith, y + ydith). Thus, if you draw an image into a window using zero dither alignment, then scroll up one pixel, subsequent draws to the window should have ydith = 1. Setting the dither alignment correctly allows updating of small parts of the screen while avoiding visible "seams" between the different dither textures.

gtk.gdk.Drawable.draw_rgb_32_image

```
    def draw_rgb_32_image(gc, x, y, width, height, dith, rgb_buf, rowstride=-1, xdith=0, ydith=(
```

gc :	a graphics context
x :	the X coordinate of the top–left corner in the drawable.
y :	the Y coordinate of the top–left corner in the drawable.
width :	the width of the image to be drawn.
height :	the height of the image to be drawn.
dith :	a dither value – one of: `gtk.gdk.RGB_DITHER_NONE`, `gtk.gdk.RGB_DITHER_NORMAL`, `gtk.gdk.RGB_DITHER_MAX`
buf :	the pixel data, represented as padded 32–bit data.
rowstride :	the number of bytes from the start of one row in *buf* to the start of the next or –1 to calculate the number of bytes.
xdith :	an X offset for dither alignment.
ydith :	a Y offset for dither alignment.

The `draw_rgb_32_image()` method draws a padded RGB image in the drawable. The image is stored as one pixel per 32–bit word. It is laid out as a red byte, a green byte, a blue byte, and a padding byte. Otherwise this method works the same as the `draw_rgb_image()` method.

gtk.gdk.Drawable.draw_gray_image

```
    def draw_gray_image(gc, x, y, width, height, dith, buf, rowstride=-1)
```

gc :	a graphics context
x :	the X coordinate of the top–left corner in the drawable.
y :	the Y coordinate of the top–left corner in the drawable.
width :	the width of the image to be drawn.
height :	the height of the image to be drawn.
dith :	a dither value – one of: `gtk.gdk.RGB_DITHER_NONE`, `gtk.gdk.RGB_DITHER_NORMAL`, `gtk.gdk.RGB_DITHER_MAX`
buf :	the pixel data, represented as 8–bit gray values.
rowstride :	the number of bytes from the start of one row in *buf* to the start of the next or –1 to calculate the number of bytes.

The `draw_gray_image()` method draws a grayscale image on the drawable at the location specified by *x* and *y* with the image data in *buf*.

Prev	Up	Next
gtk.gdk.DragContext	Home	gtk.gdk.Event

gtk.gdk.Event

Prev **The gtk.gdk Class Reference** Next

gtk.gdk.Event

gtk.gdk.Event — an object representing an event from the windowing system

Synopsis

```
class gtk.gdk.Event(gobject.GBoxed):
    gtk.gdk.Event(type)
    def put()
    def copy()
    def free()
    def get_time()
    def get_state(state)
    def get_coords()
    def get_root_coords()
    def get_axis(axis_use)
    def set_screen(screen)
    def get_screen()
```

```
Functions

    def gtk.gdk.events_pending()
    def gtk.gdk.event_peek()
    def gtk.gdk.event_get()
    def gtk.gdk.event_get_graphics_expose(window)
    def gtk.gdk.set_show_events(show_events)
    def gtk.gdk.get_show_events()
```

Attributes

The attributes available for a `gtk.gdk.Event` are dependent on the type of the event. The event types are described in the Description section.

The following attributes are available to all different kind of events:

"type"	Read	The event type – see the list below in the Description section
"window"	Read/Write	The `gtk.gdk.Window` the event occurred on.
"send_event"	Read/Write	`TRUE` if the event was sent explicitly.

gtk.gdk.EXPOSE

"area"	Read/Write	The bounding box of the area to be redrawn
"count"	Read/Write	The number of contiguous gtk.gdk.EXPOSE events following this one. The only use for this is "exposure compression", i.e. handling all contiguous gtk.gdk.EXPOSE events in one go, though `PyGTK` performs some exposure compression so this is not normally needed.

gtk.gdk.MOTION_NOTIFY

"time"	Read/Write	The time of the event in milliseconds.
"x"	Read/Write	The x coordinate of the pointer relative to the window.
"y"	Read/Write	The y coordinate of the pointer relative to the window.
"axes"	Read	*x*, *y* translated to the axes of *device*, or `None` if *device* is the mouse.
"state"	Read/Write	A bit–mask representing the state of the modifier keys (e.g. **Control**, **Shift** and **Alt**) and the pointer buttons.
"is_hint"	Read/Write	`TRUE` if the `gdk.POINTER_MOTION_HINT_MASK` is set.
"device"	Read	The device where the event originated.
"x_root"	Read/Write	The x coordinate of the pointer relative to the root of the screen.
"y_root"	Read/Write	The y coordinate of the pointer relative to the root of the screen.

gtk.gdk.BUTTON_PRESS

gtk.gdk._2BUTTON_PRESS

gtk.gdk._3BUTTON_PRESS

gtk.gdk.BUTTON_RELEASE

"time"	Read/Write	The time of the event in milliseconds.
"x"	Read/Write	The x coordinate of the pointer relative to the window.
"y"	Read/Write	The y coordinate of the pointer relative to the window.
"axes"	Read	*x*, *y* translated to the axes of *device*, or `None` if *device* is the mouse.
"state"	Read/Write	A bit–mask representing the state of the modifier keys (e.g. **Control**, **Shift** and **Alt**) and the pointer buttons.
"button"	Read/Write	The button which was pressed or released, numbered from 1 to 5. Normally button 1 is the left mouse button, 2 is the middle button, and 3 is the right button. On 2–button mice, the middle button can often be simulated by pressing both mouse buttons together.
"device"	Read	The device where the event originated.
"x_root"	Read/Write	The x coordinate of the pointer relative to the root of the screen.
"y_root"	Read/Write	The y coordinate of the pointer relative to the root of the screen.

gtk.gdk.KEY_PRESS

gtk.gdk.KEY_RELEASE

"time"	Read/Write	The time of the event in milliseconds.
"state"	Read/Write	A bit–mask representing the state of the modifier keys (e.g. **Control**, **Shift** and **Alt**) and the pointer buttons.
"keyval"	Read/Write	The key that was pressed or released.
"string"	Read/Write	A multi–byte string containing the composed characters resulting from the key press. When text is being input, in a `gtk.Entry` for example, it is these characters which should be added to the input buffer. When using Input Methods to support internationalized text input, the composed characters appear here after the pre–editing has been completed.
"hardware_keycode"	Read/Write	The raw code of the key that was pressed or released. Available in PyGTK 2.2 and above.
"group"	Read/Write	the keyboard group. Available in PyGTK 2.4 and above.

`gtk.gdk.ENTER_NOTIFY`

`gtk.gdk.LEAVE_NOTIFY`

"subwindow"	Read	The window that was entered or left.
"time"	Read/Write	The time of the event in milliseconds.
"x"	Read/Write	The x coordinate of the pointer relative to the window.
"y"	Read/Write	The y coordinate of the pointer relative to the window.
"x_root"	Read/Write	The x coordinate of the pointer relative to the root of the screen.
"y_root"	Read/Write	The y coordinate of the pointer relative to the root of the screen.
"mode"	Read/Write	The crossing mode (`gtk.gdk.CROSSING_NORMAL`, `gtk.gdk.CROSSING_GRAB` or `gtk.gdk.CROSSING_UNGRAB`).
"detail"	Read/Write	The kind of crossing that happened (`gtk.gdk.NOTIFY_INFERIOR`, `gtk.gdk.NOTIFY_ANCESTOR`, `gtk.gdk.NOTIFY_VIRTUAL`, `gtk.gdk.NOTIFY_NONLINEAR` or `gtk.gdk.NOTIFY_NONLINEAR_VIRTUAL`).
"focus"	Read/Write	`TRUE` if *`window`* is the focus window or an inferior.
"state"	Read/Write	A bit–mask representing the state of the modifier keys (e.g. **Control**, **Shift** and **Alt**) and the pointer buttons.

`gtk.gdk.FOCUS_CHANGE`

"in_"	Read/Write	`TRUE` if the window has gained the keyboard focus, `FALSE` if it has lost the focus.

gtk.gdk.CONFIGURE

"x"	Read/Write	The new x coordinate of the window relative to its parent.
"y"	Read/Write	The new y coordinate of the window relative to its parent.
"width"	Read/Write	The new width of the window.
"height"	Read/Write	The new height of the window.

gtk.gdk.PROPERTY_NOTIFY

"atom"	Read	The property that was changed.
"time"	Read/Write	The time of the event in milliseconds.
"state"	Read/Write	The property was changed (`gtk.gdk.PROPERTY_NEW_VALUE`) or deleted (`gtk.gdk.PROPERTY_DELETE`).

gtk.gdk.SELECTION_CLEAR

gtk.gdk.SELECTION_REQUEST

gtk.gdk.SELECTION_NOTIFY

"selection"	Read	The selection.
"target"	Read	The target to which the selection should be converted.
"property"	Read	The property in which to place the result of the conversion.
"requestor"	Read/Write	the native window ID on which to place property.
"time"	Read/Write	The time of the event in milliseconds.

gtk.gdk.PROXIMITY_IN

gtk.gdk.PROXIMITY_OUT

"time"	Read/Write	The time of the event in milliseconds.
"device"	Read/Write	The device where the event originated.

gtk.gdk.DRAG_ENTER

gtk.gdk.DRAG_LEAVE

gtk.gdk.DRAG_MOTION

gtk.gdk.DRAG_STATUS

`gtk.gdk.DRAG_START`

`gtk.gdk.DRAG_FINISHED`

"context"	Read	The `gtk.gdk.DragContext` for the current DND operation.
"time"	Read/Write	The time of the event in milliseconds.
"x_root"	Read/Write	The x coordinate of the pointer relative to the root of the screen only set for `gtk.gdk.DRAG_MOTION` and `gtk.gdk.DROP_START`.
"y_root"	Read/Write	The y coordinate of the pointer relative to the root of the screen only set for `gtk.gdk.DRAG_MOTION` and `gtk.gdk.DROP_START`.

`gtk.gdk.CLIENT_EVENT`

"message_type"	Read/Write	The type of the message, which can be defined by the application.
"data_format"	Read/Write	The format of the data, given as the number of bits in each data element, i.e. 8, 16, or 32.
"data"	Read/Write	The data as a string of 8−bit characters.

`gtk.gdk.VISIBILITY_NOTIFY`

"state"	Read/Write	The new visibility state (`gtk.gdk.VISIBILITY_FULLY_OBSCURED`, `gtk.gdk.VISIBILITY_PARTIAL` or `gtk.gdk.VISIBILITY_UNOBSCURED`).

`gtk.gdk.SCROLL`

"time"	Read/Write	The time of the event in milliseconds.
"x"	Read/Write	The x coordinate of the pointer relative to the window.
"y"	Read/Write	The y coordinate of the pointer relative to the window.
"state"	Read/Write	A bit−mask representing the state of the modifier keys (e.g. **Control**, **Shift** and **Alt**) and the pointer buttons.
"direction"	Read/Write	The direction to scroll to (one of `gtk.gdk.SCROLL_UP`, `gtk.gdk.SCROLL_DOWN`, `gtk.gdk.SCROLL_LEFT` or `gtk.gdk.SCROLL_RIGHT`).
"device"	Read	The device where the event originated.
"x_root"	Read/Write	The x coordinate of the pointer relative to the root of the screen.
"y_root"	Read/Write	

		The y coordinate of the pointer relative to the root of the screen.

gtk.gdk.WINDOW_STATE

"changed_mask"	Read/Write	The mask specifying what flags have changed – a combination of: `gtk.gdk.WINDOW_STATE_WITHDRAWN`, `gtk.gdk.WINDOW_STATE_ICONIFIED`, `gtk.gdk.WINDOW_STATE_MAXIMIZED` and `gtk.gdk.WINDOW_STATE_STICKY`
"new_window_state"	Read/Write	The new window state – a combination of: `gtk.gdk.WINDOW_STATE_WITHDRAWN`, `gtk.gdk.WINDOW_STATE_ICONIFIED`, `gtk.gdk.WINDOW_STATE_MAXIMIZED` and `gtk.gdk.WINDOW_STATE_STICKY`

gtk.gdk.SETTING

"action"	Read/Write	What happened to the setting (`gtk.gdk.SETTING_ACTION_NEW`, `gtk.gdk.SETTING_ACTION_CHANGED` or `gtk.gdk.SETTING_ACTION_DELETED`).
"name"	Read/Write	The name of the setting.

Description

A `gtk.gdk.Event` represents an event from the windowing system. The `gtk.gdk.Event` methods are usually not used by applications since the `PyGTK` main loop generates signals and invokes the appropriate signal handler. The event types are:

`gtk.gdk.NOTHING`	a special code to indicate a null event.
`gtk.gdk.DELETE`	the window manager has requested that the toplevel window be hidden or destroyed, usually when the user clicks on a special icon in the title bar.
`gtk.gdk.DESTROY`	the window has been destroyed.
`gtk.gdk.EXPOSE`	all or part of the window has become visible and needs to be redrawn.
`gtk.gdk.MOTION_NOTIFY`	the pointer (usually a mouse) has moved.
`gtk.gdk.BUTTON_PRESS`	a mouse button has been pressed.
`gtk.gdk._2BUTTON_PRESS`	a mouse button has been double–clicked (clicked twice within a short period of time). Note that each click also generates a `gtk.gdk.BUTTON_PRESS` event.
`gtk.gdk._3BUTTON_PRESS`	a mouse button has been clicked 3 times in a short period of time. Note that each click also generates a `gtk.gdk.BUTTON_PRESS` event.
`gtk.gdk.BUTTON_RELEASE`	a mouse button has been released.
`gtk.gdk.KEY_PRESS`	a key has been pressed.
`gtk.gdk.KEY_RELEASE`	a key has been released.

`gtk.gdk.ENTER_NOTIFY`	the pointer has entered the window.
`gtk.gdk.LEAVE_NOTIFY`	the pointer has left the window.
`gtk.gdk.FOCUS_CHANGE`	the keyboard focus has entered or left the window.
`gtk.gdk.CONFIGURE`	the size, position or stacking order of the window has changed. Note that `PyGTK` discards these events for `gtk.gdk.WINDOW_CHILD` windows.
`gtk.gdk.MAP`	the window has been mapped.
`gtk.gdk.UNMA`	the window has been unmapped.
`gtk.gdk.PROPERTY_NOTIFY`	a property on the window has been changed or deleted.
`gtk.gdk.SELECTION_CLEAR`	the application has lost ownership of a selection.
`gtk.gdk.SELECTION_REQUEST`	another application has requested a selection.
`gtk.gdk.SELECTION_NOTIFY`	a selection has been received.
`gtk.gdk.PROXIMITY_IN`	an input device has moved into contact with a sensing surface (e.g. a touchscreen or graphics tablet).
`gtk.gdk.PROXIMITY_OUT`	an input device has moved out of contact with a sensing surface.
`gtk.gdk.DRAG_ENTER`	the mouse has entered the window while a drag is in progress.
`gtk.gdk.DRAG_LEAVE`	the mouse has left the window while a drag is in progress
`gtk.gdk.DRAG_MOTION`	the mouse has moved in the window while a drag is in progress.
`gtk.gdk.DRAG_STATUS`	the status of the drag operation initiated by the window has changed.
`gtk.gdk.DROP_START`	a drop operation onto the window has started.
`gtk.gdk.DROP_FINISHED`	the drop operation initiated by the window has completed.
`gtk.gdk.CLIENT_EVENT`	a message has been received from another application.
`gtk.gdk.VISIBILITY_NOTIFY`	the window visibility status has changed.
`gtk.gdk.NO_EXPOSE`	indicates that the source region was completely available when parts of a drawable were copied. This is not very useful.
`gtk.gdk.SCROLL`	a scroll had occurred for a window
`gtk.gdk.WINDOW_STATE`	the window state has changed
`gtk.gdk.SETTING`	a setting has changed

A set of bit−flags is used to indicate which events a window is to receive. Most of these masks map onto one or more of the event types above.

The `gtk.gdk.POINTER_MOTION_HINT_MASK` is a special mask which is used to reduce the number of `gtk.gdk.MOTION_NOTIFY` events received. Normally a `gtk.gdk.MOTION_NOTIFY` event is received each time the mouse moves. However, if the application spends a lot of time processing the event (updating the display, for example), it can easily lag behind the position of the mouse. When using the `gtk.gdk.POINTER_MOTION_HINT_MASK` the server will only send a single `gtk.gdk.MOTION_NOTIFY` event (which is marked as a hint) until the application asks for more, by calling the `gtk.gdk.Window.get_pointer()` method. The masks are:

- `gtk.gdk.EXPOSURE_MASK`
- `gtk.gdk.POINTER_MOTION_MASK`
- `gtk.gdk.POINTER_MOTION_HINT_MASK`
- `gtk.gdk.BUTTON_MOTION_MASK`
- `gtk.gdk.BUTTON1_MOTION_MASK`
- `gtk.gdk.BUTTON2_MOTION_MASK`
- `gtk.gdk.BUTTON3_MOTION_MASK`
- `gtk.gdk.BUTTON_PRESS_MASK`
- `gtk.gdk.BUTTON_RELEASE_MASK`
- `gtk.gdk.KEY_PRESS_MASK`

- `gtk.gdk.KEY_RELEASE_MASK`
- `gtk.gdk.ENTER_NOTIFY_MASK`
- `gtk.gdk.LEAVE_NOTIFY_MASK`
- `gtk.gdk.FOCUS_CHANGE_MASK`
- `gtk.gdk.STRUCTURE_MASK`
- `gtk.gdk.PROPERTY_CHANGE_MASK`
- `gtk.gdk.VISIBILITY_NOTIFY_MASK`
- `gtk.gdk.PROXIMITY_IN_MASK`
- `gtk.gdk.PROXIMITY_OUT_MASK`
- `gtk.gdk.SUBSTRUCTURE_MASK`
- `gtk.gdk.SCROLL_MASK`
- `gtk.gdk.ALL_EVENTS_MASK`

`gtk.gdk.ALL_EVENTS_MASK` is a combination of all the event masks.

Constructor

```
    gtk.gdk.Event(type)
```

type :	a event type – see the Description above
Returns :	a newly–allocated `gtk.gdk.Event`. The returned `gtk.gdk.Event` should be freed with `gtk.gdk.Event.free()`.

Note

This constructor is available in PyGTK 2.2 and above.

Creates a new `gtk.gdk.Event` of the given type. All fields are set to 0.

Methods

gtk.gdk.Event.put

```
    def put()
```

The `put`() method appends a copy of the given event onto the tail of the event queue.

gtk.gdk.Event.copy

```
    def copy()
```

Returns :	a copy of the event

The `copy`() method copies the event, copying or incrementing the reference count of the resources associated with it (e.g. windows and strings). The new `gtk.gdk.Event` must be freed with the `free()` method.

gtk.gdk.Event.free

```
    def free()
```

The `free`() method frees the event, freeing or decrementing any resources associated with it. Note that this method should only be called on `gtk.gdk.Event` objects returned from methods or functions such as `gtk.gdk.event_peek()`, `gtk.gdk.event_get()`, `gtk.gdk.event_get_graphics_expose()`

and copy().

gtk.gdk.Event.get_time

```
def get_time()
```

Returns :	the time stamp field from the event

The get_time() method returns the time stamp from the event, if there is one; otherwise returns 0.

gtk.gdk.Event.get_state

```
def get_state()
```

Returns :	the modifier state

Note

This method is available in PyGTK 2.4 and above.

The get_state() method returns the value of the modifier "state" field. If the event has no "state" field the empty state value (0) is returned. The "state" field contains a modifier type: a combination of:

gtk.gdk.SHIFT_MASK	The Shift key.
gtk.gdk.LOCK_MASK	A Lock key (depending on the modifier mapping of the X server this may either be CapsLock or ShiftLock).
gtk.gdk.CONTROL_MASK	The Control key.
gtk.gdk.MOD1_MASK	The fourth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier, but normally it is the Alt key).
gtk.gdk.MOD2_MASK	The fifth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
gtk.gdk.MOD3_MASK	The sixth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
gtk.gdk.MOD4_MASK	The seventh modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
gtk.gdk.MOD5_MASK	The eighth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
gtk.gdk.BUTTON1_MASK	The first mouse button.
gtk.gdk.BUTTON2_MASK	The second mouse button.
gtk.gdk.BUTTON3_MASK	The third mouse button.
gtk.gdk.BUTTON4_MASK	The fourth mouse button.
gtk.gdk.BUTTON5_MASK	The fifth mouse button.
gtk.gdk.RELEASE_MASK	Differentiates between (keyval, modifiers) pairs from key press and release events.
gtk.gdk.MODIFIER_MASK	all of the above

`gtk.gdk.NOTHING`

gtk.gdk.DELETE

gtk.gdk.DESTROY

gtk.gdk.MAP

gtk.gdk.UNMAP

gtk.gdk.NO_EXPOSE

These events does not have any additional attributes.

gtk.gdk.Event.get_coords

```
def get_coords()
```

Returns :	a tuple containing the event window x and y coordinates or an empty tuple if the event did not deliver event window coordinates

The `get_coords`() method returns a tuple containing the x and y coordinates of an event relative to the event `gtk.gdk.Window` or an empty tuple if the event did not deliver event window coordinates.

gtk.gdk.Event.get_root_coords

```
def get_root_coords()
```

Returns :	a tuple containing the root window x and y coordinates or an empty tuple if the event did not deliver root window coordinates

The `get_root_coords`() method returns a tuple containing the x and y coordinates from an event relative to the root window or an empty tuple if the event did not deliver root window coordinates.

gtk.gdk.Event.get_axis

```
def get_axis(axis_use)
```

`axis_use` :	the axis use to look for
Returns :	the value found or `None` if the axis was not found.

The `get_axis`() method returns the axis value for the axis use specified by *`axis_use`* from an event structure. The value of *`axis_use`* must be one of:

`gtk.gdk.AXIS_IGNORE`	the axis is ignored.
`gtk.gdk.AXIS_X`	the axis is used as the x axis.
`gtk.gdk.AXIS_Y`	the axis is used as the y axis.
`gtk.gdk.AXIS_PRESSURE`	the axis is used for pressure information.
`gtk.gdk.AXIS_XTILT`	the axis is used for x tilt information.
`gtk.gdk.AXIS_YTILT`	the axis is used for y tilt information.
`gtk.gdk.AXIS_WHEEL`	the axis is used for wheel information.
`gtk.gdk.AXIS_LAST`	a constant equal to the numerically highest axis value.

If an axis with the specified axis use is not found, this method returns `None`.

gtk.gdk.Event.set_screen

```
def set_screen(screen)
```

screen :	a gtk.gdk.Screen

Note

This method is available in PyGTK 2.2 and above.

The `set_screen()` method sets the `gtk.gdk.Screen` to the value of *`screen`*. The event must have been allocated by `PyGTK`, for instance, by the `gtk.gdk.Event.copy()` method.

gtk.gdk.Event.get_screen

```
def get_screen()
```

Returns :	the screen for the event

Note

This method is available in PyGTK 2.2 and above.

The `get_screen()` method returns the `gtk.gdk.Screen` for the event. The screen is typically the screen for the event window, but for events such as mouse events, it is the screen where the the pointer was when the event occurs – that is, the screen that has the root window for the event.

Functions

gtk.gdk.events_pending

```
def gtk.gdk.events_pending()
```

Returns :	`TRUE` if any events are pending

The `gtk.gdk.events_pending()` function returns `TRUE` if any events are ready to be processed.

gtk.gdk.event_peek

```
def gtk.gdk.event_peek()
```

Returns :	a copy of the first `gtk.gdk.Event` on the event queue or `None` if there is no event in the queue.

The `gtk.gdk.event_peek()` function returns a copy of the first `gtk.gdk.Event` on the event queue or `None` if there is no event on the event queue. The returned `gtk.gdk.Event` should be freed with the `free()` method.

gtk.gdk.event_get

```
def gtk.gdk.event_get()
```

Returns :	the next `gtk.gdk.Event` to be processed, or `None` if no events are pending.

The `gtk.gdk.event_get()` function returns the next `gtk.gdk.Event` to be processed or `None` if no events are available. The returned `gtk.gdk.Event` should be freed using the `free()` method.

gtk.gdk.event_get_graphics_expose

```
    def gtk.gdk.event_get_graphics_expose(window)
```

window :	a gtk.gdk.Window
Returns :	an expose gtk.gdk.Event if a GraphicsExpose was received, or None if a NoExpose event was received.

The `gtk.gdk.event_get_graphics_expose()` function waits for and returns returns an expose gtk.gdk.Event if a `GraphicsExpose` was received, or `None` if a `NoExpose` event was received.

gtk.gdk.set_show_events

```
    def gtk.gdk.set_show_events(show_events)
```

show_events :	if TRUE output event debug information

The `gtk.gdk.set_show_events()` function sets the debug events flag if *show_events* is `TRUE`. Otherwise the debug events flag is unset.

gtk.gdk.get_show_events

```
    def gtk.gdk.get_show_events()
```

Returns :	TRUE if the debug events flag is set.

The `gtk.gdk.get_show_events()` function returns the setting of the internal debug events flag.

Prev	Up	Next
gtk.gdk.Drawable	Home	gtk.gdk.GC

gtk.gdk.GC

Prev	**The gtk.gdk Class Reference**	Next

gtk.gdk.GC

gtk.gdk.GC — objects to encapsulate drawing properties.

Synopsis

```
class gtk.gdk.GC(gobject.GObject):
    gtk.gdk.GC(drawable, foreground, background, font, function, fill, tile, stipple, clip_mask
    def set_values(foreground, background, font, function, fill, tile, stipple, clip_mask, subw
    def set_foreground(color)
    def set_background(color)
    def set_function(function)
    def set_fill(fill)
    def set_tile(tile)
    def set_stipple(stipple)
    def set_ts_origin(x, y)
    def set_clip_origin(x, y)
    def set_clip_mask(mask)
    def set_clip_rectangle(rectangle)
    def set_subwindow(mode)
    def set_exposures(exposures)
    def set_line_attributes(line_width, line_style, cap_style, join_style)
    def set_dashes(dash_offset, dash_list)
    def offset(x_offset, y_offset)
    def copy(src_gc)
    def set_colormap(colormap)
    def get_colormap()
    def set_rgb_fg_color(color)
    def set_rgb_bg_color(color)
    def get_screen()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.gdk.GC
```

Attributes

"background"	Read−Write	The background `gtk.gdk.Color`.
"cap_style"	Read−Write	The style of drawing the ends of lines – one of: `gtk.gdk.CAP_NOT_LAST`, `gtk.gdk.CAP_BUTT`, `gtk.gdk.CAP_ROUND` or `gtk.gdk.CAP_PROJECTING`.
"clip_mask"	Read−Write	A `gtk.gdk.Pixmap` bitmap used to clip the drawing operation.
"clip_x_origin"	Read−Write	The x origin of the clip mask.
"clip_y_origin"	Read−Write	The y origin of the clip mask.
"fill"	Read−Write	The fill style – one of: `gtk.gdk.SOLID`, `gtk.gdk.TILED`, `gtk.gdk.STIPPLED` or `gtk.gdk.OPAQUE_STIPPLED`.
"font"	Read−Write	The default font (deprecated and unused)
"foreground"	Read−Write	The foreground `gtk.gdk.Color`.
"function"	Read−Write	A bitwise operation type to combine source and destination pixels – one of: `gtk.gdk.COPY`, `gtk.gdk.INVERT`, `gtk.gdk.XOR`, `gtk.gdk.CLEAR`, `gtk.gdk.AND`, `gtk.gdk.AND_REVERSE`, `gtk.gdk.AND_INVERT`, `gtk.gdk.NOOP`, `gtk.gdk.OR`, `gtk.gdk.EQUIV`, `gtk.gdk.OR_REVERSE`, `gtk.gdk.COPY_INVERT`, `gtk.gdk.OR_INVERT`, `gtk.gdk.NAND`, `gtk.gdk.NOR` or `gtk.gdk.SET`.
"graphics_exposures"	Read−Write	If `TRUE` graphics exposures are enabled.
"join_style"	Read−Write	The style used to join lines – one of: `gtk.gdk.JOIN_MITER`, `gtk.gdk.JOIN_ROUND` or `gtk.gdk.JOIN_BEVEL`.
"line_style"	Read−Write	The style of drawing lines – one of: `gtk.gdk.LINE_SOLID`, `gtk.gdk.LINE_ON_OFF_DASH` or `gtk.gdk.LINE_DOUBLE_DASH`.
"line_width"	Read−Write	The width of a line in pixels.
"stipple"	Read−Write	The `gtk.gdk.Pixmap` bitmap used to stipple the background.
"sub_window"	Read−Write	The mode of drawing in a `gtk.gdk.Window` with subwindows – one of: `gtk.gdk.CLIP_BY_CHILDREN` or `gtk.gdk.INCLUDE_INFERIORS`.
"tile"	Read−Write	The `gtk.gdk.Pixmap` used to tile the background.

"ts_x_origin"	Read−Write	The x origin of the tile or stipple.
"ts_y_origin"	Read−Write	The y origin of the tile or stipple.

Description

All drawing operations in `PyGTK` take a graphics context (`gtk.gdk.GC`) argument (See the `gtk.gdk.Drawable` description). A graphics context encapsulates information about the way things are drawn, such as the foreground color or line width. By using graphics contexts, the number of arguments to each drawing call is greatly reduced, and communication overhead is minimized, since identical arguments do not need to be passed repeatedly. Most values of a graphics context can be set at creation time by using `gtk.gdk.Drawable.new_gc()`, or can be set one−by−one using functions such as `set_foreground()` or by setting a `gtk.gdk.GC` attribute. A few of the values in the `gtk.gdk.GC`, such as the dash pattern, can only be set by the latter method.

Constructor

```
    gtk.gdk.GC(drawable, foreground, background, font, function, fill, tile, stipple, clip_mask,
```

drawable :	A `gtk.gdk.Drawable` (`gtk.gdk.Window` or `gtk.gdk.Pixmap`)
foreground :	the foreground `gtk.gdk.Color`
background :	the background `gtk.gdk.Color`
font :	a font (deprecated and ignored)
function :	the bitwise operator used to combine the existing pixel value and a new pixel value − usually one of: `gtk.gdk.COPY`, `gtk.gdk.XOR` or `gtk.gdk.INVERT`.
fill :	the fill style − one of: `gtk.gdk.SOLID`, `gtk.gdk.TILED`, `gtk.gdk.STIPPLED`, `gtk.gdk.OPAQUE_STIPPLED`
tile :	a `gtk.gdk.Pixmap` used for tiling the background
stipple :	a `gtk.gdk.Pixmap` used for stippling the background
clip_mask :	a `gtk.gdk.Pixmap` of depth 1 used to mask pixels to be drawn
subwindow_mode :	the mode of drawing on subwindows in a `gtk.gdk.Window` − one of: `gtk.gdk.CLIP_BY_CHILDREN` or `gtk.gdk.INCLUDE_INFERIORS`
ts_x_origin :	the X coordinate of the origin of *`tile`* or *`stipple`*
ts_y_origin :	the Y coordinate of the origin of *`tile`* or *`stipple`*
clip_x_origin :	the X coordinate of the origin of *`clip_mask`*
clip_y_origin :	the Y coordinate of the origin of *`clip_mask`*
graphics_exposures :	if `TRUE` graphics exposures are enabled for calls to the `gtk.gdk.Drawable.draw_drawable()` method.
line_width :	the line width in pixels
line_style :	the line style − one of: `gtk.gdk.LINE_SOLID`, `gtk.gdk.LINE_ON_OFF_DASH`, `gtk.gdk.LINE_DOUBLE_DASH`
cap_style :	the style of line ends − one of: `gtk.gdk.CAP_NOT_LAST`, `gtk.gdk.CAP_BUTT`, `gtk.gdk.CAP_ROUND`, `gtk.gdk.CAP_PROJECTING`

join_style :	the style of line joins – one of: `gtk.gdk.JOIN_MITER`, `gtk.gdk.JOIN_ROUND`, `gtk.gdk.JOIN_BEVEL`
Returns :	a new `gtk.gdk.GC` object

Creates a new `gtk.gdk.GC` object for the `gtk.gdk.Drawable` specified by *`drawable`* with the optional attributes as specified by the arguments. The drawable parameter must be specified but the rest of the parameters are optional. Any attributes not specified will use a default value. This is an alternative to the `gtk.gdk.Drawable.new_gc()` method. Since there are a large number of optional attribute parameters it's probably best to specify the attribute values using keywords.

Methods

gtk.gdk.GC.set_values

```
    def set_values(values, foreground, background, font, function, fill, tile, stipple, clip_mas
```

foreground :	the foreground `gtk.gdk.Color`
background :	the background `gtk.gdk.Color`
font :	a font (deprecated and ignored)
function :	the bitwise operator used to combine the existing pixel value and a new pixel value – usually one of: `gtk.gdk.COPY`, `gtk.gdk.XOR` or `gtk.gdk.INVERT`.
fill :	the fill style – one of: `gtk.gdk.SOLID`, `gtk.gdk.TILED`, `gtk.gdk.STIPPLED`, `gtk.gdk.OPAQUE_STIPPLED`
tile :	a `gtk.gdk.Pixmap` used for tiling the background
stipple :	a `gtk.gdk.Pixmap` used for stippling the background
clip_mask :	a `gtk.gdk.Pixmap` of depth 1 used to mask pixels to be drawn
subwindow_mode :	the mode of drawing on subwindows in a `gtk.gdk.Window` – one of: `gtk.gdk.CLIP_BY_CHILDREN` or `gtk.gdk.INCLUDE_INFERIORS`
ts_x_origin :	the X coordinate of the origin of *`tile`* or *`stipple`*
ts_y_origin :	the Y coordinate of the origin of *`tile`* or *`stipple`*
clip_x_origin :	the X coordinate of the origin of *`clip_mask`*
clip_y_origin :	the Y coordinate of the origin of *`clip_mask`*
graphics_exposures :	if `TRUE` graphics exposures are enabled for calls to the `gtk.gdk.Drawable.draw_drawable()` method.
line_width :	the line width in pixels
line_style :	the line style – one of: `gtk.gdk.LINE_SOLID`, `gtk.gdk.LINE_ON_OFF_DASH`, `gtk.gdk.LINE_DOUBLE_DASH`
cap_style :	the style of line ends – one of: `gtk.gdk.CAP_NOT_LAST`, `gtk.gdk.CAP_BUTT`, `gtk.gdk.CAP_ROUND`, `gtk.gdk.CAP_PROJECTING`
join_style :	the style of line joins – one of: `gtk.gdk.JOIN_MITER`, `gtk.gdk.JOIN_ROUND`, `gtk.gdk.JOIN_BEVEL`

The `set_values()` method sets the attributes of a graphics context in bulk using the optional parameters. Since there are a large number of attribute parameters it's probably best to specify the attribute values using keywords. Any attributes not specified will be left as is.

gtk.gdk.GC.set_foreground

```
    def set_foreground(color)
```

color :	the new foreground gtk.gdk.Color

The set_foreground() method sets the foreground gtk.gdk.Color to the value specified by *color*.

gtk.gdk.GC.set_background

```
    def set_background(color)
```

color :	the new background gtk.gdk.Color

The set_background() method sets the background gtk.gdk.Color to the value specified by *color*.

gtk.gdk.GC.set_function

```
    def set_function(function)
```

function :	the bitwise operator used to combine the existing pixel value and a new pixel value – usually one of: gtk.gdk.COPY, gtk.gdk.XOR or gtk.gdk.INVERT.

The set_function() method sets the function attribute to the value specified by *function*. The value of *function* must be one of:

- gtk.gdk.COPY
- gtk.gdk.INVERT
- gtk.gdk.XOR
- gtk.gdk.CLEAR
- gtk.gdk.AND
- gtk.gdk.AND_REVERSE.
- gtk.gdk.AND_INVERT
- gtk.gdk.NOOP
- gtk.gdk.OR
- gtk.gdk.EQUIV
- gtk.gdk.OR_REVERSE
- gtk.gdk.COPY_INVERT
- gtk.gdk.OR_INVERT
- gtk.gdk.NAND
- gtk.gdk.NOR
- gtk.gdk.SET

Only a couple of these values are usually useful. For colored images, only gtk.gdk.COPY, gtk.gdk.XOR and gtk.gdk.INVERT are generally useful. For bitmaps, gtk.gdk.AND and gtk.gdk.OR are also useful.

gtk.gdk.GC.set_fill

```
    def set_fill(fill)
```

fill :	the new fill mode

The set_fill() method sets the fill mode for the graphics context to the value specified by *fill*. The value of *fill* must be one of:

gtk.gdk.SOLID	draw with the foreground color.

`gtk.gdk.TILED`	draw with a tiled pixmap.
`gtk.gdk.STIPPLED`	draw using the stipple bitmap. Pixels corresponding to bits in the stipple bitmap that are set will be drawn in the foreground color; pixels corresponding to bits that are not set will be left untouched.
`gtk.gdk.OPAQUE_STIPPLED`	draw using the stipple bitmap. Pixels corresponding to bits in the stipple bitmap that are set will be drawn in the foreground color; pixels corresponding to bits that are not set will be drawn with the background color.

gtk.gdk.GC.set_tile

```
    def set_tile(tile)
```

`tile` :	a `gtk.gdk.Pixmap`

The `set_tile()` method sets the `gtk.gdk.Pixmap` specified by *`tile`* to be used for filling the background when the fill mode is `gtk.gdk.TILED`.

gtk.gdk.GC.set_stipple

```
    def set_stipple(stipple)
```

`stipple` :	a `gtk.gdk.Pixmap` bitmap

The `set_stipple()` method sets the `gtk.gdk.Pixmap` bitmap specified by *`stipple`* to be used for stippling the background. *`stipple`* will only be used if the fill mode is `gtk.gdk.STIPPLED` or `gtk.gdk.OPAQUE_STIPPLED`.

gtk.gdk.GC.set_ts_origin

```
    def set_ts_origin(x, y)
```

`x` :	the x origin of the tile or stipple
`y` :	the y origin of the tile or stipple

The `set_ts_origin()` method sets the origin of the tile or stipple to the value specified by *`x`* and *`y`*. The tile or stipple will be aligned such that the upper left corner of the tile or stipple will coincide with this point.

gtk.gdk.GC.set_clip_origin

```
    def set_clip_origin(x, y)
```

`x` :	the x origin of the clip mask
`y` :	the y origin of the clip mask

The `set_clip_origin()` method sets the origin of the clip mask to the value specified by *`x`* and *`y`*. The coordinates are interpreted relative to the upper–left corner of the destination drawable of the current operation.

gtk.gdk.GC.set_clip_mask

```
    def set_clip_mask(mask)
```

`mask` :	a `gtk.gdk.Pixmap`

The `set_clip_mask()` method sets the clip mask (a `gtk.gdk.Pixmap` bit map) to the value specified by *`mask`*. The clip mask is interpreted relative to the clip origin. See the `set_clip_origin()` method.

gtk.gdk.GC.set_clip_rectangle

```
    def set_clip_rectangle(rectangle)
```

`rectangle` :	a `gtk.gdk.Rectangle` to use for clipping

The `set_clip_rectangle()` method sets the clip mask for the graphics context from the `gtk.gdk.Rectangle` specified by *`rectangle`* and sets the clip origin to (0, 0). The clip origin can be changed using the `set_clip_origin()` method.

gtk.gdk.GC.set_subwindow

```
    def set_subwindow(mode)
```

`mode` :	the new subwindow mode

The `set_subwindow()` method sets the mode of drawing on subwindows when drawing on a `gtk.gdk.Window` to the value specified by *`mode`*. The value of mode must be one of:

`gtk.gdk.CLIP_BY_CHILDREN`	only draw onto the window itself not the subwindows.
`gtk.gdk.INCLUDE_INFERIORS`	draw onto the window and child windows.

gtk.gdk.GC.set_exposures

```
    def set_exposures(exposures)
```

`exposures` :	if `TRUE` exposure events will be generated for non−visible areas

The `set_exposures()` method sets an attribute that determines if copying non−visible portions of a drawable using this graphics context will generate exposure events for the corresponding regions of the destination drawable. If *`exposures`* is `TRUE` exposure events will be generated for non−visible areas. See the `gtk.gdk.Drawable.draw_drawable()` method.

gtk.gdk.GC.set_line_attributes

```
    def set_line_attributes(line_width, line_style, cap_style, join_style)
```

`line_width` :	the new line width in pixels
`line_style` :	the new line style
`cap_style` :	the new line end style
`join_style` :	the new line join style

The `set_line_attributes()` method sets the attributes to be used when drawing a line using the graphics context to the values specified by *`line_width`*, *`line_style`*, *`cap_style`* and *`join_style`*. The value of *`line_style`* must be one of:

`gtk.gdk.LINE_SOLID`	Lines are drawn solid.
`gtk.gdk.LINE_ON_OFF_DASH`	Lines are drawn dashed where even segments are drawn but odd segments are not drawn.
`gtk.gdk.LINE_DOUBLE_DASH`	Lines are drawn dashed where even segments are drawn normally but odd segments are drawn in the background color if the fill style is `gtk.gdk.SOLID`, or in the

	background color masked by the stipple if the fill style is `gtk.gdk.STIPPLED`.

The value of cap_style must be one of:

`gtk.gdk.CAP_NOT_LAST`	The same as `gtk.gdk.CAP_BUTT` for lines of non−zero width but for zero width lines, the final point on the line will not be drawn.
`gtk.gdk.CAP_BUTT`	The ends of the lines are drawn squared off and extending to the coordinates of the end point.
`gtk.gdk.CAP_ROUND`	The ends of the lines are drawn as semicircles with the diameter equal to the line width and centered at the end point.
`gtk.gdk.CAP_PROJECTING`	The ends of the lines are drawn squared off and extending half the width of the line beyond the end point.

The value of join_style must be one of:

`gtk.gdk.JOIN_MITER`	The sides of each line are extended to meet at an angle.
`gtk.gdk.JOIN_ROUND`	The sides of the two lines are joined by a circular arc.
`gtk.gdk.JOIN_BEVEL`	The sides of the two lines are joined by a straight line which makes an equal angle with each line.

gtk.gdk.GC.set_dashes

```
    def set_dashes(dash_offset, dash_list, n)
```

dash_offset :	the index of the length in *dash_list* to use as the firstst dash
dash_list :	the tuple or list of dash lengths in pixels

The `set_dashes()` method sets the pattern for drawing dashed lines using the tuple or list of dash lengths specified by *dash_list* with the index of the starting dash length specified by *dash_offset*. The dashes are drawn starting with the number of pixels at the offset position; then the next number of pixels is skipped; and then the next number is drawn; and so on rotating through all the *dash_list* numbers and starting over when the end is reached. For example, if *dash_list* is (2, 4, 8, 16) and the offset is 1, the dashes will be drawn as: draw 4 pixels, skip 8 pixels, draw 16 pixels, skip 2 pixels, draw 4 pixels and so on.

gtk.gdk.GC.offset

```
    def offset(x_offset, y_offset)
```

x_offset :	the amount by which to offset the graphics context in the X direction
y_offset :	the amount by which to offset the graphics context in the Y direction

The `offset()` method sets offset attributes such as the clip and tile−stipple origins of the graphics context so that drawing at x − *x_offset*, y − *y_offset* with the offset graphics context has the same effect as drawing at x, y with the original graphics context.

gtk.gdk.GC.copy

```
    def copy(src_gc)
```

src_gc :	the `gtk.gdk.GC` to copy

The `copy()` method copies the attributes of the `gtk.gdk.GC` specified by *src_gc* into this graphics context.

gtk.gdk.GC.set_colormap

```
    def set_colormap(colormap)
```

colormap :	a gtk.gdk.Colormap

The set_colormap() method sets the colormap for the graphics context to the specified *colormap*. The depth of the colormap's visual must match the depth of the drawable for which the graphics context was created.

gtk.gdk.GC.get_colormap

```
    def get_colormap()
```

Returns :	the colormap used by the graphics context

The get_colormap() method returns the colormap for the graphics context, if it exists. A graphics context will have a colormap if the drawable for which it was created has a colormap, or if a colormap was set explicitly with the set_colormap() method.

gtk.gdk.GC.set_rgb_fg_color

```
    def set_rgb_fg_color(color)
```

color :	an unallocated gtk.gdk.Color.

The set_rgb_fg_color() method sets the foreground color of a graphics context using the specified unallocated *color*. The pixel value for *color* will be determined using GdkRGB. If the colormap for the graphics context has not previously been initialized for GdkRGB, then for pseudo–color colormaps (colormaps with a small modifiable number of colors), a colorcube will be allocated in the colormap. Calling this method for a graphics context without a colormap is an error.

gtk.gdk.GC.set_rgb_bg_color

```
    def set_rgb_bg_color(color)
```

color :	an unallocated gtk.gdk.Color.

The set_rgb_bg_color() method sets the background color of a graphics context using the specified unallocated *color*. The pixel value for *color* will be determined using GdkRGB. If the colormap for the graphics context has not previously been initialized for GdkRGB, then for pseudo–color colormaps (colormaps with a small modifiable number of colors), a colorcube will be allocated in the colormap. Calling this method for a graphics context without a colormap is an error.

gtk.gdk.GC.get_screen

```
    def get_screen()
```

Returns :	the gtk.gdk.Screen for the gc

Note

This method is available in PyGTK 2.2. and above.

The get_screen() method returns the gtk.gdk.Screen on which the gc was created.

gtk.gdk.Image

gtk.gdk.Image — an area for bit−mapped graphics stored on the X Windows client.

Synopsis

```
class gtk.gdk.Image(gobject.GObject):
    gtk.gdk.Image(type, visual, width, height)
    def put_pixel(x, y, pixel)
    def get_pixel(x, y)
    def set_colormap(colormap)
    def get_colormap()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.gdk.Image
```

Description

The `gtk.gdk.Image` type represents an area for drawing graphics. It has now been superseded to a large extent by the much more flexible RGB methods (see `gtk.gdk.Drawable`).

Constructor

```
   gtk.gdk.Image(type, visual, width, height)
```

type :	a `gtk.gdk.Image` type
visual :	a `gtk.gdk.Visual`
width :	the width in pixels of the image
height :	the height in pixels of the image
Returns :	a new `gtk.gdk.Image` object

Creates a new `gtk.gdk.Image` object of the specified *`type`*, *`width`* and *`height`* and using the `gtk.gdk.Visual` specified by *`visual`*. The value of *`type`* must be one of:

`gtk.gdk.IMAGE_NORMAL`	The original X image type, which is quite slow since the image has to be transferred from the client to the server to display it.
`gtk.gdk.IMAGE_SHARED`	A faster image type, which uses shared memory to transfer the image data between client and server. However this will only be available if client and server are on the same machine and the shared memory extension is supported by the server.
`gtk.gdk.IMAGE_FASTEST`	Specifies that gtk.gdk.IMAGE_SHARED should be tried first, and if that fails then gtk.gdk.IMAGE_NORMAL will be used.

Usually using `gtk.gdk.IMAGE_FASTEST` is the best choice.

Methods

gtk.gdk.Image.put_pixel

```
def put_pixel(x, y, pixel)
```

x :	the x coordinate of the pixel to set.
y :	the y coordinate of the pixel to set.
pixel :	the pixel value to set.

The `put_pixel`() method sets the value of the pixel in the image at the location specified by *x* and *y* to the value specified by *pixel*.

gtk.gdk.Image.get_pixel

```
def get_pixel(x, y)
```

x :	the x coordinate of the pixel to get.
y :	the y coordinate of the pixel to get.
Returns :	the pixel value at the image location

The `get_pixel`() method returns the value of the pixel at the image location specified by *x* and *y*.

gtk.gdk.Image.set_colormap

```
def set_colormap(colormap)
```

colormap :	a `gtk.gdk.Colormap`

The `set_colormap`() method sets the colormap for the image to the specified *colormap*. Normally there's no need to use this method since images are created with the correct colormap if you get the image from a drawable. If you create the image from scratch, use the colormap of the drawable you intend to render the image to.

gtk.gdk.Image.get_colormap

```
def get_colormap()
```

Returns :	the colormap for the image

The `get_colormap`() method returns the colormap for a given image, if it exists. An image will have a colormap if the drawable from which it was created has a colormap, or if a colormap was set explicitly with the `set_colormap()` method.

Prev	Up	Next
gtk.gdk.GC	Home	gtk.gdk.Keymap

gtk.gdk.Keymap

Prev	**The gtk.gdk Class Reference**	Next

gtk.gdk.Keymap

gtk.gdk.Keymap — an object containing mappings of keys to key values.

Synopsis

```
class gtk.gdk.Keymap(gobject.GObject):
    def lookup_key(keycode, group, level)
    def translate_keyboard_state(keycode, state, group)
    def get_entries_for_keyval(keyval)
    def get_entries_for_keycode(hardware_keycode)
    def get_direction()
Functions

    def gtk.gdk.keymap_get_default()
    def gtk.gdk.keymap_get_for_display(display)
    def gtk.gdk.keyval_name(keyval)
    def gtk.gdk.keyval_from_name(keyval_name)
    def gtk.gdk.keyval_convert_case(symbol)
    def gtk.gdk.keyval_to_upper(keyval)
    def gtk.gdk.keyval_to_lower(keyval)
    def gtk.gdk.keyval_is_upper(keyval)
    def gtk.gdk.keyval_is_lower(keyval)
    def gtk.gdk.keyval_to_unicode(keyval)
    def gtk.gdk.unicode_to_keyval(wc)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.gdk.Keymap
```

Signal Prototypes

"direction−changed"	`def callback(gdkkeymap, user_param1, ...)`
"keys−changed"	`def callback(gdkkeymap, user_param1, ...)`

Description

A `gtk.gdk.Keymap` defines the translation from keyboard state (including a hardware key, a modifier mask, and active keyboard group) to a keyval. This translation has two phases. The first phase is to determine the effective keyboard group and level for the keyboard state; the second phase is to look up the keycode/group/level triplet in the keymap and see what keyval it corresponds to. One `gtk.gdk.Keymap` object exists for each user display. `PyGTK` supports only one display, so `gtk.gdk.keymap_get_default()`() returns the singleton `gtk.gdk.Keymap`.

A keymap is a mapping from a Keymap key to key values. You can think of a Keymap key as a representation of a symbol printed on a physical keyboard key. That is, it contains three pieces of information. First, it contains the hardware keycode; this is an identifying number for a physical key. Second, it contains the level of the key. The level indicates which symbol on the key will be used, in a vertical direction. So on a standard US keyboard, the key with the number "1" on it also has the exclamation point ("!") character on it. The level indicates whether to use the "1" or the "!" symbol. The letter keys are considered to have a lowercase letter at level 0, and an uppercase letter at level 1, though only the uppercase letter is printed. Third, the Keymap key contains a group; groups are not used on standard US keyboards, but are used in many other countries. On a keyboard with groups, there can be 3 or 4 symbols printed on a single key. The group indicates movement in a horizontal direction. Usually groups are used for two different languages. In group 0, a key might have two English characters, and in group 1 it might have two Hebrew characters. The Hebrew characters will be printed on the key next to the English characters.

Methods

gtk.gdk.Keymap.lookup_key

```
    def lookup_key(keycode, group, level)
```

keycode :	the hardware keycode.
group :	the key group
level :	the key level
Returns :	a keyval, or 0 if none was mapped to the (*keycode*, *group*, *level*) triplet.

Note

This method is available in PyGTK 2.4 and above.

The `lookup_key()` method returns the keyval mapped to the specified (*keycode*, *group*, *level*) triplet. This method returns 0 if no keyval is found. For normal user input, you want to use the `translate_keyboard_state()` method instead of this method, since the effective group or level may not be the same as the current keyboard state.

The parameters to this method are:

keycode :	the hardware keycode. This is an identifying number for a physical key.
group :	indicates movement in a horizontal direction. Usually groups are used for two different languages. In group 0, a key might have two English characters, and in group 1 it might have two Hebrew characters. The Hebrew characters will be printed on the key next to the English characters.
level :	indicates which symbol on the key will be used, in a vertical direction. So on a standard US keyboard, the key with the number "1" on it also has the exclamation point ("!") character on it. The level indicates whether to use the "1" or the "!" symbol. The letter keys are considered to have a lowercase letter at level 0, and an uppercase letter at level 1, though only the uppercase letter is printed.

gtk.gdk.Keymap.translate_keyboard_state

```
    def translate_keyboard_state(keycode, state, group)
```

keycode :	a keycode
state :	a modifier state
group :	an active keyboard group
Returns :	a 4–tuple containing the keyval, the effective group, the level and the modifiers that were used to determine the group or level

Note

This method is available in PyGTK 2.4 and above.

The `translate_keyboard_state()` method translates the contents of a keyboard `gtk.gdk.Event` (specified by *keycode*, *state* and *group*) into a keyval, effective group, level and consumed modifiers that affected the translation (and are unavailable for application use) which are returned in a 4–tuple. See the `lookup_key()` method for an explanation of groups and levels. The effective group is the group that was actually used for the translation; some keys such as **Enter** are not affected by the active keyboard group. The level is derived from *state*. For convenience, the keyboard `gtk.gdk.Event` already contains the translated keyval, so this method isn't as useful as you might think.

The value of *state* or the consumed modifiers is a combination of:

`gtk.gdk.SHIFT_MASK`	The Shift key.
`gtk.gdk.LOCK_MASK`	A Lock key (depending on the modifier mapping of the X server this may either be CapsLock or ShiftLock).
`gtk.gdk.CONTROL_MASK`	The Control key.
`gtk.gdk.MOD1_MASK`	The fourth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier, but normally it is the Alt key).
`gtk.gdk.MOD2_MASK`	The fifth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
`gtk.gdk.MOD3_MASK`	The sixth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
`gtk.gdk.MOD4_MASK`	The seventh modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
`gtk.gdk.MOD5_MASK`	The eighth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
`gtk.gdk.BUTTON1_MASK`	The first mouse button.
`gtk.gdk.BUTTON2_MASK`	The second mouse button.
`gtk.gdk.BUTTON3_MASK`	The third mouse button.
`gtk.gdk.BUTTON4_MASK`	The fourth mouse button.
`gtk.gdk.BUTTON5_MASK`	The fifth mouse button.
`gtk.gdk.RELEASE_MASK`	Differentiates between (keyval, modifiers) pairs from key press and release events.
`gtk.gdk.MODIFIER_MASK`	all of the above

Consumed Modifiers

The consumed_modifiers are modifiers that should be masked out from state when comparing this key press to a hot key. For instance, on a US keyboard, the plus symbol is shifted, so when comparing a key press to a **Control**−plus accelerator **Shift** should be masked out. For example:

```
# We want to ignore irrelevant modifiers like ScrollLock
ALL_ACCELS_MASK = (gtk.gdk.CONTROL_MASK | gtk.gdk.SHIFT_MASK
                   | gtk.gdk.MOD1_MASK)
keyval, egroup, level, consumed = keymap.translate_keyboard_state(
                 keymap, event.hardware_keycode, event.state, event.group)
if (keyval == ord('+') and
    (event.state & ~consumed & ALL_ACCELS_MASK) == gtk.gdk.CONTROL_MASK):
  # Control was pressed
```

All single modifier combinations that could affect the key for any combination of modifiers will be returned in consumed_modifiers. Multi−modifier combinations are returned only when actually found in *state*. When you store accelerators, you should always store them with consumed modifiers removed. Store <Control>plus, not <Control><Shift>plus,

gtk.gdk.Keymap.get_entries_for_keyval

```
  def get_entries_for_keyval(keyval)
```

`keyval` :	a keyval, such as `GDK_a`, `GDK_Up`, `GDK_Return`, etc.

Returns :	a tuple containing 3−tuple containing a keycode, a group and a level that will generate *keyval*.

Note

This method is available in PyGTK 2.4 and above.

The `get_entries_for_keyval`() method returns a tuple of (keycode, group, level) 3−tuples that will generate *keyval*. Groups and levels are two kinds of keyboard mode; in general, the level determines whether the top or bottom symbol on a key is used, and the group determines whether the left or right symbol is used. On US keyboards, the shift key changes the keyboard level, and there are no groups. A group switch key might convert a keyboard between Hebrew to English modes, for example, the `gtk.gdk.KEY_PRESS` and `gtk.gdk.KEY_RELEASE` `gtk.gdk.Event` objects contain a `group` attribute that indicates the active keyboard group. The level is computed from the modifier mask.

gtk.gdk.Keymap.get_entries_for_keycode

```
    def get_entries_for_keycode(hardware_keycode)
```

`hardware_keycode` :	a keycode
Returns :	a tuple containing 4−tuples: (keyval, keycode, group, level)

Note

This method is available in PyGTK 2.4 and above.

The `get_entries_for_keycode`() method returns a tuple containing 4−tuples with: the keyvals bound to *hardware_keycode*, the keycode, the group and the level. When a keycode is pressed by the user, the keyval from this list of entries is selected by considering the effective keyboard group and level. See the `translate_keyboard_state()` method for more information.

gtk.gdk.Keymap.get_direction

```
    def get_direction()
```

Returns :	a Pango direction: `pango.DIRECTION_LTR` or `pango.DIRECTION_RTL`.

The `get_direction`() method returns the direction of the keymap.

Functions

gtk.gdk.keymap_get_default

```
    def gtk.gdk.keymap_get_default()
```

Returns :	the default gdk keymap for the display.

The `gtk.gdk.keymap_get_default`() function returns the default `gtk.gdk.Keymap` for the display.

gtk.gdk.keymap_get_for_display

```
    def gtk.gdk.keymap_get_for_display(display)
```

`display` :	a `gtk.gdk.Display`
Returns :	the keymap for *display*.

Note

This function is available in PyGTK 2.2 and above.

The `gtk.gdk.keymap_get_for_display()` function returns the `gtk.gdk.Keymap` for the `gtk.gdk.Display` specified by *`display`*.

gtk.gdk.keyval_name

```
def gtk.gdk.keyval_name(keyval)
```

`keyval` :	a key value
Returns :	a string containing the name of the key, or None if keyval is not a valid key.

The `gtk.gdk.keyval_name()` function converts the key value specified by *`keyval`* into a symbolic name.

gtk.gdk.keyval_from_name

```
def gtk.gdk.keyval_from_name(keyval_name)
```

`keyval_name` :	a key name
Returns :	the corresponding key value or 0 if the key name is not a valid key.

The `gtk.gdk.keyval_from_name()` function converts the key name specified by *`keyval_name`* to a key value.

gtk.gdk.keyval_convert_case

```
def gtk.gdk.keyval_convert_case(symbol)
```

`symbol` :	a keyval
Returns :	a 2–tuple containing the lowercase and uppercase versions of *`symbol`*

Note

This function is available in PyGTK 2.4 and above.

The `gtk.gdk.keyval_convert_case()` function returns the lowercase and uppercase versions of the keyval specified by *`symbol`*.

gtk.gdk.keyval_to_upper

```
def gtk.gdk.keyval_to_upper(keyval)
```

`keyval` :	a key value.
Returns :	the upper case form of keyval, or keyval itself if it is already in upper case or it is not subject to case conversion.

The `gtk.gdk.keyval_to_upper()` function converts the key value specified by *`keyval`* to upper case, if applicable.

gtk.gdk.keyval_to_lower

```
    def gtk.gdk.keyval_to_lower(keyval)
```

`keyval` :	the key value
Returns :	the lower case form of keyval, or keyval itself if it is already in lower case or it is not subject to case conversion.

The `gtk.gdk.keyval_to_lower()` function converts the key value specified by *`keyval`* to lower case, if applicable.

gtk.gdk.keyval_is_upper

```
    def gtk.gdk.keyval_is_upper(keyval)
```

`keyval` :	the key value
Returns :	`TRUE` if *`keyval`* is in upper case or if keyval is not subject to case conversion.

The `gtk.gdk.keyval_is_upper()` function returns `TRUE` if the key value specified by *`keyval`* is in upper case or not subject to case conversion.

gtk.gdk.keyval_is_lower

```
    def gtk.gdk.keyval_is_lower(keyval)
```

`keyval` :	the key value
Returns :	`TRUE` if *`keyval`* is in lower case, or if *`keyval`* is not subject to case conversion.

The `gtk.gdk.keyval_is_lower()` function returns `TRUE` if the key value specified by *`keyval`* is in lower case or is not subject to case conversion.

gtk.gdk.keyval_to_unicode

```
    def gtk.gdk.keyval_to_unicode(keyval)
```

`keyval` :	the key value
Returns :	the corresponding unicode character, or 0 if there is no corresponding character.

The `gtk.gdk.keyval_to_unicode()` function converts the key value specified by *`keyval`* to the corresponding ISO10646 (Unicode) character.

gtk.gdk.unicode_to_keyval

```
    def gtk.gdk.unicode_to_keyval(wc)
```

`wc` :	a ISO10646 encoded (unicode) character
Returns :	the corresponding key value, if one exists. or, if there is no corresponding symbol, *`wc`* | 0x01000000

The `gtk.gdk.unicode_to_keyval()` function converts the ISO10646 (unicode) character specified by *`wc`* to a key value.

Signals

The "direction−changed" gtk.gdk.Keymap Signal

```
    def callback(gdkkeymap, user_param1, ...)
```

gdkkeymap :	the gdkkeymap that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "direction−changed" signal is emitted when the pango text direction of *gdkkeymap* is changed

The "keys−changed" gtk.gdk.Keymap Signal

```
    def callback(gdkkeymap, user_param1, ...)
```

gdkkeymap :	the gdkkeymap that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.2 and above.

The "keys−changed" signal is emitted when the mapping represented by *keymap* changes.

gtk.gdk.Pixbuf

gtk.gdk.Pixbuf — an object containing a client side image.

Synopsis

```
class gtk.gdk.Pixbuf(gobject.GObject):
    gtk.gdk.Pixbuf(colorspace, has_alpha, bits_per_sample, width, height)
    def render_to_drawable(drawable, gc, src_x, src_y, dest_x, dest_y, width, height, dither, x
    def render_to_drawable_alpha(drawable, src_x, src_y, dest_x, dest_y, width, height, alpha_m
    def render_pixmap_and_mask(alpha_threshold=127)
    def get_from_drawable(src, cmap, src_x, src_y, dest_x, dest_y, width, height)
    def get_from_image(src, cmap, src_x, src_y, dest_x, dest_y, width, height)
    def get_colorspace()
    def get_n_channels()
    def get_has_alpha()
    def get_bits_per_sample()
    def get_pixels()
    def get_width()
    def get_height()
    def get_rowstride()
    def get_option(key)
    def copy()
    def fill(pixel)
    def save(filename, type, options=None)
    def add_alpha(substitute_color, r, g, b)
    def copy_area(src_x, src_y, width, height, dest_pixbuf, dest_x, dest_y)
    def saturate_and_pixelate(dest, saturation, pixelate)
    def scale(dest, dest_x, dest_y, dest_width, dest_height, offset_x, offset_y, scale_x, scale
    def composite(dest, dest_x, dest_y, dest_width, dest_height, offset_x, offset_y, scale_x, s
    def composite_color(dest, dest_x, dest_y, dest_width, dest_height, offset_x, offset_y, scal
    def scale_simple(dest_width, dest_height, interp_type)
    def composite_color_simple(dest_width, dest_height, interp_type, overall_alpha, check_size,
    def get_pixels_array()
    def subpixbuf(src_x, src_y, width, height)
Functions

    def gtk.gdk.pixbuf_new_from_file(filename)
    def gtk.gdk.pixbuf_new_from_file_at_size(filename, width, height)
    def gtk.gdk.pixbuf_new_from_data(data, colorspace, has_alpha, bits_per_sample, width, heigh
    def gtk.gdk.pixbuf_new_from_array(array, colorspace, bits_per_sample)
    def gtk.gdk.pixbuf_new_from_xpm_data(data)
    def gtk.gdk.pixbuf_new_from_inline(data_length, data, copy_pixels)
    def gtk.gdk.pixbuf_get_formats()
    def gtk.gdk.pixbuf_get_file_info(filename)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.gdk.Pixbuf
```

Attributes

"pixel_array"	Read	A numeric array containing the pixel values of the `gtk.gdk.Pixbuf`. The contents of the numeric array can be changed even though the array is read−only. However you cannot change the pixel data of pixbufs that are backed by a const string such as stock icon pixbufs. `PyGTK` must be compiled with Numeric Python support for this to work.

Properties

"bits−per−sample"	Read−Write−Construct Only	The number of bits per sample. Available in GTK+ 2.4 and above.
"colorspace"	Read−Write−Construct Only	The colorspace in which the samples are interpreted.Available in GTK+ 2.4 and above.
"has−alpha"	Read−Write−Construct Only	If `TRUE`,the pixbuf has an alpha channel. Available in GTK+ 2.4 and above.
"height"	Read−Write−Construct Only	The number of rows of the pixbuf. Available in GTK+ 2.4 and above.
"n−channels"	Read−Write−Construct Only	The number of samples per pixel. Available in GTK+ 2.4 and above.
"pixels"	Read−Write−Construct Only	A pointer to the pixel data of the pixbuf. Available in GTK+ 2.4 and above.
"rowstride"	Read−Write−Construct Only	The number of bytes between the start of a row and the start of the next row. Available in GTK+ 2.4 and above.
"width"	Read−Write−Construct Only	The number of columns of the pixbuf. Available in GTK+ 2.4 and above.

Description

A `gtk.gdk.Pixbuf` object contains the data that describes an image using client side resources. By contrast a `gtk.gdk.Pixmap` uses server side resources to hold image data. Manipulating the image data in a `gtk.gdk.Pixmap` may involve round trip transfers between a client and a server in X11 while manipulating image data in a `gtk.gdk.Pixbuf` involves only client side operations. Therefore using `gtk.gdk.Pixbuf` objects may be more efficient than using `gtk.gdk.Pixmap` objects if a lot of image manipulation is necessary.

In addition to the methods associated with a `gtk.gdk.Pixbuf` object there are a number of functions that can be used to create `gtk.gdk.Pixbuf` objects from file and inline data.

Constructor

```
    gtk.gdk.Pixbuf(colorspace, has_alpha, bits_per_sample, width, height)
```

colorspace :	the color space for the image – currently only `gtk.gdk.COLORSPACE_RGB`.
has_alpha :	if `TRUE` the image will have transparency information.
bits_per_sample :	the number of bits per color sample – currently must be 8.
width :	the width of image in pixels.
height :	the height of image in pixels.
Returns :	a newly–created `gtk.gdk.Pixbuf`.

Creates a new `gtk.gdk.Pixbuf` structure and allocates a buffer for it. The buffer has an optimal rowstride. Note that the buffer is not cleared; you will have to fill it completely yourself. The size of the image is specified by *`width`* and *`height`*.

Methods

gtk.gdk.Pixbuf.render_to_drawable

```
    def render_to_drawable(drawable, gc, src_x, src_y, dest_x, dest_y, width, height, dither, x_
```

drawable :	the destination `gtk.gdk.Drawable`.
gc :	the `gtk.gdk.GC` used for rendering.
src_x :	the X coordinate within the pixbuf.
src_y :	the Y coordinate within the pixbuf.
dest_x :	the X coordinate within *`drawable`*.
dest_y :	the Y coordinate within *`drawable`*.
width :	the width of region to render, in pixels, or –1 to use pixbuf width
height :	the height of region to render, in pixels, or –1 to use pixbuf height
dither :	the dithering mode.
x_dither :	the X offset for dithering.
y_dither :	the Y offset for dithering.

Warning

This method is deprecated in PyGTK 2.2 and above. Use the `gtk.gdk.Drawable.draw_pixbuf()` method instead.

The `render_to_drawable()` method renders a rectangular portion of the pixbuf to a `gtk.gdk.Drawable` specified by *`drawable`* while using the `gtk.gdk.GC` specified by *`gc`*. This is done using GdkRGB, so the specified drawable must have the GdkRGB visual and colormap. Note that this method will ignore the opacity information for images with an alpha channel; the `gtk.gdk.GC` must already have the clipping mask set if you want transparent regions to show through.

The dither offset is important when re–rendering partial regions of an image to a rendered version of the full image, or when the offsets to a base position change, as in scrolling. The dither matrix has to be shifted for consistent visual results. If you do not have any of these cases, the dither offsets can be both zero.

gtk.gdk.Pixbuf.render_to_drawable_alpha

```
    def render_to_drawable_alpha(drawable, src_x, src_y, dest_x, dest_y, width, height, alpha_mo
```

`drawable` :	the destination `gtk.gdk.Drawable`.
`gc` :	the `gtk.gdk.GC` used for rendering.
`src_x` :	the X coordinate within the pixbuf.
`src_y` :	the Y coordinate within the pixbuf.
`dest_x` :	the X coordinate within *`drawable`*.
`dest_y` :	the Y coordinate within *`drawable`*.
`width` :	the width of region to render, in pixels, or −1 to use pixbuf width
`height` :	the height of region to render, in pixels, or −1 to use pixbuf height
`alpha_mode` :	Ignored. Present for backward compatibility.
`alpha_threshold` :	Ignored. Present for backward compatibility
`dither` :	the dithering mode.
`x_dither` :	the X offset for dithering.
`y_dither` :	the Y offset for dithering.

Warning

This method is deprecated in PyGTK 2.2 and above. Use the `gtk.gdk.Drawable.draw_pixbuf()` method instead.

The `render_to_drawable_alpha()` method renders a rectangular portion of the pixbuf to a `gtk.gdk.Drawable` specified by *`drawable`*. *`drawable`* must have a colormap. All windows have a colormap; however, pixmaps only have colormap by default if they were created with a window argument.specifying a `gtk.gdk.Window`. Otherwise a colormap must be set on them with the `gtk.gdk.Drawable.set_colormap()` method. On older X servers, rendering pixbufs with an alpha channel involves round trips to the X server, and may be somewhat slow.

gtk.gdk.Pixbuf.render_pixmap_and_mask

```
    def render_pixmap_and_mask(alpha_threshold=127)
```

`alpha_threshold` :	the threshold value for opacity values.
Returns :	a tuple containing the created pixmap and mask.

The `render_to_pixmap_and_mask()` method returns a tuple containing a pixmap and a mask bitmap that the pixbuf and its corresponding thresholded alpha mask are rendered to. This is merely a convenience method; applications that need to render pixbufs with dither offsets or to specific drawables should use the `render_to_drawable_alpha()` or `render_to_drawable()` methods. If the pixbuf does not have an alpha channel, then the mask returned will be None.

gtk.gdk.Pixbuf.get_from_drawable

```
    def get_from_drawable(src, cmap, src_x, src_y, dest_x, dest_y, width, height)
```

src :	the source gtk.gdk.Drawable.
cmap :	a colormap if *src* doesn't have one set.
src_x :	the X coordinate within *drawable*.
src_y :	the Y coordinate within *drawable*.
dest_x :	the X coordinate in the pixbuf.
dest_y :	the Y coordinate in the pixbuf.
width :	the width in pixels of the region to get.
height :	the height in pixels of the region to get.
Returns :	the pixbuf or None on error

The get_from_drawable() method transfers image data from the gtk.gdk.Drawable specified by *src* and converts it to an RGB(A) representation inside a gtk.gdk.Pixbuf. In other words, copies image data from a server−side drawable to a client−side RGB(A) buffer. This allows you to efficiently read individual pixels on the client side. If *src* has no colormap (the gtk.gdk.Drawable.get_colormap() method returns None), then a suitable colormap must be specified as *cmap*. Typically a gtk.gdk.Window or a pixmap created by passing a gtk.gdk.Window to gtk.gdk.Pixmap() will already have a colormap associated with it. If *src* has a colormap, the *cmap* argument will be ignored. If *src* is a bitmap (1 bit per pixel pixmap), then a colormap is not required; pixels with a value of 1 are assumed to be white, and pixels with a value of 0 are assumed to be black. For taking screenshots, the gtk.gdk.colormap_get_system() function returns the correct colormap to use.

If *src* is a pixmap, then the requested source rectangle must be completely contained within the pixmap, otherwise the function will return None. For pixmaps only (not for windows) passing −1 for *width* or *height* is allowed to mean the full width or height of the pixmap. If *src* is a window, and the window is off the screen, then there is no image data in the obscured/offscreen regions to be placed in the pixbuf. The contents of portions of the pixbuf corresponding to the offscreen region are undefined.

If the window you're obtaining data from is partially obscured by other windows, then the contents of the pixbuf areas corresponding to the obscured regions are undefined. If the target drawable is not mapped (typically because it's iconified/minimized or not on the current workspace), None will be returned. If memory can't be allocated for the return value, None will be returned instead. (In short, there are several ways this method can fail, and if it fails it returns None; so check the return value.)

This method calls the gtk.gdk.Drawable.get_image() method internally and converts the resulting image to a gtk.gdk.Pixbuf, so the documentation for the gtk.gdk.Drawable.get_image() method may also be helpful.

gtk.gdk.Pixbuf.get_from_image

```
    def get_from_image(src, cmap, src_x, src_y, dest_x, dest_y, width, height)
```

src :	the source gtk.gdk.Image.
cmap :	a colormap if *src* doesn't have one set or None.
src_x :	the X coordinate within *src*.
src_y :	the Y coordinate within *src*.
dest_x :	the X coordinate in the pixbuf.
dest_y :	the Y coordinate in the pixbuf.
width :	the width in pixels of the region to get.

`height` :	the height in pixels of the region to get.
Returns :	the pixbuf or `None` on error

The `get_from_image()` method is the same as the `get_from_drawable()` method but gets the pixbuf from the `gtk.gdk.Image` specified by *`src`*.

gtk.gdk.Pixbuf.get_colorspace

```
def get_colorspace()
```

Returns :	the color space.

The `get_colorspace()` method returns the color space of the pixbuf.

gtk.gdk.Pixbuf.get_n_channels

```
def get_n_channels()
```

Returns :	the number of channels.

The `get_n_channels()` method returns the number of channels of a pixbuf.

gtk.gdk.Pixbuf.get_has_alpha

```
def get_has_alpha()
```

Returns :	`TRUE` if the pixbuf has an alpha channel.

The `get_has_alpha()` method returns `TRUE` if the pixbuf has an alpha channel (opacity information).

gtk.gdk.Pixbuf.get_bits_per_sample

```
def get_bits_per_sample()
```

Returns :	the number of bits per color sample.

The `get_bits_per_sample()` method returns the number of bits per color sample in a pixbuf.

gtk.gdk.Pixbuf.get_pixels

```
def get_pixels()
```

Returns :	a string containing the pixel data of the pixbuf

The `get_pixels()` method returns a sting containing the pixel data of the pixbuf.

gtk.gdk.Pixbuf.get_width

```
def get_width()
```

Returns :	the width in pixels.

The `get_width()` method returns the width of the pixbuf.

gtk.gdk.Pixbuf.get_height

```
def get_height()
```

Returns :	the height in pixels.

The `get_height()` method returns the height of the pixbuf.

gtk.gdk.Pixbuf.get_rowstride

```
def get_rowstride()
```

Returns :	the number of bytes between rows.

The `get_rowstride()` method returns the rowstride of a pixbuf, which is the number of bytes between rows.

gtk.gdk.Pixbuf.get_option

```
def get_option(key)
```

`key` :	a key string
Returns :	the value associated with *`key`*

The `get_option()` method looks up *`key`* in the list of options that may have been attached to the pixbuf when it was loaded.

gtk.gdk.Pixbuf.copy

```
def copy()
```

Returns :	A newly–created pixbuf or `None` if not enough memory could be allocated.

The `copy()` method returns a new `gtk.gdk.Pixbuf` with a copy of the data in the pixbuf.

gtk.gdk.Pixbuf.fill

```
def fill(pixel)
```

`pixel` :	the RGBA pixel to clear to (0xffffffff is opaque white, 0x00000000 transparent black)

The `fill()` method clears the pixbuf to the RGBA value specified by *`pixel`*, converting the RGBA value into the pixbuf's pixel format. The alpha will be ignored if the pixbuf doesn't have an alpha channel.

gtk.gdk.Pixbuf.save

```
def save(filename, type, options=None)
```

`filename` :	the name of file to save.
`type` :	the name of the file format.
`options` :	a Python dict containing key–value string pairs or `None`

The `save()` method saves the pixbuf to a file in the format specified by *`type`*, which may be "jpeg" or "png". If *`options`* is not `None` it should be a Python dict containing key–value string pairs that modify the save parameters. For example:

```
pixbuf.save(filename, "jpeg", {"quality":"100"})
```

Currently only a few parameters exist. JPEG images can be saved with a "quality" parameter; its value should be in the range [0,100]. Text chunks can be attached to PNG images by specifying parameters of the form "tEXt::key", where key is an ASCII string of length 1−79. The values are UTF−8 encoded strings. Note however that PNG text chunks are stored in ISO−8859−1 encoding, so you can only set texts that can be represented in this encoding.

This method raises the GError exception if an error occurs during the save operation.

gtk.gdk.Pixbuf.add_alpha

```
def add_alpha(substitute_color, r, g, b)
```

substitute_color :	if FALSE, the (*r*, *g*, *b*) arguments will be ignored.
r :	the red value to substitute.
g :	the green value to substitute.
b :	the blue value to substitute.
Returns :	a new gtk.gdk.Pixbuf.

The add_alpha() method returns a new gtk.gdk.Pixbuf created from the pixbuf with an alpha channel added. If the pixbuf already had an alpha channel, the channel values are copied; otherwise, the alpha channel is initialized to 255 (full opacity).

If *substitute_color* is TRUE, then the color specified by (*r*, *g*, *b*) will be assigned zero opacity. That is, if you pass (255, 255, 255) for the substitute color, all white pixels will become fully transparent.

gtk.gdk.Pixbuf.copy_area

```
def copy_area(src_x, src_y, width, height, dest_pixbuf, dest_x, dest_y)
```

src_x :	the X coordinate within the pixbuf.
src_y :	the Y coordinate within the pixbuf.
width :	the width of the area to copy.
height :	the height of the area to copy.
dest_pixbuf :	the destination pixbuf.
dest_x :	the X coordinate within *dest_pixbuf*.
dest_y :	the Y coordinate within *dest_pixbuf*.

The copy_area() method copies a rectangular area from the pixbuf to the destination gtk.gdk.Pixbuf specified by *dest_pixbuf*. Conversion of pixbuf formats is done automatically.

gtk.gdk.Pixbuf.saturate_and_pixelate

```
def saturate_and_pixelate(dest, saturation, pixelate)
```

dest :	the gtk.gdk.Pixbuf to write the modified version of the pixbuf
saturation :	the saturation factor
pixelate :	if TRUE pixelate *dest*

The saturate_and_pixelate() method modifies the saturation and optionally pixelates the pixbuf, placing the result in *dest*. the pixbuf and *dest* may be the same gtk.gdk.Pixbuf with no ill effects. If *saturation* is 1.0 then saturation is not changed. If it's less than 1.0, saturation is reduced (the image is darkened); if greater than 1.0, saturation is increased (the image is brightened). If *pixelate* is TRUE, then pixels are faded in a checkerboard pattern to create a pixelated image. The pixbuf and *dest* must have the same image format, size, and rowstride.

gtk.gdk.Pixbuf.scale

```
    def scale(dest, dest_x, dest_y, dest_width, dest_height, offset_x, offset_y, scale_x, scale_
```

dest :	the gtk.gdk.Pixbuf the scaling is rendered to.
dest_x :	the X coordinate for the rectangle
dest_y :	the Y coordinate for the rectangle
dest_width :	the width of the rectangle
dest_height :	the height of the rectangle
offset_x :	the offset in the X direction (currently rounded to an integer)
offset_y :	the offset in the Y direction (currently rounded to an integer)
scale_x :	the scale factor in the X direction
scale_y :	the scale factor in the Y direction
interp_type :	the interpolation type for the transformation.

The `scale()` method creates a transformation of the pixbuf's image by scaling by *scale_x* and *scale_y* and translating by *offset_x* and *offset_y* it, then rendering the rectangle (*dest_x*, *dest_y*, *dest_width*, *dest_height*) of the resulting image onto the destination image specified by *dest* replacing the previous contents. The value of *interp_type* must be one of:

`gtk.gdk.INTERP_NEAREST`	Nearest neighbor sampling; this is the fastest and lowest quality mode. Quality is normally unacceptable when scaling down, but may be OK when scaling up.
`gtk.gdk.INTERP_TILES`	This is an accurate simulation of the PostScript image operator without any interpolation enabled. Each pixel is rendered as a tiny parallelogram of solid color, the edges of which are implemented with antialiasing. It resembles nearest neighbor for enlargement, and bilinear for reduction.
`gtk.gdk.INTERP_BILINEAR`	Best quality/speed balance; use this mode by default. Bilinear interpolation. For enlargement, it is equivalent to point−sampling the ideal bilinear−interpolated image. For reduction, it is equivalent to laying down small tiles and integrating over the coverage area.
`gtk.gdk.INTERP_HYPER`	This is the slowest and highest quality reconstruction function. It is derived from the hyperbolic filters in Wolberg's "Digital Image Warping", and is formally defined as the hyperbolic−filter sampling the ideal hyperbolic−filter interpolated image (the filter is designed to be idempotent for 1:1 pixel mapping).

Try the `scale_simple()` method as an alternative with a simpler interface.

gtk.gdk.Pixbuf.composite

```
    def composite(dest, dest_x, dest_y, dest_width, dest_height, offset_x, offset_y, scale_x, sc
```

dest :	the output gtk.gdk.Pixbuf
dest_x :	the X coordinate for the rectangle
dest_y :	the top coordinate for the rectangle
dest_width :	the width of the rectangle
dest_height :	the height of the rectangle
offset_x :	the offset in the X direction (currently rounded to an integer)

offset_y :	the offset in the Y direction (currently rounded to an integer)
scale_x :	the scale factor in the X direction
scale_y :	the scale factor in the Y direction
interp_type :	the interpolation type for the transformation.
overall_alpha :	overall alpha for source image (0..255)

The `composite`() method creates a transformation of the pixbuf's image by scaling by *`scale_x`* and *`scale_y`* and translating by *`offset_x`* and *`offset_y`*, then compositing the rectangle (*`dest_x`*, *`dest_y`*, *`dest_width`*, *`dest_height`*) of the resulting image onto the destination image. The value of *`interp_type`* must be one of:

`gtk.gdk.INTERP_NEAREST`	Nearest neighbor sampling; this is the fastest and lowest quality mode. Quality is normally unacceptable when scaling down, but may be OK when scaling up.
`gtk.gdk.INTERP_TILES`	This is an accurate simulation of the PostScript image operator without any interpolation enabled. Each pixel is rendered as a tiny parallelogram of solid color, the edges of which are implemented with antialiasing. It resembles nearest neighbor for enlargement, and bilinear for reduction.
`gtk.gdk.INTERP_BILINEAR`	Best quality/speed balance; use this mode by default. Bilinear interpolation. For enlargement, it is equivalent to point−sampling the ideal bilinear−interpolated image. For reduction, it is equivalent to laying down small tiles and integrating over the coverage area.
`gtk.gdk.INTERP_HYPER`	This is the slowest and highest quality reconstruction function. It is derived from the hyperbolic filters in Wolberg's "Digital Image Warping", and is formally defined as the hyperbolic−filter sampling the ideal hyperbolic−filter interpolated image (the filter is designed to be idempotent for 1:1 pixel mapping).

gtk.gdk.Pixbuf.composite_color

```
    def composite_color(dest, dest_x, dest_y, dest_width, dest_height, offset_x, offset_y, scale
```

dest :	the output `gtk.gdk.Pixbuf`
dest_x :	the X coordinate for the rectangle
dest_y :	the top coordinate for the rectangle
dest_width :	the width of the rectangle
dest_height :	the height of the rectangle
offset_x :	the offset in the X direction (currently rounded to an integer)
offset_y :	the offset in the Y direction (currently rounded to an integer)
scale_x :	the scale factor in the X direction
scale_y :	the scale factor in the Y direction
interp_type :	the interpolation type for the transformation.
overall_alpha :	overall alpha for source image (0..255)
check_x :	the X offset for the checkboard (origin of checkboard is at −*`check_x`*, −*`check_y`*)
check_y :	the Y offset for the checkboard
check_size :	the size of checks in the checkboard (must be a power of two)
color1 :	the color of check at upper left
color2 :	the color of the other check

The `composite_color`() method creates a transformation of the source image *`src`* by scaling by *`scale_x`* and *`scale_y`* and translating by *`offset_x`* and *`offset_y`*, then compositing the rectangle (*dest_x* ,*dest_y*, *dest_width*, *dest_height*) of the resulting image with a checkboard of the colors *color1* and *color2* and renders it onto the destination image. The value of *interp_type* must be one of:

`gtk.gdk.INTERP_NEAREST`	Nearest neighbor sampling; this is the fastest and lowest quality mode. Quality is normally unacceptable when scaling down, but may be OK when scaling up.
`gtk.gdk.INTERP_TILES`	This is an accurate simulation of the PostScript image operator without any interpolation enabled. Each pixel is rendered as a tiny parallelogram of solid color, the edges of which are implemented with antialiasing. It resembles nearest neighbor for enlargement, and bilinear for reduction.
`gtk.gdk.INTERP_BILINEAR`	Best quality/speed balance; use this mode by default. Bilinear interpolation. For enlargement, it is equivalent to point−sampling the ideal bilinear−interpolated image. For reduction, it is equivalent to laying down small tiles and integrating over the coverage area.
`gtk.gdk.INTERP_HYPER`	This is the slowest and highest quality reconstruction function. It is derived from the hyperbolic filters in Wolberg's "Digital Image Warping", and is formally defined as the hyperbolic−filter sampling the ideal hyperbolic−filter interpolated image (the filter is designed to be idempotent for 1:1 pixel mapping).

See the `composite_color_simple()` method for a simpler variant of this method suitable for most tasks.

gtk.gdk.Pixbuf.scale_simple

```
    def scale_simple(dest_width, dest_height, interp_type)
```

`dest_width` :	the width of destination image
`dest_height` :	the height of destination image
`interp_type` :	the interpolation type for the transformation.
Returns :	the new `gtk.gdk.Pixbuf`, or `None` if not enough memory could be allocated for it.

The `scale_simple`() method returns a new `gtk.gdk.Pixbuf` containing a copy of the pixbuf scaled to *dest_width* x *dest_height*. The pixbuf is unaffected by the scaling operation. The value of *interp_type* must be one of:

`gtk.gdk.INTERP_NEAREST`	Nearest neighbor sampling; this is the fastest and lowest quality mode. Quality is normally unacceptable when scaling down, but may be OK when scaling up.
`gtk.gdk.INTERP_TILES`	This is an accurate simulation of the PostScript image operator without any interpolation enabled. Each pixel is rendered as a tiny parallelogram of solid color, the edges of which are implemented with antialiasing. It resembles nearest neighbor for enlargement, and bilinear for reduction.
`gtk.gdk.INTERP_BILINEAR`	Best quality/speed balance; use this mode by default. Bilinear interpolation. For enlargement, it is equivalent to point−sampling the ideal bilinear−interpolated image. For reduction, it is equivalent to laying down small tiles and integrating over the coverage area.
`gtk.gdk.INTERP_HYPER`	This is the slowest and highest quality reconstruction function. It is derived from the hyperbolic filters in Wolberg's "Digital Image Warping", and is formally defined as the hyperbolic−filter sampling the ideal hyperbolic−filter interpolated image (the filter is designed to be idempotent for 1:1 pixel mapping).

interp_type should be `gtk.gdk.INTERP_NEAREST` if you want maximum speed (but when scaling down `gtk.gdk.INTERP_NEAREST` is usually unusably ugly). The default *interp_type* should be

GDK_INTERP_BILINEAR which offers reasonable quality and speed.

You can scale a sub−portion of *src* by creating a sub−pixbuf pointing into *src*; see the subpixbuf() method for more information.

For more complicated scaling/compositing see the scale() and composite() methods.

gtk.gdk.Pixbuf.composite_color_simple

```
    def composite_color_simple(dest_width, dest_height, interp_type, overall_alpha, check_size,
```

dest_width :	the width of destination image
dest_height :	the height of destination image
interp_type :	the interpolation type for the transformation.
overall_alpha :	overall alpha for source image (0..255)
check_size :	the size of checks in the checkboard (must be a power of two)
color1 :	the color of check at upper left
color2 :	the color of the other check
Returns :	the new gtk.gdk.Pixbuf, or NULL if not enough memory could be allocated for it.

The composite_color_simple() method returns a new gtk.gdk.Pixbuf by scaling the pixbuf to *dest_width* x *dest_height* and compositing the result with a checkboard of colors *color1* and *color2*. The value of *interp_type* must be one of:

gtk.gdk.INTERP_NEAREST	Nearest neighbor sampling; this is the fastest and lowest quality mode. Quality is normally unacceptable when scaling down, but may be OK when scaling up.
gtk.gdk.INTERP_TILES	This is an accurate simulation of the PostScript image operator without any interpolation enabled. Each pixel is rendered as a tiny parallelogram of solid color, the edges of which are implemented with antialiasing. It resembles nearest neighbor for enlargement, and bilinear for reduction.
gtk.gdk.INTERP_BILINEAR	Best quality/speed balance; use this mode by default. Bilinear interpolation. For enlargement, it is equivalent to point−sampling the ideal bilinear−interpolated image. For reduction, it is equivalent to laying down small tiles and integrating over the coverage area.
gtk.gdk.INTERP_HYPER	This is the slowest and highest quality reconstruction function. It is derived from the hyperbolic filters in Wolberg's "Digital Image Warping", and is formally defined as the hyperbolic−filter sampling the ideal hyperbolic−filter interpolated image (the filter is designed to be idempotent for 1:1 pixel mapping).

See the composite_color() method for a more powerful but complicated interface.

gtk.gdk.Pixbuf.get_pixels_array

```
    def get_pixels_array()
```

Returns :	a Numeric Python array containing the pixel data of the pixbuf

Note

This method is available in PyGTK 2.2 and above.

The get_pixels_array() method returns a Numeric Python array containing the pixel data of the pixbuf.

Note

PyGTK must be compiled with Numeric Python support for this method to be available.

gtk.gdk.Pixbuf.subpixbuf

```
    def subpixbuf(src_x, src_y, width, height)
```

`src_x` :	X coord in the pixbuf
`src_y` :	Y coord in the pixbuf
`width` :	width of region in the pixbuf
`height` :	height of region in the pixbuf
Returns :	a new `gtk.gdk.Pixbuf`

Note

This method is available in PyGTK 2.4 and above.

The `subpixbuf`() method creates a new `gtk.gdk.Pixbuf` that represents a sub–region of the pixbuf. The new pixbuf shares its pixels with the original pixbuf, so writing to one affects both.

Functions

gtk.gdk.pixbuf_new_from_file

```
    def gtk.gdk.pixbuf_new_from_file(filename)
```

`filename` :	the name of the file containing the image to load
Returns :	a new `gtk.gdk.Pixbuf`.

The `gtk.gdk.pixbuf_new_from_file`() function returns a new `gtk.gdk.Pixbuf` containing an image loaded from the file specified by *`filename`*. The image file format is detected automatically. The application will block until the image is done loading. This method can be used by applications in which blocking is acceptable while an image is being loaded (small images in general). Applications that need progressive loading should use `gtk.gdk.PixbufLoader` instead.

This function raises the GError exception if an error occurs during the loading of the pixbuf.

gtk.gdk.pixbuf_new_from_file_at_size

```
    def gtk.gdk.pixbuf_new_from_file_at_size(filename, width, height)
```

`filename` :	the name of the file containing the image to load
`width` :	The width the image should have
`height` :	The height the image should have
Returns :	a new `gtk.gdk.Pixbuf`.

Note

This function is available in PyGTK 2.4 and above.

The `gtk.gdk.pixbuf_new_from_file_at_size`() function returns a new `gtk.gdk.Pixbuf` containing an image loaded from the file specified by *`filename`* with it scaled to the size specified by *`width`* and *`height`*. The image file format is detected automatically. The application will block until the image is done loading. This function can be used by applications in which blocking is acceptable while an image is being loaded (small images in general). Applications that need progressive loading should use a `gtk.gdk.PixbufLoader` instead.

This function raises the GError exception if an error occurs during the loading of the pixbuf.

gtk.gdk.pixbuf_new_from_data

```
    def gtk.gdk.pixbuf_new_from_data(data, colorspace, has_alpha, bits_per_sample, width, height
```

`data` :	a string containing image data in 8–bit/sample packed format.
`colorspace` :	the colorspace for the image data.
`has_alpha` :	If `TRUE`, the data has an opacity channel.
`bits_per_sample` :	the number of bits per sample.
`width` :	the width of the image in pixels.
`height` :	the height of the image in pixels.
`rowstride` :	the distance in bytes between row starts.
Returns :	a `gtk.gdk.Pixbuf`

Note

This function is available in PyGTK 2.2 and above.

The `gtk.gdk.pixbuf_new_from_data`() function returns a new pixbuf created from the string specified by *`data`*. *`data`* must be RGB image data with 8 bits per sample. *`colorspace`* must be `gtk.gdk.COLORSPACE_RGB`.

gtk.gdk.pixbuf_new_from_array

```
    def gtk.gdk.pixbuf_new_from_array(array, colorspace, bits_per_sample)
```

`array` :	a string containing image data in 8–bit/sample packed format.
`colorspace` :	the colorspace for the image data.
`bits_per_sample` :	the number of bits per sample.
Returns :	a `gtk.gdk.Pixbuf`

Note

This function is available in PyGTK 2.2 and above.

The `gtk.gdk.pixbuf_new_from_array`() function returns a new pixbuf created from the Numeric Python array specified by *`array`*. *`array`* must be a 3 or 4 dimensional array (4 if the image has an alpha channel) with *`bits_per_sample`* bits per sample. *`colorspace`* must be `gtk.gdk.COLORSPACE_RGB`.

Note

PyGTK must be compiled with the Numeric Python module to support this function.

gtk.gdk.pixbuf_new_from_xpm_data

```
def gtk.gdk.pixbuf_new_from_xpm_data(data)
```

data :	a list of strings containing the XPM image data
Returns :	a gtk.gdk.Pixbuf

The `gtk.gdk.pixbuf_new_from_xpm_data()` function returns a new gtk.gdk.Pixbuf by parsing `XPM` data in memory specified by *data*. *data* is a list of strings containing the `XPM` data.

gtk.gdk.pixbuf_new_from_inline

```
def gtk.gdk.pixbuf_new_from_inline(data_length, data, copy_pixels)
```

data_length :	the length in bytes of the *data*
data :	a string containing the inline pixbuf data
copy_pixels :	if `TRUE` the pixel data should be copied
Returns :	a new gtk.gdk.Pixbuf object

The `gtk.gdk.pixbuf_new_from_inline()` function returns a gtk.gdk.Pixbuf from a flat representation that is suitable for storing as inline data in a program. This is useful if you want to ship a program with images, but don't want to depend on any external files.

GTK+ ships with a program called **gdk−pixbuf−csource** which allows for conversion of an image into such a inline representation.In almost all cases, you should pass the `--raw` flag to **gdk−pixbuf−csource**. A sample invocation would be:

```
gdk-pixbuf-csource --raw --name=myimage_inline myimage.png
```

For the typical case where the inline pixbuf is read−only static data, you don't need to copy the pixel data unless you intend to write to it, so you can pass `FALSE` for *copy_pixels*.

This function raises the GError exception if an error occurs during the loading of the pixbuf.

gtk.gdk.pixbuf_get_formats

```
def gtk.gdk.pixbuf_get_formats()
```

Returns :	a list of image formats as Python dicts

The `gtk.gdk.pixbuf_get_formats()` function returns a list of the supported image formats as a Python dict. The keys of the image format dict are:

name :	the name of the image format.
description :	a description of the image format.
mime_types :	a list of the mime types this image matches.
extensions :	a list of typical filename extensions for the image format.
is_writable :	if `TRUE` the image can be written to a file

gtk.gdk.pixbuf_get_file_info

```
def gtk.gdk.pixbuf_get_file_info(filename)
```

filename :	the name of the file to check
Returns :	an image format as a Python dict

The `gtk.gdk.pixbuf_get_file_info()` function reads enough of the file specified by *filename* to determine its image format and then returns the image format information in a Python dict. See the gtk.gdk.pixbuf_get_formats() function for more details on the image format dict.

gtk.gdk.PixbufAnimation

gtk.gdk.PixbufAnimation — an object holding an animation

Synopsis

```
class gtk.gdk.PixbufAnimation(gobject.GObject):
    gtk.gdk.PixbufAnimation(filename)
    def get_width()
    def get_height()
    def is_static_image()
    def get_static_image()
    def get_iter(start_time=0.0)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.gdk.PixbufAnimation
```

Description

A gtk.gdk.PixbufAnimation is an object that holds an animation. An animation is conceptually a series of frames to be displayed over time. Each frame is the same size. The animation may not be represented as a series of frames internally; for example, it may be stored as a sprite and instructions for moving the sprite around a background. To display an animation you don't need to understand its representation, you just ask a gtk.gdk.PixbufAnimationIter for the next frame that should be displayed at a given point in time.

Constructor

```
    gtk.gdk.PixbufAnimation(filename)
```

filename :	the name of the file to load.
Returns :	a new gtk.gdk.PixbufAnimation object.

Creates a new gtk.gdk.PixbufAnimation by loading it from a file. The file format is detected automatically. If the file's format does not support multi–frame images, then an animation with a single frame will be created.

This constructor raises the GError exception if any of several error conditions occurred: the file could not be opened, there was no loader for the file's format, there was not enough memory to allocate the image buffer, or the image file contained invalid data.

Methods

gtk.gdk.PixbufAnimation.get_width

```
def get_width()
```

Returns :	the width of the bounding box of the animation.

The `get_width`() method returns the width of the bounding box of a pixbuf animation.

gtk.gdk.PixbufAnimation.get_height

```
def get_height()
```

Returns :	the height of the bounding box of the animation.

The `get_height()` method returns the height of the bounding box of a pixbuf animation.

gtk.gdk.PixbufAnimation.is_static_image

```
def is_static_image()
```

Returns :	`TRUE` if the "animation" was really just an image

The `is_static_image`() method returns TRUE if you load a file containing a plain, unanimated image. Use the `get_static_image()` method to retrieve the image.

gtk.gdk.PixbufAnimation.get_static_image

```
def get_static_image()
```

Returns :	the unanimated image representing the animation

The `get_static_image`() method returns a `gtk.gdk.Pixbuf` that represents a static image of the animation. If the animation is really just a plain image (has only one frame), this method returns that image. If the animation is an animation, this method returns a reasonable thing to display as a static unanimated image, which might be the first frame, or something more sophisticated. If an animation hasn't loaded any frames yet, this method will return `None`.

gtk.gdk.PixbufAnimation.get_iter

```
def get_iter(start_time=0.0)
```

`start_time` :	the time when the animation starts playing
Returns :	a `gtk.gdk.PixbufAnimationIter` object

The `get_iter`() method returns a `gtk.gdk.PixbufAnimationIter` that is used to access the frames of the animation. The iterator provides the frames that should be displayed at specific times. *`start_time`* is the start time specified as a float as output from the Python time.time() function. *`start_time`* marks the beginning of the animation playback. After creating an iterator, you should immediately display the pixbuf returned by the `gtk.gdk.PixbufAnimationIter.get_pixbuf()` method. Then, you should install a timeout (with the `gobject.timeout_add()`() function) or by some other mechanism ensure that you'll update the image after the number of milliseconds specified by the `gtk.gdk.PixbufAnimationIter.get_delay_time()` method. Each time the image is updated, you should reinstall the timeout with the new, possibly−changed delay time. As a shortcut, if *`start_time`* is 0.0 (the default), the current time will be used.

To update the image (i.e. possibly change the result of the `gtk.gdk.PixbufAnimationIter.get_pixbuf()` method to a new frame of the animation), call the `gtk.gdk.PixbufAnimationIter.advance()` method.

If you're using a `gtk.gdk.PixbufLoader`, in addition to updating the image after the delay time, you should also update it whenever you receive the "area_updated" signal and the `gtk.gdk.PixbufAnimationIter.on_currently_loading_frame()` method returns `TRUE`. In this case, the frame currently being fed into the loader has received new data, so needs to be refreshed. The delay time for a frame may also be modified after an "area_updated" signal, for example if the delay time for a frame is encoded in the data after the frame itself. So your timeout should be reinstalled after any area_updated signal. A delay time of −1 is possible, indicating "infinite."

Prev Up Next
gtk.gdk.Pixbuf Home gtk.gdk.PixbufAnimationIter

gtk.gdk.PixbufAnimationIter

gtk.gdk.PixbufAnimationIter — an object providing access to the frames of a `gtk.gdk.PixbufAnimation`

Synopsis

```
class gtk.gdk.PixbufAnimationIter(gobject.GObject):
    def get_delay_time()
    def get_pixbuf()
    def on_currently_loading_frame()
    def advance(current_time=0.0)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.gdk.PixbufAnimationIter
```

Description

A `gtk.gdk.PixbufAnimationIter` is used to access the frames of a `gtk.gdk.PixbufAnimation` at specified times. A `gtk.gdk.PixbufAnimationIter` object is created using the `gtk.gdk.PixbufAnimation.get_iter()` method. After creating an iterator, you should immediately display the pixbuf returned by the `get_pixbuf()` method. Then, you should install a timeout (with the `gobject.timeout_add()`() function) or by some other mechanism ensure that you'll update the image after the number of milliseconds specified by the `get_delay_time()` method. Each time the image is updated, you should reinstall the timeout with the new, possibly−changed delay time.

To update the image (i.e. possibly change the result of the `get_pixbuf()` method to a new frame of the animation), call the `advance()` method.

If you're using a `gtk.gdk.PixbufLoader`, in addition to updating the image after the delay time, you should also update it whenever you receive the "area_updated" signal and the `on_currently_loading_frame()` method returns `TRUE`. In this case, the frame currently being fed into the loader has received new data, so needs to be refreshed. The delay time for a frame may also be modified after an "area_updated" signal, for example if the delay time for a frame is encoded in the data after the frame itself. So your timeout should be reinstalled after any area_updated signal. A delay time of −1 is possible, indicating "infinite."

Methods

gtk.gdk.PixbufAnimationIter.get_delay_time

```
def get_delay_time()
```

Returns :	the delay time in milliseconds (thousandths of a second)

The `get_delay_time()` method returns the number of milliseconds the current pixbuf should be displayed, or −1 if the current pixbuf should be displayed forever. The `gobject.timeout_add()`() function conveniently takes a timeout in milliseconds, so you can use a timeout to schedule the next update.

gtk.gdk.PixbufAnimationIter.get_pixbuf

```
def get_pixbuf()
```

Returns :	the current `gtk.gdk.Pixbuf` to be displayed

The `get_pixbuf()` method returns the current `gtk.gdk.Pixbuf` that should be displayed. The pixbuf will be the same size as the animation itself (see the `gtk.gdk.PixbufAnimation.get_width()` and `gtk.gdk.PixbufAnimation.get_height()` methods). The `gtk.gdk.Pixbuf` should be displayed for the number of milliseconds specified by the `get_delay_time()` method.

gtk.gdk.PixbufAnimationIter.on_currently_loading_frame

```
def on_currently_loading_frame()
```

Returns :	`TRUE` if the frame we're on is partially loaded, or the last frame

The `on_currently_loading_frame()` method returns `TRUE` if the frame currently pointed to by the iterator is partially loaded or the last frame. This method is used to determine how to respond to the "area_updated" signal on `gtk.gdk.PixbufLoader` when loading an animation. The "area_updated" signal is emitted for an area of the frame currently streaming in to the loader. So if you're on the currently loading frame, you need to redraw the screen for the updated area.

gtk.gdk.PixbufAnimationIter.advance

```
def advance(current_time=0.0)
```

`current_time` :	the current time as a float or 0.0 to automatically determine the current time
Returns :	`TRUE` if the image may need updating

The `advance()` method attempts to advance an animation to a new frame. The frame is chosen based on the start time passed to the `gtk.gdk.PixbufAnimation.get_iter()` method. *`current_time`* is normally the current time (as specified by the Python time.time() function) and must be greater than or equal to the time passed to the `gtk.gdk.PixbufAnimation.get_iter()` method, and must increase or remain unchanged each time the `get_pixbuf()` method is called. That is, you can't go backward in time; animations only play forward. As a shortcut, pass 0.0 (the default) for the current time and the current time will automatically be determined an used. So you only need to explicitly pass *`current_time`* if you're doing something odd like playing the animation at double speed.

If this method returns `FALSE`, there's no need to update the animation display, assuming the display had been rendered prior to advancing; if `TRUE`, you need to call the `get_pixbuf()` method and update the display with the new pixbuf.

Prev	Up	Next
gtk.gdk.PixbufAnimation	Home	gtk.gdk.PixbufLoader

gtk.gdk.PixbufLoader

gtk.gdk.PixbufLoader — an object providing application−driven progressive image loading

Synopsis

```
class gtk.gdk.PixbufLoader(gobject.GObject):
    gtk.gdk.PixbufLoader(image_type=None)
    def write(buf, count=-1)
    def get_pixbuf()
    def get_animation()
    def close()
    def set_size(width, height)
    def get_format()
Functions

    def gtk.gdk.pixbuf_loader_new_with_mime_type(mime_type)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.gdk.PixbufLoader
```

Signal Prototypes

"area−prepared"	`def callback(pixbufloader, user_param1, ...)`
"area−updated"	`def callback(pixbufloader, x, y, width, height, user_param1, ...)`
"closed"	`def callback(pixbufloader, user_param1, ...)`
"size−prepared"	`def callback(pixbufloader, width, height, user_param1, ...)`

Description

A `gtk.gdk.PixbufLoader` provides a way for applications to drive the process of loading an image, by letting them send the image data directly to the loader instead of having the loader read the data from a file. Applications can use this instead of the `gtk.gdk.pixbuf_new_from_file` function or the `gtk.gdk.PixbufAnimation` constructor when they need to parse image data in small chunks. For example, it should be used when reading an image from a (potentially) slow network connection, or when loading an extremely large file.

To use `gtk.gdk.PixbufLoader` to load an image, just create a new one, and call the `write()` method to send the data to it. When done, the `close()` method should be called to end the stream and finalize everything. The loader will emit two important signals throughout the process. The first, "area–prepared", will be called as soon as the image has enough information to determine the size of the image to be used. The application can call the `get_pixbuf()` method to retrieve the pixbuf. No actual image data will be in the pixbuf, so it can be safely filled with any temporary graphics (or an initial color) as needed.

The "area–updated" signal is emitted every time a region is updated. This way you can update a partially completed image. Note that you do not know anything about the completeness of an image from the area updated. For example, in an interlaced image, you need to make several passes before the image is done loading.

Loading an Animation

Loading an animation is almost as easy as loading an image. Once the first "area–prepared" signal has been emitted, you can call the `get_animation()` method to get the `gtk.gdk.PixbufAnimation` object and the `gtk.gdk.PixbufAnimation.get_iter()` method to get an `gtk.gdk.PixbufAnimationIter` for displaying it.

Constructor

```
    gtk.gdk.PixbufLoader(image_type=None)
```

`image_type` :	the name of the image format or `None`
Returns :	A new `gtk.gdk.PixbufLoader` object.

Creates a new `gtk.gdk.PixbufLoader` object. If *`image_type`* is not specified or is `None` the image type will be automatically deduced from the image data. If *`image_type`* is specified the `gtk.gdk.PixbufLoader` attempts to parse the image data as if it were an image of the specified type. Identifying the image type is useful if you want an error if the image isn't the expected type, for loading image formats that can't be reliably identified by looking at the data, or if the user manually forces a specific type.

This constructor raises the GError exception if an error occurs trying to load the module for *`image_type`*.

Methods

gtk.gdk.PixbufLoader.write

```
    def write(buf, count=-1)
```

`buf` :	a string containing some portion of the image data.
`count` :	the length of *`buf`* in bytes.
Returns :	`TRUE` if the write was successful.

The `write()` method causes the pixbuf loader to parse the bytes of an image contained in the string specified by *`buf`*. If *`count`* is specified and is in the range (0, len(buf)) only *`count`* bytes of *`buf`* are used. This method returns `TRUE` if the image data was loaded successfully. If an error occurred this method raises the GError exception and will not accept further writes. The loader may or may not be closed depending on the error.

gtk.gdk.PixbufLoader.get_pixbuf

```
def get_pixbuf()
```

Returns : the `gtk.gdk.Pixbuf` that the loader is creating, or `None` if not enough data has been read to determine how to create the image buffer.

The `get_pixbuf()` method returns the `gtk.gdk.Pixbuf` that a pixbuf loader is currently creating. In general it only makes sense to call this method after the "area–prepared" signal has been emitted by the loader which means that enough data has been read to know the size of the image that will be allocated. If the loader has not received enough data via the `write()` method, this method returns `None`. The same pixbuf will be returned in all future calls to this method. Also, if the loader is an animation, it will return the "static image" of the animation (see the `gtk.gdk.PixbufAnimation.get_static_image()` method).

gtk.gdk.PixbufLoader.get_animation

```
def get_animation()
```

Returns : the `gtk.gdk.PixbufAnimation` that the loader is loading, or `None` if not enough data has been read to determine the information.

The `get_animation()` method returns the `gtk.gdk.PixbufAnimation` that the pixbuf loader is currently creating. In general it only makes sense to call this method after the "area–prepared" signal has been emitted by the loader. If the loader doesn't have enough bytes yet (hasn't emitted the "area–prepared" signal) this method will return `None`.

gtk.gdk.PixbufLoader.close

```
def close()
```

Returns : `TRUE` if all image data written so far was successfully passed out via the "area_update" signal

The `close()` method informs the pixbuf loader that no further writes using the `write()` will occur, so that it can free its internal loading structures. Also, the pixbuf loader tries to parse any data that hasn't yet been parsed and if the remaining data is partial or corrupt, the GError exception will be raised.

gtk.gdk.PixbufLoader.set_size

```
def set_size(width, height)
```

`width` :	The desired width for the image being loaded.
`height` :	The desired height for the image being loaded.

Note

This method is available in PyGTK 2.4 and above.

The `set_size()` method causes the image to be scaled to the size specified by *`width`* and *`height`* while it is being loaded. The desired image size can be determined relative to the original size of the image by calling the `set_size()` from a signal handler for the "size–prepared" signal.

Attempts to set the desired image size are ignored after the emission of the "size–prepared".

gtk.gdk.PixbufLoader.get_format

```
def get_format()
```

Returns :	a Python dict containing the image format information or `None`

Note

This method is available in PyGTK 2.4 and above.

The `get_format()` method returns the available information about the format of the currently loading image file. This method returns None if their is no information available e.g. before the image has started loading.

Functions

gtk.gdk.pixbuf_loader_new_with_mime_type

```
def gtk.gdk.pixbuf_loader_new_with_mime_type(mime_type)
```

`mime_type` :	the mime type to be loaded
Returns :	a new `gtk.gdk.PixbufLoader` object.

Note

This function is available in PyGTK 2.4 and above.

The `gtk.gdk.pixbuf_loader_new_with_mime_type()` function creates a new pixbuf loader object that always attempts to parse image data as if it were an image of the mime type specified by *`mime_type`*, instead of identifying the type automatically. Useful if you want an error if the image isn't the expected mime type, for loading image formats that can't be reliably identified by looking at the data, or if the user manually forces a specific mime type.

This function raises the GError exception if an error occurs during the loading of the mime type image module.

Signals

The "area–prepared" gtk.gdk.PixbufLoader Signal

```
def callback(pixbufloader, user_param1, ...)
```

`pixbufloader` :	the pixbufloader that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "area–prepared" signal is emitted when sufficient image data has been received and parsed to determine the size of the image.

The "area–updated" gtk.gdk.PixbufLoader Signal

```
def callback(pixbufloader, x, y, width, height, user_param1, ...)
```

`pixbufloader` :	the pixbufloader that received the signal

x :	the X coordinate of the region
y :	the Y coordinate of the region
width :	the width of the region
height :	the height of the region
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "area−updated" signal is emitted when sufficient image data has been received and parsed to allow the region specified by *x*, *y*, *width* and *height* to be displayed.

The "closed" gtk.gdk.PixbufLoader Signal

```
    def callback(pixbufloader, user_param1, ...)
```

pixbufloader :	the pixbufloader that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "closed" signal is emitted when the *pixbufloader* is closed by calling the close() method.

The "size−prepared" gtk.gdk.PixbufLoader Signal

```
    def callback(pixbufloader, width, height, user_param1, ...)
```

pixbufloader :	the pixbufloader that received the signal
width :	the original width of the image
height :	he original height of the image
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.2 and above.

The "size−prepared" signal is emitted when the pixbuf loader has been fed the initial amount of data that is required to figure out the size of the image that it will create. Applications can call the set_size() method in response to this signal to set the desired size of the image.

Prev Up Next
gtk.gdk.PixbufAnimationIter Home gtk.gdk.Pixmap

gtk.gdk.Pixmap
Prev **The gtk.gdk Class Reference** Next

gtk.gdk.Pixmap

gtk.gdk.Pixmap — An offscreen gtk.gdk.Drawable

Synopsis

```
class gtk.gdk.Pixmap(gtk.gdk.Drawable):
    gtk.gdk.Pixmap(drawable, width, height, depth=-1)
```

```
Functions

    def gtk.gdk.bitmap_create_from_data(drawable, data, width, height)
    def gtk.gdk.pixmap_create_from_data(drawable, data, width, height, depth, fg, bg)
    def gtk.gdk.pixmap_create_from_xpm(window, transparent_color, filename)
    def gtk.gdk.pixmap_colormap_create_from_xpm(window, colormap, transparent_color, filename)
    def gtk.gdk.pixmap_create_from_xpm_d(window, transparent_color, data)
    def gtk.gdk.pixmap_colormap_create_from_xpm_d(window, colormap, transparent_color, data)
    def gtk.gdk.pixmap_foreign_new(anid)
    def gtk.gdk.pixmap_lookup(anid)
    def gtk.gdk.pixmap_foreign_new_for_display(display, anid)
    def gtk.gdk.pixmap_lookup_for_display(display, anid)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.gdk.Drawable
    +-- gtk.gdk.Pixmap
```

Description

A `gtk.gdk.Pixmap` is an offscreen `gtk.gdk.Drawable`. It can be drawn upon with the standard `gtk.gdk.Drawable` drawing primitives, then copied to another `gtk.gdk.Drawable` (such as a `gtk.gdk.Window`) with the draw_drawable() method. The depth of a pixmap is the number of bits per pixels. A bitmaps are simply a `gtk.gdk.Pixmap` with a depth of 1. (That is, they are monochrome pixmaps – each pixel can be either on or off).

Constructor

```
    gtk.gdk.Pixmap(drawable, width, height, depth=-1)
```

drawable :	a `gtk.gdk.Drawable` used to determine default values for the new pixmap or `None` if *depth* is specified.
width :	the width of the new pixmap in pixels.
height :	the height of the new pixmap in pixels.
depth :	the depth (number of bits per pixel) of the new pixmap or –1 to use the depth of *drawable*.
Returns :	a new `gtk.gdk.Pixmap` object

Creates a new `gtk.gdk.Pixmap` with the size specified by *width* and *height* and the number of bits per pixel specified by *depth*.

Functions

gtk.gdk.bitmap_create_from_data

```
    def gtk.gdk.bitmap_create_from_data(drawable, data, width, height)
```

drawable :	a `gtk.gdk.Drawable` used to determine default values for the new pixmap or `None` to use the root window.
data :	a string containing the XBM data
width :	the width of the new bitmap in pixels.
height :	the height of the new bitmap in pixels.

Returns :	a new bitmap (`gtk.gdk.Pixmap`) object

The `gtk.gdk.bitmap_create_from_data()` function returns a new bitmap of the size specified by *`width`* and *`height`* from the `XBM` format string specified by *`data`*.

gtk.gdk.pixmap_create_from_data

```
    def gtk.gdk.pixmap_create_from_data(drawable, data, width, height, depth, fg, bg)
```

`drawable` :	a `gtk.gdk.Drawable` used to determine default values for the new pixmap or `None` if *`depth`* is specified.
`data` :	the string containing the pixmap data.
`width` :	the width of the new pixmap in pixels.
`height` :	the height of the new pixmap in pixels.
`depth` :	the depth (number of bits per pixel) of the new pixmap or −1 to use the depth of *`drawable`*.
`fg` :	the foreground color.
`bg` :	he background color.
Returns :	a new `gtk.gdk.Pixmap` object

The `gtk.gdk.pixmap_create_from_data()` function creates a two−color `gtk.gdk.Pixmap` of the size specified by *`width`* and *`height`* from the `XBM` format string specified by *`data`*. The foreground and background colors of the pixmap are specified by *`fg`* and *`bg`* respectively. If *`depth`* is −1 *`drawable`* is used to determine the bits per pixels otherwise the value of *`depth`* is used.

gtk.gdk.pixmap_create_from_xpm

```
    def gtk.gdk.pixmap_create_from_xpm(window, transparent_color, filename)
```

`window` :	a `gtk.gdk.Drawable`, used to determine default values for the new `gtk.gdk.Pixmap`.
`transparent_color` :	the color to be used for the pixels that are transparent in the input file or `None` to use a default color.
`filename` :	the name of a file containing `XPM` data.
Returns :	a tuple containing a new `gtk.gdk.Pixmap` object and a bitmap that is the transparency mask.

The `gtk.gdk.pixmap_create_from_xpm()` function returns a tuple containing a `gtk.gdk.Pixmap` and a bitmap transparency mask created from the `XPM` data in the file specified by *`filename`*. *`transparent_color`* (if not `None`) specifies the `gtk.gdk.Color` to by used for the transparent pixels.

gtk.gdk.pixmap_colormap_create_from_xpm

```
    def gtk.gdk.pixmap_colormap_create_from_xpm(window, colormap, transparent_color, filename)
```

`window` :	a `gtk.gdk.Drawable`, used to determine default values for the new `gtk.gdk.Pixmap` or `None` if a `gtk.gdk.Colormap` is specified.
`colormap` :	the `gtk.gdk.Colormap` that the new `gtk.gdk.Pixmap` will be use or `None` to use the colormap of *`window`*.
`transparent_color` :	the color to be used for the pixels that are transparent in the input file or `None` to use a default color.
`filename` :	the name of a file containing `XPM` data.
Returns :	a tuple containing a new `gtk.gdk.Pixmap` object and a bitmap that is the transparency mask.

The `gtk.gdk.pixmap_colormap_create_from_xpm()` function returns a tuple containing a `gtk.gdk.Pixmap` and a bitmap transparency mask created from the `XPM` data in the file specified by *`filename`*. *`transparent_color`* (if not `None`) specifies the `gtk.gdk.Color` to by used for the transparent pixels. If *`colormap`* is not `None` it must specify a `gtk.gdk.Colormap` that the new `gtk.gdk.Pixmap` will use. If *`colormap`* is `None` the new `gtk.gdk.Pixmap` will use the colormap of *`window`*.

gtk.gdk.pixmap_create_from_xpm_d

```
    def gtk.gdk.pixmap_create_from_xpm_d(window, transparent_color, data)
```

`window` :	a `gtk.gdk.Drawable`, used to determine default values for the new `gtk.gdk.Pixmap`.
`transparent_color` :	the color to be used for the pixels that are transparent in the input file or `None` to use a default color.
`data` :	a list of strings containing the `XPM` data.
Returns :	a tuple containing a new `gtk.gdk.Pixmap` object and a bitmap that is the transparency mask.

The `gtk.gdk.pixmap_create_from_xpm_d()` function returns a tuple containing a new `gtk.gdk.Pixmap` and a bitmap transparency mask created from the `XPM` data contained in *`data`*. *`transparent_color`* (if not `None`) specifies the `gtk.gdk.Color` to by used for the transparent pixels.

gtk.gdk.pixmap_colormap_create_from_xpm_d

```
    def gtk.gdk.pixmap_colormap_create_from_xpm_d(window, colormap, transparent_color, data)
```

`window` :	a `gtk.gdk.Drawable`, used to determine default values for the new `gtk.gdk.Pixmap`.
`colormap` :	the `gtk.gdk.Colormap` that the new `gtk.gdk.Pixmap` will be use or `None` to use the colormap of *`window`*.
`transparent_color` :	the color to be used for the pixels that are transparent in the input file or `None` to use a default color.
`data` :	a list of strings containing the `XPM` data.
Returns :	a tuple containing a new `gtk.gdk.Pixmap` object and a bitmap that is the transparency mask.

The `gtk.gdk.pixmap_colormap_create_from_xpm_d()` function returns a tuple containing a new `gtk.gdk.Pixmap` and a bitmap transparency mask created from the `XPM` data contained in *`data`*. *`transparent_color`* (if not `None`) specifies the `gtk.gdk.Color` to by used for the transparent pixels. If *`colormap`* is not `None` it must specify a `gtk.gdk.Colormap` that the new `gtk.gdk.Pixmap` will use. If *`colormap`* is `None` the new `gtk.gdk.Pixmap` will use the colormap of *`window`*.

gtk.gdk.pixmap_foreign_new

```
    def gtk.gdk.pixmap_foreign_new(anid)
```

`anid` :	a native window system pixmap handle.
Returns :	the new `gtk.gdk.Pixmap` wrapper for the native pixmap or `None` if the pixmap has been destroyed.

The `gtk.gdk.pixmap_foreign_new()` function returns a `gtk.gdk.Pixmap` that wraps the native window specified by *`anid`* for the default display. If the pixmap has been destroyed this function returns `None`. In the X backend, *`anid`* must specify an Xlib XID that is a native pixmap handle.

gtk.gdk.pixmap_lookup

```
    def gtk.gdk.pixmap_lookup(anid)
```

anid :	a native window system pixmap handle.
Returns :	the new gtk.gdk.Pixmap wrapper for the native pixmap or None if the pixmap has been destroyed.

The `gtk.gdk.pixmap_lookup()` function returns looks up and returns the gtk.gdk.Pixmap that wraps the native pixmap handle specified by *anid*. This method returns None if no gtk.gdk.Pixmap wraps *anid*. In the X backend, *anid* must specify an Xlib XID that is a native pixmap handle.

gtk.gdk.pixmap_foreign_new_for_display

```
    def gtk.gdk.pixmap_foreign_new_for_display(display, anid)
```

display :	a gtk.gdk.Display object
anid :	a native window system pixmap handle.
Returns :	the new gtk.gdk.Pixmap wrapper for the native pixmap or None if the pixmap has been destroyed.

Note

This function is available in PyGTK2.2 and above.

The `gtk.gdk.pixmap_foreign_new_for_display()` function returns a gtk.gdk.Pixmap that wraps the native window specified by *anid* for the gtk.gdk.Display specified by *display*. If the pixmap has been destroyed this function returns None. In the X backend, *anid* must specify an Xlib XID that is a native pixmap handle.

gtk.gdk.pixmap_lookup_for_display

```
    def gtk.gdk.pixmap_lookup_for_display(display, anid)
```

display :	a gtk.gdk.Display object
anid :	a native window system pixmap handle.
Returns :	the new gtk.gdk.Pixmap wrapper for the native pixmap or None if the pixmap has been destroyed.

Note

This function is available in PyGTK2.2 and above.

The `gtk.gdk.pixmap_lookup_for_display()` function returns looks up and returns the gtk.gdk.Pixmap that wraps the native pixmap handle specified by *anid* for the gtk.gdk.Display specified by *display*. This method returns None if no gtk.gdk.Pixmap wraps *anid*. In the X backend, *anid* must specify an Xlib XID that is a native pixmap handle.

Prev	Up	Next
gtk.gdk.PixbufLoader	Home	gtk.gdk.Rectangle

gtk.gdk.Rectangle

gtk.gdk.Rectangle — an object holding data about a rectangle

Synopsis

```
class gtk.gdk.Rectangle(gobject.GBoxed):
    gtk.gdk.Rectangle(x=0, y=0, width=0, height=0)
    def intersect(src)
    def union(src)
```

Attributes

"x"	Read–Write	The X coordinate of the top left corner of the rectangle.
"y"	Read–Write	The Y coordinate of the top left corner of the rectangle.
"width"	Read–Write	The width of the rectangle.
"height"	Read–Write	The height of the rectangle.

Description

A `gtk.gdk.Rectangle` holds the position and size of a rectangle. The position is specified by the "x" and "y" attributes and the size, by the "width" and "height" attributes.

Constructor

```
    gtk.gdk.Rectangle(x=0, y=0, width=0, height=0)
```

x :	the X coordinate of the top left corner of the rectangle
y :	the Y coordinate of the top left corner of the rectangle
width :	the width of the rectangle
height :	the height of the rectangle
Returns :	a new `gtk.gdk.Rectangle` object

Creates a new `gtk.gdk.Rectangle` with the attributes specified by *x*, *y*, *width* and *height*. Any unspecified attributes default to 0.

Methods

gtk.gdk.Rectangle.intersect

```
    def intersect(src)
```

src :	a `gtk.gdk.Rectangle` of a 4–tuple specifying the attributes of a rectangle as (x, y, width, height)
Returns :	a `gtk.gdk.Rectangle` that is the intersection of *src* and the rectangle

The `intersect`() method returns a `gtk.gdk.Rectangle` that is the intersection of this rectangle and the `gtk.gdk.Rectangle` specified by *src*. The value of *src* is either a `gtk.gdk.Rectangle` or a

4–tuple containing the position and size of a rectangle. If the rectangles do not intersect the returned gtk.gdk.Rectangle will have all attributes set to 0.

gtk.gdk.Rectangle.union

```
    def union(src)
```

src :	a gtk.gdk.Rectangle of a 4–tuple specifying the attributes of a rectangle as (x, y, width, height)
Returns :	a gtk.gdk.Rectangle that includes both *src* and the rectangle

The union() method returns a gtk.gdk.Rectangle that is the smallest rectangle containing both this rectangle and the gtk.gdk.Rectangle specified by *src*. The value of *src* is either a gtk.gdk.Rectangle or a 4–tuple containing the position and size of a rectangle.

Prev Up Next
gtk.gdk.Pixmap Home gtk.gdk.Screen

gtk.gdk.Screen

gtk.gdk.Screen — an object representing a physical screen

Synopsis

```
class gtk.gdk.Screen(gobject.GObject):
    def get_default_colormap()
    def set_default_colormap(colormap)
    def get_system_colormap()
    def get_system_visual()
    def get_rgb_colormap()
    def get_rgb_visual()
    def get_root_window()
    def get_display()
    def get_number()
    def get_width()
    def get_height()
    def get_width_mm()
    def get_height_mm()
    def list_visuals()
    def get_toplevel_windows()
    def make_display_name()
    def get_n_monitors()
    def get_monitor_geometry(monitor_num)
    def get_monitor_at_point(x, y)
    def get_monitor_at_window(window)
    def broadcast_client_message(event)
    def get_setting(name)
```

```
Functions

    def gtk.gdk.screen_width()
    def gtk.gdk.screen_height()
    def gtk.gdk.screen_width_mm()
    def gtk.gdk.screen_height_mm()
    def gtk.gdk.screen_get_default()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.gdk.Screen
```

Signal Prototypes

"size–changed"	`def callback(screen, user_param1, ...)`

Description

Note

This object is available in PyGTK 2.2 and above.

`gtk.gdk.Screen` objects are the `PyGTK` representation of a physical screen. It is used throughout `PyGTK` to specify which screen the top level windows are to be displayed on. It is also used to query the screen specification and default settings such as the default colormap (the `get_default_colormap()` method), the screen width (the `get_width()` method), etc.

Note

a screen may consist of multiple monitors that are merged to form a large screen area.

Methods

gtk.gdk.Screen.get_default_colormap

```
  def get_default_colormap()
```

Returns :	the default `gtk.gdk.Colormap`.

Note

This method is available in PyGTK 2.2 and above.

The `get_default_colormap()` method returns the default `gtk.gdk.Colormap` for the screen.

gtk.gdk.Screen.set_default_colormap

```
  def set_default_colormap(colormap)
```

`colormap` :	a `gtk.gdk.Colormap`

Note

This method is available in PyGTK 2.2 and above.

The `set_default_colormap()` method sets the `gtk.gdk.Colormap` specified by *`colormap`* as the default colormap for the screen.

gtk.gdk.Screen.get_system_colormap

```
def get_system_colormap()
```

Returns :	the default colormap for the screen. Since: 2.2

Note

This method is available in PyGTK 2.2 and above.

The `get_system_colormap()` method returns the system's default colormap for the screen

gtk.gdk.Screen.get_system_visual

```
def get_system_visual()
```

Returns :	the system `gtk.gdk.Visual`

Note

This method is available in PyGTK 2.2 and above.

The `get_system_visual()` method returns the system's default `gtk.gdk.Visual` for the screen. This is the visual for the root window of the display.

gtk.gdk.Screen.get_rgb_colormap

```
def get_rgb_colormap()
```

Returns :	a `gtk.gdk.Colormap`

Note

This method is available in PyGTK 2.2 and above.

The `get_rgb_colormap()` method returns the preferred colormap for rendering image data on the screen. Not a very useful function; historically, `GDK` could only render RGB image data to one colormap and visual, but in the current version it can render to any colormap and visual. So there's no need to call this function.

gtk.gdk.Screen.get_rgb_visual

```
def get_rgb_visual()
```

Returns :	a `gtk.gdk.Visual`

Note

This method is available in PyGTK 2.2 and above.

The `get_rgb_visual()` method returns a "preferred visual" chosen for rendering `RGB` image data on the screen.

gtk.gdk.Screen.get_root_window

```
def get_root_window()
```

Returns :	the root gtk.gdk.Window

Note

This method is available in PyGTK 2.2 and above.

The `get_root_window`() method returns the root gtk.gdk.Window of the screen.

gtk.gdk.Screen.get_display

```
    def get_display()
```

Returns :	the display that the screen belongs to

Note

This method is available in PyGTK 2.2 and above.

The `get_display`() method returns the gtk.gdk.Display that the screen belongs to.

gtk.gdk.Screen.get_number

```
    def get_number()
```

Returns :	the index

Note

This method is available in PyGTK 2.2 and above.

The `get_number`() method returns the index of the screen among the screens in its display. (See the `get_display()` method)

gtk.gdk.Screen.get_width

```
    def get_width()
```

Returns :	the width of the screen in pixels.

Note

This method is available in PyGTK 2.2 and above.

The `get_width`() method returns the width of the screen in pixels

gtk.gdk.Screen.get_height

```
    def get_height()
```

Returns :	the height of the screen in pixels.

Note

This method is available in PyGTK 2.2 and above.

The `get_height`() method returns the height of the screen in pixels

gtk.gdk.Screen.get_width_mm

```
def get_width_mm()
```

Returns : the width of the screen in millimeters.

Note

This method is available in PyGTK 2.2 and above.

The `get_width_mm()` method returns the width of the screen in millimeters. Note that on some X servers this value will not be correct.

gtk.gdk.Screen.get_height_mm

```
def get_height_mm()
```

Returns : the height of the screen in pixels. Since: 2.2

Note

This method is available in PyGTK 2.2 and above.

The `get_height_mm()` method returns the height of the screen in millimeters. Note that on some X servers this value will not be correct.

gtk.gdk.Screen.list_visuals

```
def list_visuals()
```

Returns : a list of `gtk.gdk.Visual`

Note

This method is available in PyGTK 2.2 and above.

The `list_visuals()` method returns a list of the available visuals for the screen. A `gtk.gdk.Visual` describes a hardware image data format. For example, a visual might support 24−bit color, or 8−bit color, and might expect pixels to be in a certain format.

gtk.gdk.Screen.get_toplevel_windows

```
def get_toplevel_windows()
```

Returns : a list of the toplevel `gtk.gdk.Window` objects

Note

This method is available in PyGTK 2.2 and above.

The `get_toplevel_windows()` method returns a list of all toplevel `gtk.gdk.Window` objects known to `PyGTK` on the screen. A toplevel window is a child of the root window (see the `gtk.gdk.get_default_root_window()` function).

gtk.gdk.Screen.make_display_name

```
def make_display_name()
```

Returns :	a generated nae

Note

This method is available in PyGTK 2.2 and above.

The `make_display_name`() method determines the name to pass to `gtk.gdk.Display()` to get a `gtk.gdk.Display` with this screen as the default screen.

gtk.gdk.Screen.get_n_monitors

```
def get_n_monitors()
```

Returns :	the number of monitors that the screen consists of.

Note

This method is available in PyGTK 2.2 and above.

The `get_n_monitors`() method returns the number of monitors that the screen consists of.

gtk.gdk.Screen.get_monitor_geometry

```
def get_monitor_geometry(monitor_num)
```

`monitor_num` :	the monitor number.
Returns :	a `gtk.gdk.Rectangle` containing the monitor geometry

Note

This method is available in PyGTK 2.2 and above.

The `get_monitor_geometry`() method returns a `gtk.gdk.Rectangle` representing the size and position of the individual monitor within the the entire screen area.

Note that the size of the entire screen area can be retrieved via the `get_width()` and `get_height()`. methods.

gtk.gdk.Screen.get_monitor_at_point

```
def get_monitor_at_point(x, y)
```

`x` :	an x coordinate in the virtual screen.
`y` :	a y coordinate in the virtual screen.
Returns :	the number of the monitor that the point (*x*,*y*) lies in, or a monitor close to (*x*,*y*) if the point is not in any monitor.

Note

This method is available in PyGTK 2.2 and above.

The `get_monitor_at_point`() method returns the number of the monitor in which the point (*x*,*y*) is located or the closest monitor if not in a monitor.

gtk.gdk.Screen.get_monitor_at_window

```
    def get_monitor_at_window(window)
```

`window` :	a `gtk.gdk.Window`
Returns :	the number of the monitor that most of *`window`* is located. Since: 2.2

Note

This method is available in PyGTK 2.2 and above.

The `get_monitor_at_window`() method returns the number of the monitor that most of the `gtk.gdk.Window` specified by *`window`* is in. If *`window`* does not intersect any monitors, the closest monitor to the main bounding rectangle of *`window`* is returned.

gtk.gdk.Screen.broadcast_client_message

```
    def broadcast_client_message(event)
```

`event` :	the `gtk.gdk.Event`.

Note

This method is available in PyGTK 2.2 and above.

The `broadcast_client_message`() method sends a message to all top level windows. On X11, sends an X ClientMessage event to all toplevel windows on the screen.

Toplevel windows are determined by checking for the `WM_STATE` property, as described in the Inter−Client Communication Conventions Manual (ICCCM). If no windows are found with the `WM_STATE` property set, the message is sent to all children of the root window.

On Windows, broadcasts a message registered with the name `GDK_WIN32_CLIENT_MESSAGE` to all top−level windows. The amount of data is limited to one long, i.e. four bytes.

gtk.gdk.Screen.get_setting

```
    def get_setting(name)
```

`name` :	the name of the setting
Returns :	the value of *`setting`*

Note

This method is available in PyGTK 2.2 and above.

The `get_setting`() method returns the value of the desktop−wide setting (specified by *`setting`*) such as double−click time for the screen.

Functions

gtk.gdk.screen_width

```
def gtk.gdk.screen_width()
```

Returns :	the width of the default screen in pixels.

The `gtk.gdk.screen_width`() function returns the width of the default screen in pixels.

gtk.gdk.screen_height

```
def gtk.gdk.screen_height()
```

Returns :	the height of the default screen in pixels.

The `gtk.gdk.screen_height`() function returns the height of the default screen in pixels.

gtk.gdk.screen_width_mm

```
def gtk.gdk.screen_width_mm()
```

Returns :	the width of the default screen in millimeters, though it is not always correct.

The `gtk.gdk.screen_width_mm`() function returns the width of the default screen in millimeters. Note that on many X servers this value will not be correct.

gtk.gdk.screen_height_mm

```
def gtk.gdk.screen_height_mm()
```

Returns :	the height of the default screen in millimeters, though it is not always correct.

The `gtk.gdk.screen_height_mm`() function returns the height of the default screen in millimeters. Note that on many X servers this value will not be correct.

gtk.gdk.screen_get_default

```
def gtk.gdk.screen_get_default()
```

Returns :	a `gtk.gdk.Screen`, or `None` if there is no default display.

Note

This function is available in PyGTK 2.2 and above.

The `gtk.gdk.screen_get_default`() function returns the default `gtk.gdk.Screen` for the default `gtk.gdk.Display`. (See the `gtk.gdk.display_get_default()` function).

Signals

The "size−changed" gtk.gdk.Screen Signal

```
    def callback(screen, user_param1, ...)
```

screen :	the screen that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "size−changed" signal is emitted when the pixel width or height of a screen changes.

gtk.gdk.Visual

gtk.gdk.Visual — an object containing hardware display information

Synopsis

```
class gtk.gdk.Visual(gobject.GObject):
    gtk.gdk.Visual(depth, visual_type)
    def get_screen()
Functions

    def gtk.gdk.list_visuals()
    def gtk.gdk.visual_get_best()
    def gtk.gdk.visual_get_best_depth()
    def gtk.gdk.visual_get_best_type()
    def gtk.gdk.visual_get_best_with_depth(depth)
    def gtk.gdk.visual_get_best_with_type(type)
    def gtk.gdk.visual_get_system()
    def gtk.gdk.query_depths()
    def gtk.gdk.query_visual_types()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.gdk.Visual
```

Attributes

"bits_per_rgb"	Read	The number of significant bits per red, green, or blue when specifying colors for this visual. (For instance, for the `gtk.gdk.Colormap.alloc_color()` method)
"blue_mask"	Read	A mask giving the bits in a pixel value that correspond to the blue field.
"blue_prec"	Read	See above.
"blue_shift"	Read	The *blue_shift* and *blue_prec* give an alternate presentation of the information in *blue_mask*.

"byte_order"	Read	The byte−order for this visual − either: `gtk.gdk.LSB_FIRST` or `gtk.gdk.MSB_FIRST`.
"colormap_size"	Read	The number of entries in the colormap, for visuals of type `gtk.gdk.VISUAL_PSEUDO_COLOR` or `gtk.gdk.VISUAL_GRAY_SCALE`. For other visual types, it is the number of possible levels per color component. If the visual has different numbers of levels for different components, the value of this field is undefined.
"depth"	Read	The number of bits per pixel.
"green_mask"	Read	A mask giving the bits in a pixel value that correspond to the green field.
"green_prec"	Read	See above.
"green_shift"	Read	The *`green_shift`* and *`green_prec`* give an alternate presentation of the information in *`green_mask`*.
"red_mask"	Read	A mask giving the bits in a pixel value that correspond to the red field. Significant only for `gtk.gdk.VISUAL_PSEUDOCOLOR` and `gtk.gdk.VISUAL_DIRECTCOLOR`.
"red_prec"	Read	See above.
"red_shift"	Read	The *`red_shift`* and *`red_prec`* give an alternate presentation of the information in *`red_mask`*. *`red_mask`* is a contiguous sequence of *`red_prec`* bits starting at bit number *`red_shift`*.
"type"	Read	The type of this visual − one of: `gtk.gdk.VISUAL_STATIC_GRAY`, `gtk.gdk.VISUAL_GRAYSCALE`, `gtk.gdk.VISUAL_STATIC_COLOR`, `gtk.gdk.VISUAL_PSEUDO_COLOR`, `gtk.gdk.VISUAL_TRUE_COLOR`, `gtk.gdk.VISUAL_DIRECT_COLOR`

Description

A `gtk.gdk.Visual` describes a particular video hardware display format. It includes information about the number of bits used for each color, the way the bits are translated into an RGB value for display, and the way the bits are stored in memory. For example, a piece of display hardware might support 24−bit color, 16−bit color, or 8−bit color; meaning 24/16/8−bit pixel sizes. For a given pixel size, pixels can be in different formats; for example the "red" element of an RGB pixel may be in the top 8 bits of the pixel, or may be in the lower 4 bits.

Usually you can avoid thinking about visuals in `PyGTK`. Visuals are useful to interpret the contents of a `gtk.gdk.Image`, but you should avoid `gtk.gdk.Image` precisely because its contents depend on the display hardware; use `gtk.gdk.Pixbuf` instead, for all but the most low−level purposes. Also, anytime you provide a `gtk.gdk.Colormap`, the visual is implied as part of the colormap (see the `gtk.gdk.Colormap.get_visual()` method), so you won't have to provide a visual in addition.

There are several standard visuals. The visual returned by the `gtk.gdk.visual_get_system()` function is the system's default visual. The `gtk.gdk.rgb_get_visual()` function returns the visual most suited to displaying full−color image data. If you use the calls in GdkRGB, you should create your windows using this visual (and the colormap returned by the `gtk.gdk.rgb_get_colormap()` function).

A number of functions are provided for determining the "best" available visual. For the purposes of making this determination, higher bit depths are considered better, and for visuals of the same bit depth, `gtk.gdk.VISUAL_PSEUDO_COLOR` is preferred at 8bpp, otherwise, the visual types are ranked in the order of (highest to lowest) `gtk.gdk.VISUAL_DIRECT_COLOR`, `gtk.gdk.VISUAL_TRUE_COLOR`, `gtk.gdk.VISUAL_PSEUDO_COLOR`, `gtk.gdk.VISUAL_STATIC_COLOR`, `gtk.gdk.VISUAL_GRAYSCALE`, then `gtk.gdk.VISUAL_STATIC_GRAY`.

Constructor

```
gtk.gdk.Visual(depth, visual_type)
```

depth :	a bit depth
visual_type :	a visual type
Returns :	the best `gtk.gdk.Visual` with both *depth* and *visual_type*, or `None` if none

Creates a new `gtk.gdk.Visual` that is the best choice for the specified *depth* and *visual_type*. Color visuals and visuals with mutable colormaps are preferred over grayscale or fixed–colormap visuals and visuals with higher color depths are considered better. The value of *visual_type* must be one of:

`gtk.gdk.VISUAL_STATIC_GRAY`	Each pixel value indexes a grayscale value directly.
`gtk.gdk.VISUAL_GRAYSCALE`	Each pixel is an index into a color map that maps pixel values into grayscale values. The color map can be changed by an application.
`gtk.gdk.VISUAL_STATIC_COLOR`	Each pixel value is an index into a predefined, unmodifiable color map that maps pixel values into RGB values.
`gtk.gdk.VISUAL_PSEUDO_COLOR`	Each pixel is an index into a color map that maps pixel values into rgb values. The color map can be changed by an application.
`gtk.gdk.VISUAL_TRUE_COLOR`	Each pixel value directly contains red, green, and blue components. The red_mask, green_mask, and blue_mask fields of the `gtk.gdk.Visual` structure describe how the components are assembled into a pixel value.
`gtk.gdk.VISUAL_DIRECT_COLOR`	Each pixel value contains red, green, and blue components as for `gtk.gdk.TRUE_COLOR`, but the components are mapped via a color table into the final output table instead of being converted directly.

Methods

gtk.gdk.Visual.get_screen

```
def get_screen()
```

Returns :	the screen that this visual belongs to.

Note

This method is available in PyGTK 2.2 and above.

The `get_screen()` method returns the `gtk.gdk.Screen` that this visual belongs to.

Functions

gtk.gdk.list_visuals

```
def gtk.gdk.list_visuals()
```

Returns :	a list of gtk.gdk.Visual objects

The `gtk.gdk.list_visuals()` function returns a list containing the available `gtk.gdk.Visual` objects for the default screen.

gtk.gdk.visual_get_best

```
def gtk.gdk.visual_get_best()
```

Returns :	the best gtk.gdk.Visual for the default screen

The `gtk.gdk.visual_get_best()` function returns the visual with the most available colors for the default screen.

gtk.gdk.visual_get_best_depth

```
def gtk.gdk.visual_get_best_depth()
```

Returns :	the gtk.gdk.Visual with the best depth.

The `gtk.gdk.visual_get_best_depth()` function returns the `gtk.gdk.Visual` with the best depth for the default screen where "best" means the largest

gtk.gdk.visual_get_best_type

```
def gtk.gdk.visual_get_best_type()
```

Returns :	the gtk.gdk.Visual with the best type for the default screen

The `gtk.gdk.visual_get_best_type()` function returns the best available `gtk.gdk.Visual` type for the default screen.

gtk.gdk.visual_get_best_with_depth

```
def gtk.gdk.visual_get_best_with_depth(depth)
```

`depth` :	the number of bits per pixel
Returns :	the best gtk.gdk.Visual for the specified *`depth`*

The `gtk.gdk.visual_get_best_with_depth()` function returns the best `gtk.gdk.Visual` with the specified *`depth`*.

gtk.gdk.visual_get_best_with_type

```
def gtk.gdk.visual_get_best_with_type(type)
```

`type` :	a visual type
Returns :	the best gdkVisual for the visual type specified by *`type`*

The `gtk.gdk.visual_get_best_with_type()` function returns the best `gtk.gdk.Visual` for the visual type specified by *`type`*. The value of *`visual_type`* must be one of:

`gtk.gdk.VISUAL_STATIC_GRAY`	Each pixel value indexes a grayscale value directly.
`gtk.gdk.VISUAL_GRAYSCALE`	

	Each pixel is an index into a color map that maps pixel values into grayscale values. The color map can be changed by an application.
`gtk.gdk.VISUAL_STATIC_COLOR`	Each pixel value is an index into a predefined, unmodifiable color map that maps pixel values into RGB values.
`gtk.gdk.VISUAL_PSEUDO_COLOR`	Each pixel is an index into a color map that maps pixel values into rgb values. The color map can be changed by an application.
`gtk.gdk.VISUAL_TRUE_COLOR`	Each pixel value directly contains red, green, and blue components. The red_mask, green_mask, and blue_mask fields of the `gtk.gdk.Visual` structure describe how the components are assembled into a pixel value.
`gtk.gdk.VISUAL_DIRECT_COLOR`	Each pixel value contains red, green, and blue components as for `gtk.gdk.TRUE_COLOR`, but the components are mapped via a color table into the final output table instead of being converted directly.

gtk.gdk.visual_get_system

```
def gtk.gdk.visual_get_system()
```

The `gtk.gdk.visual_get_system`() function returns the default `gtk.gdk.Visual` for the system's default screen.

gtk.gdk.query_depths

```
def gtk.gdk.query_depths()
```

Note

This function is available in PyGTK 2.4 and above.

The `gtk.gdk.query_depths`() function returns a tuple containing the unique supported visual depths for the default screen. It's equivalent to listing the visuals (see the `gtk.gdk.list_visuals()` function) and then looking at the depth field in each visual, removing duplicates.

gtk.gdk.query_visual_types

```
def gtk.gdk.query_visual_types()
```

The `gtk.gdk.query_visual_types`() function returns a tuple containing the unique visual types supported by the default screen.It's equivalent to listing the visuals (see the `gtk.gdk.list_visuals()` function) and then looking at the type field in each visual, removing duplicates.

The returned value will be one of:

`gtk.gdk.VISUAL_STATIC_GRAY`	Each pixel value indexes a grayscale value directly.
`gtk.gdk.VISUAL_GRAYSCALE`	Each pixel is an index into a color map that maps pixel values into grayscale values. The color map can be changed by an application.
`gtk.gdk.VISUAL_STATIC_COLOR`	Each pixel value is an index into a predefined, unmodifiable color map that maps pixel values into RGB values.
`gtk.gdk.VISUAL_PSEUDO_COLOR`	Each pixel is an index into a color map that maps pixel values into rgb values. The color map can be changed by an application.

`gtk.gdk.VISUAL_TRUE_COLOR`	Each pixel value directly contains red, green, and blue components. The red_mask, green_mask, and blue_mask fields of the `gtk.gdk.Visual` structure describe how the components are assembled into a pixel value.
`gtk.gdk.VISUAL_DIRECT_COLOR`	Each pixel value contains red, green, and blue components as for `gtk.gdk.VISUAL_TRUE_COLOR`, but the components are mapped via a color table into the final output table instead of being converted directly.

gtk.gdk.Window

gtk.gdk.Window — on–screen display areas in the target window system

Synopsis

```
class gtk.gdk.Window(gtk.gdk.Drawable):
    gtk.gdk.Window(parent, width, height, window_type, event_mask, wclass, title=None, x=-1, y=
    def drag_begin(targets)
    def input_set_extension_events(mask, mode)
    def property_get(property, type=None, pdelete=FALSE)
    def property_change(property, type, format, mode, data)
    def property_delete(property)
    def selection_convert(selection, target, time)
    def set_keep_above(setting)
    def set_keep_below(setting)
    def destroy()
    def get_window_type()
    def show()
    def hide()
    def withdraw()
    def move(x, y)
    def resize(width, height)
    def move_resize(x, y, width, height)
    def reparent(new_parent, x, y)
    def clear()
    def clear_area(x, y, width, height)
    def clear_area_e(x, y, width, height)
    def raise_()
    def lower()
    def focus(timestamp=0L)
    def set_user_data(user_data)
    def get_user_data()
    def set_override_redirect(override_redirect)
    def add_filter(function, data=None)
    def scroll(dx, dy)
    def shape_combine_mask(shape_mask, offset_x, offset_y)
    def set_child_shapes()
    def merge_child_shapes()
    def is_visible()
    def is_viewable()
    def get_state()
    def set_static_gravities(use_static)
    def set_type_hint(hint)
    def set_modal_hint(modal)
    def set_skip_taskbar_hint(skips_taskbar)
    def set_skip_pager_hint(skips_pager)
    def set_geometry_hints(min_width=-1, min_height=-1, max_width=-1, max_height=-1, base_widt
    def begin_paint_rect(rectangle)
    def end_paint()
    def set_title(title)
    def set_role(role)
    def set_transient_for(leader)
    def set_background(color)
    def set_back_pixmap(pixmap, parent_relative)
    def set_cursor(cursor)
    def get_geometry()
    def get_position()
    def get_origin()
    def get_root_origin()
    def get_frame_extents()
    def get_pointer()
    def get_parent()
    def get_toplevel()
    def get_children()
    def get_events()
    def set_events(event_mask)
    def set_icon_list(pixbufs)
    def set_icon(icon_window, pixmap, mask)
    def set_icon_name(name)
    def set_group(leader)
    def get_group()
    def set_decorations(decorations)
    def get_decorations()
    def set_functions(functions)
    def iconify()
    def deiconify()
    def stick()
    def unstick()
    def maximize()
    def unmaximize()
    def fullscreen()
    def unfullscreen()
    def register_dnd()
    def begin_resize_drag(edge, button, root_x, root_y, timestamp)
    def begin_move_drag(button, root_x, root_y, timestamp)
    def invalidate_rect(rect, invalidate_children)
    def freeze_updates()
    def thaw_updates()
    def process_updates(update_children)
    def set_accept_focus(accept_focus)
    def enable_synchronized_configure()
    def configure_finished()
    def set_focus_on_map(focus_on_map)
```

Functions

```
    def gtk.gdk.window_foreign_new(anid)
    def gtk.gdk.window_foreign_new_for_display(display, anid)
    def gtk.gdk.get_default_root_window()
    def gtk.gdk.window_get_toplevels()
    def gtk.gdk.window_lookup(anid)
    def gtk.gdk.window_lookup_for_display(display, anid)
    def gtk.gdk.window_process_all_updates()
    def gtk.gdk.gdk_window_set_debug_updates(setting)
    def gtk.gdk.window_at_pointer()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.gdk.Drawable
    +-- gtk.gdk.Window
```

Description

`gtk.gdk.Window` is a rectangular region on the screen. It's a low–level object, used to implement high–level objects such as `gtk.Widget` and `gtk.Window`. A `gtk.Window` is a toplevel window, the object a user might think of as a "window" with a titlebar and so on. A `gtk.Window` may contain several `gtk.gdk.Window` objects since most widgets use a `gtk.gdk.Window`.

A `gtk.gdk.Window` object interacts with the native window system for input and events. Some `gtk.Widget` objects do not have an associated `gtk.gdk.Window` and therefore cannot receive events. To receive events on behalf of these "windowless" widgets a `gtk.EventBox` must be used.

Constructor

```
    gtk.gdk.Window(parent, width, height, window_type, event_mask, wclass, title=None, x=-1, y=-
```

`parent` :	a `gtk.gdk.Window`
`width` :	the width of the window in pixels
`height` :	the height of the window in pixels
`window_type` :	the window type
`event_mask` :	the bitmask of events received by the window
`wclass` :	the class of window – either `gtk.gdk.INPUT_OUTPUT` or `gtk.gdk.INPUT_ONLY`
`title` :	the window title if a toplevel window
`x` :	the x coordinate of the window position relative to *`parent`*
`y` :	the y coordinate of the window position relative to *`parent`*
`visual` :	the `gtk.gdk.Visual` for the window
`colormap` :	the `gtk.gdk.Colormap` for the window
`cursor` :	the `gtk.gdk.Cursor` for the window
`wmclass_name` :	don't use this – see the `gtk.Window.set_wmclass()` method for more information.
`wmclass_class` :	don't use this – see the `gtk.Window.set_wmclass()` method for more information.
`override_redirect` :	if `TRUE` bypass the window manager
Returns :	the new `gtk.gdk.Window`

Creates a new `gtk.gdk.Window` of the type and class specified by *`window_type`* and *`wclass`*. The window will be a child of the specified *`parent`* and will have the specified *`width`* and *`height`*. *`event_mask`* is a bitfield specifying the events that the window will receive – see the `set_events()` method for more information. The value of *`window_type`* must be one of:

`gtk.gdk.WINDOW_ROOT`	The root window; this window has no parent, covers the entire screen, and is created by the window system.
`gtk.gdk.WINDOW_TOPLEVEL`	A toplevel window (used to implement `gtk.Window`).
`gtk.gdk.WINDOW_CHILD`	A child window (used to implement widgets e.g. `gtk.Entry`).

`gtk.gdk.WINDOW_DIALOG`	A useless/deprecated compatibility type.
`gtk.gdk.WINDOW_TEMP`	An override redirect temporary window (used to implement `gtk.Menu`).
`gtk.gdk.WINDOW_FOREIGN`	A foreign window (see the `gtk.gdk.window_foreign_new()` function).

The value of *`wclass`* must be one of:

`gtk.gdk.INPUT_OUTPUT`	A window for graphics and events.
`gtk.gdk.INPUT_ONLY`	A window for events only.

If the optional parameters are not specified the corresponding attribute values will have default values:

`x`	0
`y`	0
`visual`	the default system visual – see the `gtk.gdk.visual_get_system()` function
`colormap`	either the system `gtk.gdk.Colormap` if using the system `gtk.gdk.Visual` (see the `gtk.gdk.colormap_get_system()` function) or a new `gtk.gdk.Colormap` using *`visual`*
`cursor`	use the parent window's cursor
`override_redirect`	gtk.FALSE

Methods

gtk.gdk.Window.drag_begin

```
    def drag_begin(targets)
```

`targets` :	a list of offered targets
Returns :	a new `gtk.gdk.DragContext`

The `drag_begin()` method starts a drag operation and returns the new `gtk.gdk.DragContext` created for it. The list of targets (integer values) supported by the drag source are specified by *`targets`*.

gtk.gdk.Window.input_set_extension_events

```
    def input_set_extension_events(mask, mode)
```

`mask` :	the event mask to be used
`mode` :	the set of extension events to receive

The `input_set_extension_events()` method enables or disables the extension events specified by *`mode`* for the window when using the event mask specified by *`mask`*. The value of *`mode`* must be one of:

`gtk.gdk.EXTENSION_EVENTS_NONE`	no extension events are desired.
`gtk.gdk.EXTENSION_EVENTS_ALL`	all extension events are desired.
`gtk.gdk.EXTENSION_EVENTS_CURSOR`	extension events are desired only if a cursor will be displayed for the device.

gtk.gdk.Window.property_get

```
    def property_get(property, type=0, pdelete=FALSE)
```

`property` :	the property to get
`type` :	the type of property to get or not specified if any type of property data is acceptable.
`pdelete` :	if `TRUE`, delete the property after retrieving the data.
Returns :	a tuple containing the actual property type, the data format and the data

The `property_get()` method returns a tuple containing the actual property type (as a `gtk.gdk.Atom`), the format and the data of the specified *`property`* with the specified *`type`*. The value of *`type`* may not be be specified in which case it will be 0 to match any type of property. the returned data will be a string if the data format is 8; a list of integers if the data format is 16; or, a list of `gtk.gdk.Atom` objects or integers if the data format is 32. If *`property`* cannot be found `None` is returned. *`property`* and *`type`* (if specified) must be a string or a `gtk.gdk.Atom`.

gtk.gdk.Window.property_change

```
    def property_change(property, type, format, mode, data)
```

`property` :	the property to change
`type` :	the new type of the property. If *`mode`* is `gtk.gdk.PROP_MODE_PREPEND` or `gtk.gdk.PROP_MODE_APPEND`, then this must match the existing type or an error will occur.
`format` :	the new format for the property. If *`mode`* is `gtk.gdk.PROP_MODE_PREPEND` or `gtk.gdk.PROP_MODE_APPEND`, then this must match the existing format or an error will occur.
`mode` :	a value describing how the new data is to be combined with the current data.
`data` :	the data for the property

The `property_change()` method changes the contents of the specified *`property`* to the specified *`data`* with the specified *`type`* and *`format`*. The value of *`mode`* must be one of:

`gtk.gdk.PROP_MODE_REPLACE`	The new data replaces the existing data.
`gtk.gdk.PROP_MODE_PREPEND`	The new data is prepended to the existing data.
`gtk.gdk.PROP_MODE_APPEND`	The new data is appended to the existing data.

which describes how the new data is to be combined with the existing property data.The value of *`format`* must be 8, 16 or 32. *`property`* and *`type`* must be a string or a `gtk.gdk.Atom`.

gtk.gdk.Window.property_delete

```
    def property_delete(property)
```

`property` :	the property to delete

The `property_delete()` method deletes the specified *`property`* from the window. *`property`* must be a string or a gtk.gdk.Atom.

gtk.gdk.Window.selection_convert

```
    def selection_convert(selection, target, time)
```

`selection` :	the selection to retrieve
`target` :	the target form of *`selection`*
`time` :	the timestamp to use when retrieving *`selection`*. The selection owner may refuse the request if it did not own the selection at the time indicated by the timestamp.

The `selection_convert()` method converts the specified *`selection`* to the specified *`form`*.

gtk.gdk.Window.set_keep_above

```
    def set_keep_above(setting)
```

`setting` :	xif `TRUE` keep the window above other windows

Note

This method is available in PyGTK 2.4 and above.

The `set_keep_above`() method sets the "keep−above" setting to the value of *`setting`*. If *`setting`* is `TRUE` the window must be kept above other windows. If the window is already above, then this method does nothing.

On X11, asks the window manager to keep the window above, if the window manager supports this operation. Not all window managers support this, and some deliberately ignore it or don't have a concept of "keep above", but most standard window managers do.

gtk.gdk.Window.set_keep_below

```
    def set_keep_below(setting)
```

`setting` :	if `TRUE`, keep the window below other windows

Note

This method is available in PyGTK 2.4 and above.

The `set_keep_below`() method sets the "keep−below" setting to the value of *`setting`*. If *`setting`* is `TRUE` the window must be kept below other windows. If the window was already below, then this method does nothing.

On X11, asks the window manager to keep the window below, if the window manager supports this operation. Not all window managers support this, and some deliberately ignore it or don't have a concept of "keep below" but most standard window managers do.

gtk.gdk.Window.destroy

```
    def destroy()
```

The `destroy`() method destroys the window (destroys the server−side resource associated with the window). All children of the window are also destroyed. There's normally no need to use this method since windows are automatically destroyed when their reference count reaches 0.

gtk.gdk.Window.get_window_type

```
    def get_window_type()
```

Returns :	the type of window

The `get_window_type`() method returns the type of the window:

`gtk.gdk.WINDOW_ROOT`	the root window; this window has no parent, covers the entire screen, and is created by the window system
`gtk.gdk.WINDOW_TOPLEVEL`	a toplevel window (used to implement `gtk.Window`)
`gtk.gdk.WINDOW_CHILD`	a child window (used to implement e.g. `gtk.Button`)

`gtk.gdk.WINDOW_DIALOG`	a useless and deprecated compatibility type
`gtk.gdk.WINDOW_TEMP`	an override redirect temporary window (used to implement `gtk.Menu`)
`gtk.gdk.WINDOW_FOREIGN`	a foreign window (see the `gtk.gdk.window_foreign_new()` function)

gtk.gdk.Window.show

```
    def show()
```

The `show()` method maps the window so it's visible on−screen and also raises it to the top of the window stack (moves the window to the front of the Z−order). This method is opposite to the `hide()` method. When implementing a `gtk.Widget`, you should call this method on the widget's `gtk.gdk.Window` as part of the "map" method.

gtk.gdk.Window.hide

```
    def hide()
```

The `hide()` method withdraws toplevel windows, so they will no longer be known to the window manager and for all windows, unmaps them, so they won't be displayed. This is normally done automatically as part of the `gtk.Widget.hide()` method.

gtk.gdk.Window.withdraw

```
    def withdraw()
```

The `withdraw()` method withdraws the window (unmaps it and asks the window manager to forget about it). This is normally done automatically by the `gtk.Widget.hide()` method called on a `gtk.Window`.

gtk.gdk.Window.move

```
    def move(x, y)
```

x :	the X coordinate relative to the window's parent
y :	the Y coordinate relative to the window's parent

The `move()` method repositions the window to the location specified by *x* and *y* relative to its parent window. For toplevel windows, window managers may ignore or modify the move. You should probably use the `gtk.Window.move()` method on a `gtk.Window` widget anyway, instead of using this method. For child windows, the move will reliably succeed. If you're also planning to resize the window, use the `move_resize()` method to both move and resize simultaneously, for a nicer visual effect.

gtk.gdk.Window.resize

```
    def resize(width, height)
```

width :	the new width of the window
height :	the new height of the window

The `resize()` method resizes the window to the specified *width* and *height*. For toplevel windows, this method asks the window manager to resize the window. However, the window manager may not allow the resize. You should use the `gtk.Window.resize()` method instead of this low−level method. Windows may not be resized smaller than 1x1. If you're also planning to move the window, use the `move_resize()` method to both move and resize simultaneously, for a nicer visual effect.

gtk.gdk.Window.move_resize

```
def move_resize(x, y, width, height)
```

x :	the new X position relative to the window's parent
y :	the new Y position relative to the window's parent
width :	the new width
height :	the new height

The `move_resize()` method repositions the window to the location specified by *x* and *y* with the size specified by *width* and *height*. This method is equivalent to calling the `move()` and `resize()` methods, except that both operations are performed at once, avoiding strange visual effects. (i.e. the user may be able to see the window first move, then resize, if you don't use the `move_resize()` method.)

gtk.gdk.Window.reparent

```
def reparent(new_parent, x, y)
```

new_parent :	the new parent `gtk.gdk.Window` to move the window into
x :	the X location inside the new parent
y :	the Y location inside the new parent

The `reparent()` method reparents the window into the `gtk.gdk.Window` specified *new_parent*. The window being reparented will be unmapped as a side effect.

gtk.gdk.Window.clear

```
def clear()
```

The `clear()` method clears an entire the window to the background color or background pixmap.

gtk.gdk.Window.clear_area

```
def clear_area(x, y, width, height)
```

x :	the X coordinate of the rectangle to clear
y :	the Y coordinate of the rectangle to clear
width :	the width of the rectangle to clear
height :	the height of the rectangle to clear

The `clear_area()` method clears the area (specified by *x*, *y*, *width* and *height*) of the window to the background color or background pixmap.

gtk.gdk.Window.clear_area_e

```
def clear_area_e(x, y, width, height)
```

x :	the X coordinate of the rectangle to clear
y :	the Y coordinate of the rectangle to clear
width :	the width of the rectangle to clear
height :	the height of the rectangle to clear

The `clear_area_e()` method is like the `clear_area()`, but also generates an expose event for the cleared area.

gtk.gdk.Window.raise_

```
    def raise_()
```

The `raise_()` method raises the window to the top of the Z–order (stacking order), so that other windows with the same parent window appear below the window. If the window is a toplevel, the window manager may choose to deny the request to move the window in the Z–order. Therefore, the `raise_()` method only requests the restack, it does not guarantee it.

Note

This method is called `raise()` in the C API, but was renamed `raise_()` since `raise` is a reserved Python keyword.

gtk.gdk.Window.lower

```
    def lower()
```

The lower() method lowers the window to the bottom of the Z–order (stacking order), so that other windows with the same parent window appear above the window. If the window is a toplevel, the window manager may choose to deny the request to move the window in the Z–order. Therefore, the `lower()` only requests the restack, it does not guarantee it. Note that the `show()` method raises the window again, so don't call this method before calling the `show()` method to avoid duplication.

gtk.gdk.Window.focus

```
    def focus(timestamp=0L)
```

`timestamp` :	the timestamp of the event triggering the window focus

The `focus()` method sets keyboard focus to the window. If the window is not on–screen this will not work. In most cases, the `gtk.Window.present()` method should be used on a `gtk.Window`, rather than calling this method.

gtk.gdk.Window.set_user_data

```
    def set_user_data(user_data)
```

`user_data` :	a `gtk.Widget`

Note

This method is available in PyGTK 2.4 and above.

The `set_user_data()` method stores the underlying GTK+ widget of the PyGTK widget that is specified by *`user_data`* as the user data of the window. In general GTK+ stores the widget that owns a `gtk.gdk.Window` as user data on a `gtk.Window`. So, custom widget implementations in `PyGTK` should use this method to provide that capability. If `GTK+` receives an event for a `gtk.gdk.Window`, and the user data for the window is set, `GTK+` will assume the user data is a `gtk.Widget`, and forward the event to that widget.

In PyGTK 2.4 and above this method will raise the TypeError exception if *`user_data`* is not a `gtk.Widget`.

Note

This method is deprecated for any use other than the above. To set other user data on a `gtk.gdk.Window` use the `gobject.GObject.set_data()` method instead.

gtk.gdk.Window.get_user_data

```
def get_user_data()
```

Returns :	the user data set on the window

Note

This method is available in PyGTK 2.4 and above.

The `get_user_data`() method returns the PyGTK widget that was set as the user data of the window using the `set_user_data()` method. This method raises the ValueError exception if the user data is not set or is not a PyGTK object.

gtk.gdk.Window.set_override_redirect

```
def set_override_redirect(override_redirect)
```

`override_redirect` :	if `TRUE` the window should be override redirect

The `set_override_redirect()` method sets the "override redirect" attribute on the window to the value specified by *`override_redirect`*. If *`override_redirect`* is `TRUE` the window is not under the control of the window manager. This means it won't have a titlebar, won't be minimizable, etc. – it will be entirely under the control of the application. The window manager can't see the override redirect window at all. Override redirect should only be used for short−lived temporary windows, such as popup menus. `gtk.Menu` uses an override redirect window in its implementation, for example. This method does not work on MS Windows.

gtk.gdk.Window.add_filter

```
def add_filter(function, data=None)
```

`function` :	a function
`data` :	data to pass to *`function`*

Note

This method is available in PyGTK 2.2 and above.

The `add_filter()` method adds an event filter function specified by *`function`* to the window, allowing you to intercept events before they reach GDK. This is a low−level operation and makes it easy to break GDK and/or GTK+, so you have to know what you're doing. Once added there is no way to remove a filter function. The function signature is:

```
def function(event, user_data)
```

where *`event`* is a `gtk.gdk.Event` and *`user_data`* is `data`. If `data` is not specified then *`user_data`* is not passed to *`function`*.

`function` should return one of the following values:

`gtk.gdk.FILTER_CONTINUE`	the event was not handled, continue processing.
`gtk.gdk.FILTER_TRANSLATE`	the native event was translated into a GDK event and stored in the event that was passed in.
`gtk.gdk.FILTER_REMOVE`	the event was handled, terminate processing.

gtk.gdk.Window.scroll

```
    def scroll(dx, dy)
```

dx :	the amount to scroll in the X direction
dy :	the amount to scroll in the Y direction

The `scroll()` method scrolls the contents of the window, both pixels and children, by the horizontal and vertical amounts specified by *dx* and *dy* respectively. The window itself does not move. Portions of the window that the scroll operation brings in from off−screen areas are invalidated. The invalidated region may be bigger than what would strictly be necessary. (For X11, a minimum area will be invalidated if the window has no subwindows, or if the edges of the window's parent do not extend beyond the edges of the window. In other cases, a multi−step process is used to scroll the window which may produce temporary visual artifacts and unnecessary invalidations.)

gtk.gdk.Window.shape_combine_mask

```
    def shape_combine_mask(shape_mask, offset_x, offset_y)
```

shape_mask :	the shape bitmap mask
offset_x :	the X position of shape mask with respect to the window
offset_y :	the Y position of shape mask with respect to the window

The `shape_combine_mask()` method applies the bitmap mask specified by *shape_mask* to the window at the location specified by *x* and *y*. Pixels in the window corresponding to set bits in the *shape_mask* will be visible; pixels in the window corresponding to unset bits in the *shape_mask* will be transparent. This method provides a non−rectangular window. If *shape_mask* is `None`, the shape mask will be unset, and the *x*/*y* parameters are not used.

On the X11 platform, this uses an X server extension which is widely available on most common platforms, but not available on very old X servers, and occasionally the implementation will be buggy. On servers without the shape extension, this function will do nothing.

gtk.gdk.Window.set_child_shapes

```
    def set_child_shapes()
```

The `set_child_shapes()` method sets the shape mask of the window to the union of shape masks for all children of the window, ignoring the shape mask of the window itself. Contrast this method with the `merge_child_shapes()` method that includes the shape mask of the window in the masks to be merged.

gtk.gdk.Window.merge_child_shapes

```
    def merge_child_shapes()
```

The `merge_child_shapes()` method merges the shape masks for any child windows into the shape mask for the window. i.e. the union of all masks for the window and its children will become the new mask for the window. See the `shape_combine_mask()` method. This method is distinct from the `set_child_shapes()` method because it includes the window's shape mask in the set of shapes to be merged.

gtk.gdk.Window.is_visible

```
    def is_visible()
```

Returns :	`TRUE` if the window is mapped

The `is_visible`() method returns `TRUE` if the window has been mapped (with the `show()` method.

gtk.gdk.Window.is_viewable

```
    def is_viewable()
```

Returns :	`TRUE` if the window is viewable

The `is_viewable`() method returns `TRUE` if the window and all its ancestors are mapped. (This is not necessarily "viewable" in the X sense, since we only check as far as we have `gtk.gdk.Window` parents, not to the root window.)

gtk.gdk.Window.get_state

```
    def get_state()
```

Returns :	the window state bitfield

The `get_state`() method returns the bitwise `OR` of the currently active window state flags:

`gtk.gdk.WINDOW_STATE_WITHDRAWN`	The window is not shown.
`gtk.gdk.WINDOW_STATE_ICONIFIED`	The window is minimized.
`gtk.gdk.WINDOW_STATE_MAXIMIZED`	The window is maximized.
`gtk.gdk.WINDOW_STATE_STICKY`	The window is sticky.
`GDK_WINDOW_STATE_FULLSCREEN`	the window is maximized without decorations. Available in PyGTK 2.2 and above.
`GDK_WINDOW_STATE_ABOVE`	the window is kept above other windows. Available in PyGTK 2.4 and above.
`GDK_WINDOW_STATE_BELOW`	the window is kept below other windows. Available in PyGTK 2.4 and above.

gtk.gdk.Window.set_static_gravities

```
    def set_static_gravities(use_static)
```

`use_static` :	if `TRUE` turn on static gravity
Returns :	`TRUE` if the server supports static gravity

The `set_static_gravities`() method sets the bit gravity of the given window to the value specified by *`use_static`*. If *`use_static`* is `TRUE` the window uses static gravity and all children get static subwindow gravity as well. This method returns `TRUE` if the window system server supports static gravity.

gtk.gdk.Window.set_type_hint

```
    def set_type_hint(hint)
```

hint :	a hint of the function this window will have

The `set_type_hint()` method provides the specified *`hint`* to the window manager about the functionality of a window. The window manager can use this information when determining the decoration and behavior of the window. The hint must be set before the window is mapped. The value of hint must be one of:

`gtk.gdk.WINDOW_TYPE_HINT_NORMAL`	A normal toplevel window.
`gtk.gdk.WINDOW_TYPE_HINT_DIALOG`	A dialog window.
`gtk.gdk.WINDOW_TYPE_HINT_MENU`	A window used to implement a menu.
`gtk.gdk.WINDOW_TYPE_HINT_TOOLBAR`	A window used to implement a toolbar.

gtk.gdk.Window.set_modal_hint

```
    def set_modal_hint(modal)
```

modal :	if `TRUE` the window is modal.

The `set_modal_hint()` method sets the window's modal hint to the value specified by *`modal`*. If *`modal`* is `TRUE` the window is modal. The window manager can use this information to handle modal windows in a special way which usually means that the window gets all the input for the application effectively blocking input to other windows in the application. . You should only use this on windows for which you have previously called the `set_transient_for()` method

gtk.gdk.Window.set_skip_taskbar_hint

```
    def set_skip_taskbar_hint(modal)
```

skip_taskbar :	if `TRUE` skip the taskbar.

Note

This method is available in PyGTK 2.2 and above.

The `set_skip_taskbar_hint()` method sets the "skip_taskbar" setting to the value specified by *`skips_taskbar`*. If *`skips_taskbar`* is `TRUE` the window should **not** appear in a task list or window list. If the window's semantic type as specified with the `set_type_hint()` method already fully describes the window, this method should **not** be called in addition; instead you should allow the window to be treated according to standard policy for its semantic type.

gtk.gdk.Window.set_skip_pager_hint

```
    def set_skip_pager_hint(skips_pager)
```

skips_pager :	if `TRUE` skip the pager

Note

This method is available in PyGTK 2.2 and above.

The `set_skip_pager_hint()` method sets the "skip_pager" setting to the value of skips_pager. If skips_pager is TRUE the window should **not** appear in a pager (a workspace switcher, or other desktop utility program that displays a small thumbnail representation of the windows on the desktop). If the window's semantic type as specified with `set_type_hint()` already fully describes the window, this method should **not** be called in addition, instead you should allow the window to be treated according to standard policy for its semantic type.

gtk.gdk.Window.set_geometry_hints

```
def set_geometry_hints(min_width=-1, min_height=-1, max_width=-1, max_height=-1, base_width=
```

min_width :	minimum width of window or −1 to use requisition
min_height :	minimum height of window or −1 to use requisition
max_width :	maximum width of window or −1 to use requisition
max_height :	maximum height of window or −1 to use requisition
base_width :	allowed window widths are base_width + width_inc * N (where N is any integer) or −1
base_height :	allowed window widths are base_height + height_inc * N (where N is any integer) or −1
width_inc :	width resize increment
height_inc :	height resize increment
min_aspect :	minimum width/height ratio
max_aspect :	maximum width/height ratio

Note

This method is available in PyGTK 2.2 and above.

The `set_geometry_hints()` method sets the geometry hints for the window.

This method provides hints to the windowing system about acceptable sizes for a toplevel window. The purpose of this is to constrain user resizing, but the windowing system will typically (but is not required to) also constrain the current size of the window to the provided values and constrain programmatic resizing via gdk_window_resize() or gdk_window_move_resize().

Note that on X11, this effect has no effect on windows of type GDK_WINDOW_TEMP or windows where override_redirect has been turned on via the `set_override_redirect()` method since these windows are not resizable by the user.

gtk.gdk.Window.begin_paint_rect

```
def begin_paint_rect(rectangle)
```

rectangle :	the rectangle you intend to draw to

The `begin_paint_rect()` method indicates that you are beginning the process of redrawing the area specified by *`rectangle`*. A backing store (off−screen buffer) large enough to contain *`rectangle`* will be created. The backing store will be initialized with the background color or background pixmap for window. Then, all drawing operations performed on the window will be diverted to the backing store. When you call the `end_paint()` method, the backing store will be copied to the window, making it visible on−screen. Only the part of window contained in region will be modified; that is, drawing operations are clipped to *`rectangle`*. The net result of all this is to remove flicker, because the user sees the finished product appear all at once when you call the `end_paint()` method. If you draw to window directly without calling the `begin_paint_rect()` method, the user may see flicker as individual drawing operations are performed in sequence. The clipping and background initializing features of the `begin_paint_rect()` are conveniences for the programmer, so you can avoid doing that work yourself.

gtk.gdk.Window.end_paint

```
def end_paint()
```

The `end_paint()` method indicates that the backing store created by the most recent call to the `begin_paint_rect()` method should be copied on−screen and deleted, leaving the next−most−recent backing store or no backing store at all as the active paint region. It is an error to call this function without a matching call to the `begin_paint_rect()` method first.

gtk.gdk.Window.set_title

```
def set_title(title)
```

`title` :	the new title of the window

The `set_title()` method sets the title of a toplevel window, to the string specified by *`title`*. If you haven't explicitly set the icon name for the window (using the `set_icon_name()` method), the icon name will be set to *`title`* as well. *`title`* must be in UTF−8 encoding (as with all user−readable strings in `PyGTK`).

gtk.gdk.Window.set_role

```
def set_role(role)
```

`role` :	a string indicating its role

The `set_role()` method sets the string specified by *`role`* as the window's role. When using `PyGTK`, you should generally use the `gtk.Window.set_role()` method instead of this low−level function. The window manager and session manager use a window's role to distinguish it from other kinds of window in the same application. When an application is restarted after being saved in a previous session, all windows with the same title and role are treated as interchangeable. So if you have two windows with the same title that should be distinguished for session management purposes, you should set the role on those windows. It doesn't matter what string you use for the role, as long as you have a different role for each non−interchangeable kind of window.

gtk.gdk.Window.set_transient_for

```
def set_transient_for(leader)
```

`leader` :	another `gtk.gdk.Window`

The `set_transient_for()` method indicates to the window manager that the window is a transient dialog associated with the application window *`leader`*. This allows the window manager to do things like center the window on *`leader`* and keep the window above *`leader`*. See the `gtk.Window.set_transient_for()` method if you're using a `gtk.Window` or `gtk.Dialog`.

gtk.gdk.Window.set_background

```
def set_background(color)
```

`color` :	an allocated `gtk.gdk.Color`

The `set_background()` method sets the background `gtk.gdk.Color` of the window to the value specified by *`color`*. (However, when using `PyGTK`, set the background of a widget with the `gtk.Widget.modify_bg()` method from an application − or the `gtk.Style.set_background()` method from a custom widget implementation.) The *`color`* must be allocated Also see the `set_back_pixmap()` method.

gtk.gdk.Window.set_back_pixmap

```
def set_back_pixmap(pixmap, parent_relative)
```

pixmap :	a `gtk.gdk.Pixmap`, or `None`
parent_relative :	if `TRUE` the tiling origin is at the origin of the window's parent

The `set_back_pixmap()` method sets the background pixmap of the window to the value specified by *`pixmap`* A background pixmap will be tiled, positioning the first tile at the origin of the window, or if *`parent_relative`* is `TRUE`, the tiling will be done based on the origin of the parent window (useful to align tiles in a parent with tiles in a child). If *`pixmap`* is `None` the window will have no background which means it will never have its background filled by the windowing system. Instead the window will contain whatever pixels were already in the corresponding area of the display. The windowing system will normally fill a window with its background when the window is obscured then exposed, and when you call the `clear()` method.

gtk.gdk.Window.set_cursor

```
def set_cursor(cursor)
```

cursor :	a `gtk.gdk.Cursor` or `None`

The `set_cursor()` method sets the mouse pointer for a `gtk.gdk.Window`. Use either the gtk.gdk.Cursor() or gtk.gdk.Cursor() constructors to create the cursor. To make the cursor invisible, use the gtk.gdk.Cursor() constructor to create a cursor with no pixels in it. Passing `None` for the *`cursor`* argument to the `set_cursor()` method means that the window will use the cursor of its parent window. Most windows should use this default.

gtk.gdk.Window.get_geometry

```
def get_geometry()
```

Returns :	a 5–tuple containing the X and Y coordinate of the location of the window relative to its parent and the width and height of the window and the bit depth of the window.

The `get_geometry()` method returns a 5–tuple containing the window's location and size (x, y, width, height) and the bit depth of the window. The X and Y coordinates returned are relative to the parent window of the window, which for toplevels usually means relative to the window decorations (titlebar, etc.) rather than relative to the root window (screen–size background window).

On the X11 platform, the geometry is obtained from the X server, so reflects the latest position of the window; this may be out–of–sync with the position of the window delivered in the most–recently–processed `GdkEventConfigure`. the `get_position()` method in contrast gets the position from the most recent configure event.

gtk.gdk.Window.get_position

```
def get_position()
```

Returns :	a 2–tuple containing the X and Y coordinates of the window location.

The `get_position()` returns a 2–tuple containing the position of the window as reported in the most–recently–processed `GdkEventConfigure`. By comparison with the `get_geometry()` method that queries the X server for the current window position, regardless of what events have been received or processed. The position coordinates are relative to the window's parent window.

gtk.gdk.Window.get_origin

```
def get_origin()
```

Returns :	a 2−tuple containing the X and Y coordinates of the window

The `get_origin()` method returns a 2−tuple containing the x and y coordinates of the position of a window in root window coordinates. (Compare this method with the `get_position()` and `get_geometry()` methods that return the position of a window relative to its parent window.)

gtk.gdk.Window.get_root_origin

```
def get_root_origin()
```

Returns :	a 2−tuple containing the X and Y coordinates of the window frame position

The `get_root_origin()` method returns a 2−tuple containing the top−left corner of the window manager frame in root window coordinates.

gtk.gdk.Window.get_frame_extents

```
def get_frame_extents()
```

Returns :	a `gtk.gdk.Rectangle` specifying the bounding box of the window frame

The `get_frame_extents()` method returns a `gtk.gdk.Rectangle` specifying the bounding box of the window, including window manager titlebar/borders if any. The frame position is given in root window coordinates. To get the position of the window itself (rather than the frame) in root window coordinates, use the `get_origin()` method.

gtk.gdk.Window.get_pointer

```
def get_pointer()
```

Returns :	a 3−tuple containing the X and Y coordinates of the mouse pointer and the modifier mask

The `get_pointer()` method returns a 3−tuple containing the coordinates of the mouse pointer location relative to the window and the modifier state. The modifier state is a combination of the following:

`gtk.gdk.SHIFT_MASK`	The Shift key.
`gtk.gdk.LOCK_MASK`	A Lock key (depending on the modifier mapping of the X server this may either be CapsLock or ShiftLock).
`gtk.gdk.CONTROL_MASK`	The Control key.
`gtk.gdk.MOD1_MASK`	The fourth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier, but normally it is the Alt key).
`gtk.gdk.MOD2_MASK`	The fifth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
`gtk.gdk.MOD3_MASK`	The sixth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
`gtk.gdk.MOD4_MASK`	The seventh modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
`gtk.gdk.MOD5_MASK`	The eighth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
`gtk.gdk.BUTTON1_MASK`	The first mouse button.
`gtk.gdk.BUTTON2_MASK`	The second mouse button.

`gtk.gdk.BUTTON3_MASK`	The third mouse button.
`gtk.gdk.BUTTON4_MASK`	The fourth mouse button.
`gtk.gdk.BUTTON5_MASK`	The fifth mouse button.
`gtk.gdk.RELEASE_MASK`	Differentiates between (keyval, modifiers) pairs from key press and release events.
`gtk.gdk.MODIFIER_MASK`	all of the above

gtk.gdk.Window.get_parent

```
    def get_parent()
```

Returns : the parent `gtk.gdk.Window` of the window

The `get_parent()` method returns the parent of the window as set when the `gtk.gdk.Window` was created or when the `reparent()` method was called.

gtk.gdk.Window.get_toplevel

```
    def get_toplevel()
```

Returns : the toplevel `gtk.gdk.Window` containing the window

The `get_toplevel()` method returns the toplevel `gtk.gdk.Window` that's an ancestor of the window.

gtk.gdk.Window.get_children

```
    def get_children()
```

Returns : the list of child windows inside the window

The `get_children()` method returns the list of children `gtk.gdk.Window` objects of the window. This method only returns children created via `PyGTK`, so for example it's useless when used with the root window; it only returns windows an application created itself.

gtk.gdk.Window.get_events

```
    def get_events()
```

Returns : the event mask for the window

The `get_events()` method returns the event mask for the window. See the `set_events()` method for more detail.

gtk.gdk.Window.set_events

```
    def set_events(event_mask)
```

`event_mask` : the event mask for the window

The `set_events()` method sets the event mask to the value specified by event_mask for the window. The event mask determines which events will be reported for the window. For example, an event mask including `gtk.gdk.BUTTON_PRESS_MASK` means the window should report button press events. The event mask is the bitwise OR of the following:

- `gtk.gdk.EXPOSURE_MASK`
- `gtk.gdk.POINTER_MOTION_MASK`

- `gtk.gdk.POINTER_MOTION_HINT_MASK`
- `gtk.gdk.BUTTON_MOTION_MASK`
- `gtk.gdk.BUTTON1_MOTION_MASK`
- `gtk.gdk.BUTTON2_MOTION_MASK`
- `gtk.gdk.BUTTON3_MOTION_MASK`
- `gtk.gdk.BUTTON_PRESS_MASK`
- `gtk.gdk.BUTTON_RELEASE_MASK`
- `gtk.gdk.KEY_PRESS_MASK`
- `gtk.gdk.KEY_RELEASE_MASK`
- `gtk.gdk.ENTER_NOTIFY_MASK`
- `gtk.gdk.LEAVE_NOTIFY_MASK`
- `gtk.gdk.FOCUS_CHANGE_MASK`
- `gtk.gdk.STRUCTURE_MASK`
- `gtk.gdk.PROPERTY_CHANGE_MASK`
- `gtk.gdk.VISIBILITY_NOTIFY_MASK`
- `gtk.gdk.PROXIMITY_IN_MASK`
- `gtk.gdk.PROXIMITY_OUT_MASK`
- `gtk.gdk.SUBSTRUCTURE_MASK`
- `gtk.gdk.SCROLL_MASK`
- `gtk.gdk.ALL_EVENTS_MASK`

`gtk.gdk.ALL_EVENTS_MASK` is a combination of all the event masks.

gtk.gdk.Window.set_icon_list

```
    def set_icon_list(pixbufs)
```

`pixbufs` :	a list (or tuple) containing pixbufs, of different sizes.

Note

This method is available in PyGTK 2.2 and above.

The `set_icon_list()` method sets the list of icons for the window. pixbufs is a list or tuple containing `gtk.gdk.Pixbuf` objects to be used as the icon images. One of these will be used to represent the window when it has been iconified. The icon is usually shown in an icon box or some sort of task bar. Which icon size is shown depends on the window manager. The window manager can scale the icon but setting several size icons can give better image quality since the window manager may only need to scale the icon by a small amount or not at all.

gtk.gdk.Window.set_icon

```
    def set_icon(icon_window, pixmap, mask)
```

`icon_window` :	a `gtk.gdk.Window` to use for the icon
`pixmap` :	a `gtk.gdk.Pixmap` to use as the icon
`mask` :	a 1−bit pixmap (`GdkBitmap`) to use as mask for *`pixmap`*

The `set_icon()` method sets the icon of the window as a `gtk.gdk.Pixmap` (specified by *`pixmap`*) or `gtk.gdk.Window` specified by *`icon_window`*). Investigate the `gtk.window_set_default_icon_list()`() function first, and then the `gtk.Window.set_icon_list()` and `gtk.Window.set_icon()` methods. If those don't meet your needs, look at the `set_icon_list()` method. Only if all those are too high−level do you want to fall back to the `set_icon()`.

gtk.gdk.Window.set_icon_name

```
    def set_icon_name(name)
```

name :	the name of the window while iconified (minimized)

The `set_icon_name()` method sets the name of the window when it is iconified to the value of *name*. Windows may have a name used while minimized, distinct from the name they display in their titlebar. Most of the time this is a bad idea from a user interface standpoint. But you can set such a name with this method, if you like.

gtk.gdk.Window.set_group

```
    def set_group(leader)
```

leader :	the group leader `gtk.gdk.Window`

The `set_group()` method sets the group leader for the window to the `gtk.gdk.Window` specified by *leader*. By default, the group leader for all toplevel windows is set to a global window implicitly created by `PyGTK`. With this method you can override this default. The group leader window allows the window manager to distinguish all windows that belong to a single application. It may for example allow users to minimize or unminimize all windows belonging to an application at once. You should only set a non–default group window if your application pretends to be multiple applications. The group leader window may not be changed after a window has been mapped (with the show() method for example).

gtk.gdk.Window.get_group

```
    def get_group()
```

Returns :	the group leader `gtk.gdk.Window` for the window

Note

This method is available in PyGTK 2.4 and above.

The `get_group()` method returns the group leader `gtk.gdk.Window` for the window. See the `set_group()` method for more information.

gtk.gdk.Window.set_decorations

```
    def set_decorations(decorations)
```

decorations :	the decoration hint mask

The `set_decorations()` method sets the specified decorations for the window. "Decorations" are the features the window manager adds to a toplevel `gtk.gdk.Window`. This method sets the traditional Motif window manager hints that tell the window manager which decorations you would like your window to have. Usually you should use the `gtk.Window.set_decorated()` method on a `gtk.Window` instead of using this method directly. The value of *decorations* is the logical OR of the following:

`gtk.gdk.DECOR_ALL`	All decorations should be applied.
`gtk.gdk.DECOR_BORDER`	A frame should be drawn around the window.
`gtk.gdk.DECOR_RESIZEH`	The frame should have resize handles.
`gtk.gdk.DECOR_TITLE`	A titlebar should be placed above the window.
`gtk.gdk.DECOR_MENU`	A button for opening a menu should be included.
`gtk.gdk.DECOR_MINIMIZE`	A minimize button should be included.

`gtk.gdk.DECOR_MAXIMIZE`	A maximize button should be included.

If `gtk.gdk.DECOR_ALL` is included in the mask, the other bits indicate which decorations should be turned off. If `gtk.gdk.DECOR_ALL` is not included, then the other bits indicate which decorations should be turned on. Most window managers honor a decorations hint of 0 to disable all decorations, but very few honor all possible combinations of bits.

gtk.gdk.Window.get_decorations

```
    def get_decorations()
```

Returns :	the window decorations

The `get_decorations()` method returns the decorations set on the window with the `set_decorations` method.

gtk.gdk.Window.set_functions

```
    def set_functions(functions)
```

`functions` :	the bitmask of operations to allow on the window

The set_functions() method sets the traditional Motif window manager hint for which operations the window manager should allow on a toplevel window. However, few window managers do anything reliable or interesting with this hint. Many ignore it entirely. The *`functions`* argument is the logical OR of the following:

`gtk.gdk.FUNC_ALL`	All functions should be offered.
`gtk.gdk.FUNC_RESIZE`	The window should be resizable.
`gtk.gdk.FUNC_MOVE`	The window should be movable.
`gtk.gdk.FUNC_MINIMIZE`	The window should be minimizable.
`gtk.gdk.FUNC_MAXIMIZE`	The window should be maximizable.
`gtk.gdk.FUNC_CLOSE`	The window should be closeable.

If the bitmask includes `gtk.gdk.FUNC_ALL`, then the other bits indicate which functions to disable; if it doesn't include `gtk.gdk.FUNC_ALL`, it indicates which functions to enable.

gtk.gdk.Window.iconify

```
    def iconify()
```

The `iconify()` method asks the window manager to iconify (minimize) the window. The window manager may choose to ignore the request, but normally will honor it. Using the `gtk.Window.iconify()` method is preferred, if you have a `gtk.Window` widget.

gtk.gdk.Window.deiconify

```
    def deiconify()
```

The `deiconify()` method asks the window manager to deiconify (unminimize) the window. On X11 the window manager may choose to ignore the request to deiconify. Using the `gtk.Window.deiconify()` method is preferred. Or better yet, use the `gtk.Window.present()`, which raises the window, focuses it, unminimizes it, and puts it on the current desktop.

gtk.gdk.Window.stick

```
    def stick()
```

The `stick()` method "pins" a window such that it's on all workspaces and does not scroll with viewports, for window managers that have scrollable viewports. (When using a `gtk.Window`, the `gtk.Window.stick()` method may be more useful.) On the X11 platform, this method depends on window manager support, so may have no effect with many window managers. However, `PyGTK` will do the best it can to convince the window manager to stick the window. For window managers that don't support this operation, there's nothing you can do to force it to happen.

gtk.gdk.Window.unstick

```
    def unstick()
```

The `unstick()` method reverses the effect of the `stick()` method. See the `stick()` and `gtk.Window.unstick()` methods for more information.

gtk.gdk.Window.maximize

```
    def maximize()
```

The `maximize()` method asks the window manager to maximize the window, if the window manager supports this operation. Not all window managers support this, and some deliberately ignore it or don't have a concept of "maximized"; so you can't rely on the maximization actually happening. But it will happen with most standard window managers. If the window was already maximized, then this method does nothing.

gtk.gdk.Window.unmaximize

```
    def unmaximize()
```

The `unmaximize()` method asks the window manager to unmaximize the window, if the window manager supports this operation. Not all window managers support this, and some deliberately ignore it or don't have a concept of "maximized"; so you can't rely on the unmaximization actually happening. But it will happen with most standard window managers. If the window wasn't maximized, then this method does nothing.

gtk.gdk.Window.fullscreen

```
    def fullscreen()
```

Note

This method is available in PyGTK 2.2 and above.

The `fullscreen()` method moves the window into fullscreen mode. This means the window covers the entire screen and is above any panels or task bars.

If the window was already fullscreen, then this method does nothing.

On X11, asks the window manager to put the window in a fullscreen state, if the window manager supports this operation. Not all window managers support this, and some deliberately ignore it or don't have a concept of "fullscreen" but most standard window managers do.

gtk.gdk.Window.unfullscreen

```
    def unfullscreen()
```

Note

This method is available in PyGTK 2.2 and above.

The `unfullscreen`() method moves the window out of fullscreen mode. If the window was not fullscreen, does nothing.

On X11, asks the window manager to move the window out of the fullscreen state, if the window manager supports this operation. Not all window managers support this, and some deliberately ignore it or don't have a concept of "fullscreen" but most standard window managers do.

gtk.gdk.Window.register_dnd

```
    def register_dnd()
```

The `register_dnd`() method registers the window as a potential drop destination.

gtk.gdk.Window.begin_resize_drag

```
    def begin_resize_drag(edge, button, root_x, root_y, timestamp)
```

`edge` :	the edge or corner from which the drag is started
`button` :	the mouse button being used to drag
`root_x` :	the root window X coordinate of the mouse click that began the drag
`root_y` :	the root window Y coordinate of the mouse click that began the drag
`timestamp` :	the timestamp of the mouse click that began the drag (use the `gtk.gdk.Event.get_time()` method)

The `begin_resize_drag`() method begins a window resize operation (for a toplevel `gtk.gdk.Window`) from the specified *`edge`* using the specified *`button`* starting at the location specified by *`root_x`* and *`root_y`*. The value of edge must be one of:

`gtk.gdk.WINDOW_EDGE_NORTH_WEST`	The top left corner.
`gtk.gdk.WINDOW_EDGE_NORTH`	The top edge.
`gtk.gdk.WINDOW_EDGE_NORTH_EAST`	The top right corner.
`gtk.gdk.WINDOW_EDGE_WEST`	The left edge.
`gtk.gdk.WINDOW_EDGE_EAST`	The right edge.
`gtk.gdk.WINDOW_EDGE_SOUTH_WEST`	The lower left corner.
`gtk.gdk.WINDOW_EDGE_SOUTH`	The lower edge.
`gtk.gdk.WINDOW_EDGE_SOUTH_EAST`	The lower right corner.

You might use this method to implement a "window resize grip," for example; in fact the `gtk.Statusbar` uses it. The method works best with window managers that support the Extended Window Manager Hints spec (see http://www.freedesktop.org), but has a fallback implementation for other window managers.

gtk.gdk.Window.begin_move_drag

```
    def begin_move_drag(button, root_x, root_y, timestamp)
```

button :	the button being used to drag
root_x :	the root window X coordinate of the mouse click that began the drag
root_y :	the root window Y coordinate of the mouse click that began the drag
timestamp :	the timestamp of the mouse click that began the drag

The `begin_move_drag()` method begins a window move operation (for a toplevel window) using the specified *button* starting at the location specified by *root_x* and *root_y*. You might use this method to implement a "window move grip," for example. The method works best with window managers that support the Extended Window Manager Hints spec (see http://www.freedesktop.org), but has a fallback implementation for other window managers.

gtk.gdk.Window.invalidate_rect

```
def invalidate_rect(rect, invalidate_children)
```

rect :	the rectangle to invalidate
invalidate_children :	if `TRUE` invalidate child `gtk.gdk.Window` objects

The `invalidate_rect()` method invalidates the rectangular region specified by rect. If *invalidate_children* is `TRUE` the child `gtk.gdk.Window` object of the window are also invalidated.

gtk.gdk.Window.freeze_updates

```
def freeze_updates()
```

The `freeze_updates()` method temporarily freezes the window such that it won't receive expose events. The window will begin receiving expose events again when the `thaw_updates()` method is called. If the `freeze_updates()` method has been called more than once, the `thaw_updates()` method must be called an equal number of times to begin processing exposes.

gtk.gdk.Window.thaw_updates

```
def thaw_updates()
```

The `thaw_updates()` method thaws a window frozen with the `freeze_updates()` method.

gtk.gdk.Window.process_updates

```
def process_updates(update_children)
```

update_children :	if `TRUE` process updates for child windows

The `process_updates()` method sends one or more expose events to the window. The areas in each expose event will cover the entire update area for the window (see the `invalidate_rect()` method for details). Normally `PyGTK` calls the `gtk.gdk.window_process_all_updates()` function on your behalf, so there's no need to call this method unless you want to force expose events to be delivered immediately and synchronously (vs. the usual case, where `PyGTK` delivers them in an idle handler). Occasionally this is useful to produce nicer scrolling behavior, for example.

gtk.gdk.Window.set_accept_focus

```
def set_accept_focus(accept_focus)
```

accept_focus :	if `TRUE`, the window should receive input focus

Note

This method is available in PyGTK 2.4 and above.

The `set_accept_focus()` method sets the "accept_focus setting to the value of *`accept_focus`*. If *`accept_focus`* is `TRUE` the window will accept focus; if `FALSE` hints to the desktop environment that the window doesn't want to receive input focus.

On X, it is the responsibility of the window manager to interpret this hint. ICCCM−compliant window manager usually respect it.

gtk.gdk.Window.enable_synchronized_configure

```
    def enable_synchronized_configure()
```

Note

This method is available in PyGTK 2.6 and above.

The `enable_synchronized_configure()` method indicates that the application will cooperate with the window system in synchronizing the window repaint with the window manager during resizing operations. After an application calls this method, it must call the `configure_finished()` method every time it has finished all processing associated with a set of Configure events. Toplevel GTK+ windows automatically use this protocol.

On X, calling this function makes window participate in the _NET_WM_SYNC_REQUEST window manager protocol.

gtk.gdk.Window.configure_finished

```
    def configure_finished()
```

Note

This method is available in PyGTK 2.6 and above.

The `configure_finished()` method signals to the window system that the application has finished handling all the Configure events it has received. Window Managers can use this to better synchronize the frame repaint with the application. GTK+ applications will automatically call this function when appropriate.

This function can only be called if the `enable_synchronized_configure()` method was called previously.

gtk.gdk.Window.set_focus_on_map

```
    def set_focus_on_map(focus_on_map)
```

`focus_on_map` :	if `TRUE` the window should receive input focus when mapped.

Note

This method is available in PyGTK 2.6 and above.

The `set_focus_on_map()` method sets the a hint for the desktop environment to the value specified by *`focus_on_map`*. If *`focus_on_map`* is `TRUE` the window sets a hint for the desktop environment indicating that it would like to receive input focus when mapped.

On X, it is the responsibility of the window manager to interpret this hint. Window managers following the freedesktop.org window manager extension specification should respect it.

Functions

gtk.gdk.window_foreign_new

```
def gtk.gdk.window_foreign_new(anid)
```

`anid` :	a native window system ID
Returns :	the new `gtk.gdk.Window` wrapper for the native window or `None` if the window has been destroyed.

The `gtk.gdk.window_foreign_new()` function wraps a native window specified by *`anid`* for the default display in a `gtk.gdk.Window`. This may fail if the window has been destroyed. For example in the X Window System backend, a native window handle is an Xlib `XID`.

gtk.gdk.window_foreign_new_for_display

```
def gtk.gdk.window_foreign_new_for_display(display, anid)
```

`display` :	a `gtk.gdk.Display`
`anid` :	a native window system ID
Returns :	the new `gtk.gdk.Window` wrapper for the native window or `None` if the window has been destroyed.

Note

This function is available in PyGTK 2.2 and above.

The `gtk.gdk.window_foreign_new_for_display()` function wraps a native window specified by *`anid`* for the `gtk.gdk.Display` specified by *`display`* in a `gtk.gdk.Window`. This may fail if the window has been destroyed. For example in the X Window System backend, a native window handle is an Xlib `XID`.

gtk.gdk.get_default_root_window

```
def gtk.gdk.get_default_root_window()
```

Returns :	the default root `gtk.gdk.Window`

The `gtk.gdk.get_default_root_window()` function returns the root `gtk.gdk.Window` (the parent window that all other windows are inside) for the default display and screen.

gtk.gdk.window_get_toplevels

```
def gtk.gdk.window_get_toplevels()
```

Returns :	a list containing the toplevel `gtk.gdk.Window` object

The `gtk.gdk.window_get_toplevels()` function returns a list of all toplevel windows known to `PyGTK` on the default screen. A toplevel window is a child of the root window (see the gtk.gdk.get_default_root_window() function).

gtk.gdk.window_lookup

```
    def gtk.gdk.window_lookup(anid)
```

`anid` :	a native window system ID
Returns :	the `gtk.gdk.Window` wrapper for the native window or `None` if there is none.

The `gtk.gdk.window_lookup()` function looks up the `gtk.gdk.Window` that wraps the native window handle specified by *`anid`*. For example in the X Window System backend, a native window handle is an Xlib `XID`.

gtk.gdk.window_lookup_for_display

```
    def gtk.gdk.window_lookup_for_display(display, anid)
```

`display` :	a `gtk.gdk.Display`
`anid` :	a native window system ID
Returns :	the `gtk.gdk.Window` wrapper for the native window or `None` if there is none.

Note

This function is available in PyGTK 2.2 and above.

The `gtk.gdk.window_lookup_for_display()` function looks up the `gtk.gdk.Window` that wraps the native window handle specified by *`anid`* for the `gtk.gdk.Display` specified by *`display`*. For example in the X Window System backend, a native window handle is an Xlib `XID`.

gtk.gdk.window_process_all_updates

```
    def gtk.gdk.window_process_all_updates()
```

The `gtk.gdk.process_all_updates()` function calls the `process_updates()` method for each `gtk.gdk.Window` in the application.

gtk.gdk.gdk_window_set_debug_updates

```
    def gtk.gdk.gdk_window_set_debug_updates(setting)
```

`setting` :	if `TRUE` enable update debugging

The `gtk.gdk.gdk_set_debug_updates()` function sets the update debugging flag to the value of *`setting`*. If *`setting`* is `TRUE`, update debugging is enabled. With update debugging enabled, calls to the `invalidate_rect()` method clear the invalidated rectangle of the screen to a noticeable color, and `PyGTK` pauses for a short time before sending exposes to windows during the `process_updates()` method. The net effect is that you can see the invalid region for each window and watch redraws as they occur. This allows you to diagnose inefficiencies in your application.In essence, because the `GDK` rendering model prevents all flicker, if you are redrawing the same region 400 times you may never notice, aside from noticing a speed problem. Enabling update debugging causes `PyGTK` to flicker slowly and noticeably, so you can see exactly what's being redrawn when, in what order.

The `--gtk-debug=updates` command line option passed to `PyGTK` programs enables this debug option at application startup time. That's usually more useful than calling `gtk.gdk.gdk_set_debug_updates()` yourself, though you might want to use this function to enable updates sometime after application startup time.

gtk.gdk.window_at_pointer

```
    def gtk.gdk.window_at_pointer()
```

Returns : a 3 tuple containing the `gtk.gdk.Window` and the pointer location in the window or `None`.

Note

This function is available in PyGTK 2.4 and above.

The `gtk.gdk.window_at_pointer()` function returns a 3−tuple containing the `gtk.gdk.Window` underneath the mouse pointer, and the location of the pointer in the window. This function returns `None` if the window under the mouse pointer is not known to GDK (if the window belongs to another application and a `gtk.gdk.Window` hasn't been created for it with the `gtk.gdk.window_foreign_new()` function)

Note

For multi−head−aware widgets or applications use the `gtk.gdk.Display.get_window_at_pointer()` method instead.

Prev	Up	Next
gtk.gdk.Visual	Home	gtk.gdk Functions

gtk.glade.XML

Prev **The gtk.glade Class Reference** Next

gtk.glade.XML

gtk.glade.XML Allows dynamic loading of user interfaces from XML descriptions

Synopsis

```
class gtk.glade.XML(gobject.GObject):
    gtk.glade.XML(fname, root="", domain="", typedict={})
    def signal_connect(handler_name, func)
    def signal_autoconnect(dict)
    def get_widget(name)
    def get_widget_prefix(name)
    def relative_file(filename)
Functions

    def gtk.glade.xml_new_from_buffer(buffer, size, root="", domain="", typedict={})
    def gtk.glade.get_widget_name(widget)
    def gtk.glade.get_widget_tree(widget)
```

Ancestry

```
+-- gobject.GObject
      +-- gtk.glade.XML
```

Description

This object represents an `instantiation' of an XML interface description. When one of these objects is created, the XML file is read, and the interface is created. The `gtk.glade.XML` object then provides an interface for accessing the widgets in the interface by the names assigned to them inside the XML description.

The `gtk.glade.XML` object can also be used to connect handlers to the named signals in the description. Libglade also provides an interface by which it can look up the signal handler names in the program's symbol table and automatically connect as many handlers up as it can that way.

Constructor

```
gtk.glade.XML(fname, root="", domain="", typedict={})
```

`fname` :	the XML file name
`root` :	the widget node in *`fname`* to start building from (or `""`)
`domain` :	the translation domain for the XML file (or `""` for default)
`typedict` :	A dictionary used to lookup types (or `{}` for default)
Returns :	a new `gtk.glade.XML` object

Creates a new `gtk.glade.XML` object (and the corresponding widgets) from the XML file specified by *`fname`*. Optionally it will only build the interface from the widget node specified by *`root`* (if it is not `""`). This feature is useful if you only want to build say a toolbar or menu from the XML file, but not the window it is embedded in. Note also that the XML parse tree is cached to speed up creating another `gtk.glade.XML` object for the same file.

Methods

gtk.glade.XML.signal_connect

```
def signal_connect(handler_name, func)
```

`handler_name` :	the signal handler name
`func` :	the signal handler function

In the glade interface descriptions, signal handlers are specified for widgets by name. The `signal_connect()` method allows you to connect a callback specified by *`func`* to all signals in the `gtk.glade.XML` file with the signal handler name specified by *`handler_name`*.

gtk.glade.XML.signal_autoconnect

```
def signal_connect(dict)
```

`dict` :	a mapping or an instance

The `signal_autoconnect()` method is a variation of the gtk.glade.XML.signal_connect method. It uses Python's introspective features to look at the keys (if *`dict`* is a mapping) or attributes (if *`dict`* is an instance) and tries to match them with the signal handler names given in the interface description. The callbacks referenced by each matched key or attribute are connected to their matching signals. The argument is called `dict` due to compatibility reasons since originally only the mapping interface was supported. The instance variant was introduced in PyGTK 2.0.

gtk.glade.XML.get_widget

```
def get_widget(name)
```

name :	the name of the widget
Returns :	the widget matching the name or `None`

The `get_widget()` method returns a reference to the `gtk.Widget` specified by *`name`* in the interface description. `None` is returned, if *`name`* doesn't specify a widget in the interface.

gtk.glade.XML.get_widget_prefix

```
def get_widget_prefix(name)
```

name :	the prefix the widget names or `""` for all widgets
Returns :	A list of widgets that match *`name`* as the start of their name or `None`

The `get_widget_prefix()` method returns a list of interface `gtk.Widget` objects that have names prefixed by *`name`*. `None` is returned if no interface widget names match the prefix *`name`*.

gtk.glade.XML.relative_file

```
def relative_file(filename)
```

filename :	a relative file pathname
Returns :	the absolute file pathname

The `relative_file()` method resolves the relative pathname specified by *`filename`*, using the directory of the XML file as a base. If *`filename`* contains an absolute pathname, then the original file name is returned.

Functions

gtk.glade.xml_new_from_buffer

```
def gtk.image_new_from_buffer(buffer, size, root="", domain="", typedict={})
```

buffer :	the string containing the XML buffer
size :	size of the string
root :	the widget node in fname to start building from (or `""`)
domain :	the translation domain for the XML file (or `""` for default)
typedict :	A dictionary used to lookup types (or `{}` for default)
Returns :	a new `gtk.glade.XML` object.

The `gtk.glade.xml_new_from_buffer()` function creates a new `gtk.glade.XML` object (and the corresponding widgets) from the string specified by *`buffer`*. Optionally it will only build the interface from the widget node specified by *`root`* (if it is not `""`). This feature is useful if you only want to build say a toolbar or menu from the XML document, but not the window it is embedded in.

gtk.glade.get_widget_name

```
def gtk.glade.get_widget_name(widget)
```

widget :	a `gtk.Widget`

Returns :	the name of the widget

The `gtk.glade.get_widget_name()` function returns the name of the `gtk.Widget` specified by *`widget`* that was generated by a `gtk.glade.XML` object.

gtk.glade.get_widget_tree

```
    def gtk.glade.get_widget_tree(widget)
```

`widget` :	a `gtk.Widget`
Returns :	the `gtk.glade.XML` object that built *`widget`*

This `gtk.glade.get_widget_tree()` function is used to get the `gtk.glade.XML` object that built the `gtk.Widget` specified by *`widget`*.

gobject.GObject

gobject.GObject the base class

Synopsis

```
class gobject.GObject:
    def get_property(property_name)
    def set_property(property_name, value)
    def freeze_notify()
    def notify(property_name)
    def thaw_notify()
    def get_data(key)
    def set_data(key, data)
    def connect(detailed_signal, handler)
    def connect_after(detailed_signal, handler)
    def connect_object(detailed_signal, handler)
    def connect_object_after(detailed_signal, handler)
    def disconnect(handler_id)
    def handler_disconnect(handler_id)
    def handler_is_connected(handler_id)
    def handler_block(handler_id)
    def handler_unblock(handler_id)
    def emit(detailed_signal)
    def stop_emission(detailed_signal)
    def emit_stop_by_name(detailed_signal)
    def chain()
```

Ancestry

```
+-- gobject.GObject
```

Description

The GObject class is the base class providing the common attributes and methods for the PyGTK classes. The GObject class is not a user interface widget class.

The GObject class provides the signal management methods, the object property access methods and the object data management methods.

Attributes

"__doc__"	Read	The documentation for the object type. Uses "__gdoc__" if no specific documentation set.
"__gdoc__"	Read	The generated documentation for the underlying GObject type.
"__gtype__"	Read	The underlying GObject type.
"__grefcount__"	Read	The reference count for the underlying GObject.

Methods

gobject.GObject.get_property

```
    def get_property(property_name)
```

property_name :	a string containing the property name for the GObject
Returns :	a Python object containing the value of the property

The `get_property()` method returns the value of the property specified by *property_name* or None if there is no value associated with the property.

The `TypeError` exception is raised if the property name is not registered with the object class.

gobject.GObject.set_property

```
    def set_property(property_name, value)
```

property_name :	a string containing the property name
value :	a Python object containing the property value to be set

The `set_property()` method sets the property specified by *property_name* to the specified *value*.

The `TypeError` exception is raised if the property name is not registered with the object class or if the value specified could not be converted to the property type.

gobject.GObject.freeze_notify

```
    def freeze_notify()
```

The `freeze_notify()` method freezes the object's property−changed notification queue so that "notify" signals are blocked until the `thaw_notify()` method is called.

gobject.GObject.notify

```
def notify(property_name)
```

property_name :	a string containing a property name

The `notify`() method causes the "notify" signal for the property specified by *`property_name`* to be emitted.

gobject.GObject.thaw_notify

```
def thaw_notify()
```

The `thaw_notify`() method thaws the object's property−changed notification queue so that "notify" signals are emitted.

gobject.GObject.get_data

```
def get_data(key)
```

key :	a string used as the key
Returns :	a Python object containing the associated data

The `get_data`() method returns the Python object associated with the specified *`key`* or None if there is no data associated with the *`key`* or if there is no key associated with the object.

gobject.GObject.set_data

```
def set_data(key, data)
```

key :	a string used as a key
data :	a Python object that is the value to be associated with the key

The `set_data`() method associates the specified Python object (*`data`*) with *`key`*.

gobject.GObject.connect

```
def connect(detailed_signal, handler, ...)
```

detailed_signal :	a string containing the signal name
handler :	a Python function or method object.
... :	additional optional parameters
Returns :	an integer identifier

The `connect()` method adds a function or method (*`handler`*)to the end of the list of signal handlers for the named *`detailed_signal`* but before the default class signal handler. An optional set of parameters may be specified after the *`handler`* parameter. These will all be passed to the signal handler when invoked.

For example if a function handler was connected to a signal using:

```
handler_id = object.connect("signal_name", handler, arg1, arg2, arg3)
```

The handler should be defined as:

```
def handler(object, arg1, arg2, arg3):
```

A method handler connected to a signal using:

```
handler_id = object.connect("signal_name", self.handler, arg1, arg2)
```

requires an additional argument when defined:

```
def handler(self, object, arg1, arg2)
```

A `TypeError` exception is raised if *`detailed_signal`* identifies a signal name that is not associated with the object.

gobject.GObject.connect_after

```
def connect_after(detailed_signal, handler, ...)
```

`detailed_signal` :	a string containing the signal name
`handler` :	a Python function or method object
`...` :	additional optional parameters
Returns :	an integer handler identifier

The `connect_after()` method is similar to the `connect()` method except that the *`handler`* is added to the signal handler list after the default class signal handler. Otherwise the details of *`handler`* definition and invocation are the same.

gobject.GObject.connect_object

```
def connect_object(detailed_signal, handler, gobject)
```

`detailed_signal` :	a string containing the signal name
`handler` :	a Python function or method object
`gobject` :	a GObject
Returns :	an integer handler identifier

The `connect_object()` method is the same as the `connect()` method except that the *`handler`* is invoked with the specified *`gobject`* in place of the object invoking the `connect_object()` method. For example, a call with a function handler:

```
handler_id = object("signal_name", handler, gobject)
```

will cause the *`handler`* to be invoked as:

```
handler(gobject)
```

Likewise a method handler will be invoked as:

```
self.handler(gobject)
```

This can be helpful in invoking PyGTK widget methods that require no arguments except the widget itself (e.g. `widget.destroy()`) by using the class method as the handler. For example, a Button "clicked" signal can be set up to invoke the Window `destroy()` method as:

```
handler_id = button.connect_object("clicked", Window.destroy, window)
```

When the button is clicked the handler is invoked as:

```
Window.destroy(window)
```

which is the same as:

```
window.destroy()
```

Additional arguments may be passed to the handler as with the `connect()` method handler invocations.

gobject.GObject.connect_object_after

```
def connect_object_after(detailed_signal, handler)
```

detailed_signal :	a string containing the signal name
handler :	a Python function or method object
gobject :	a GObject
Returns :	an integer handler identifier

The `connect_object_after()` method is similar to the `connect_object()` method except that the *handler* is added to the signal handler list after the default class signal handler. Otherwise the details of *handler* definition and invocation are the same.

gobject.GObject.disconnect

```
def disconnect(handler_id)
```

handler_id :	an integer handler identifier

The `disconnect()` method removes the signal handler with the specified *handler_id* from the list of signal handlers for the object.

gobject.GObject.handler_disconnect

```
def handler_disconnect(handler_id)
```

handler_id :	an integer handler identifier

The `handler_disconnect()` method removes the signal handler with the specified *handler_id* from the list of signal handlers for the object.

gobject.GObject.handler_is_connected

```
def handler_is_connected(handler_id)
```

handler_id :	an integer handler identifier
Returns :	`TRUE` if the signal handler is connected to the object.

The `handler_is_connected()` method returns `TRUE` if the signal handler with the specified *handler_id* is connected to the object.

gobject.GObject.handler_block

```
def handler_block(handler_id)
```

handler_id :	an integer handler identifier

The `handler_block()` method blocks the signal handler with the specified *handler_id* from being invoked until it is unblocked.

gobject.GObject.handler_unblock

```
def handler_unblock(handler_id)
```

handler_id :	an integer handler identifier

The `handler_unblock()` method unblocks the signal handler with the specified *handler_id* thereby allowing it to be invoked when the associated signal is emitted.

gobject.GObject.emit

```
  def emit(detailed_signal, ...)
```

detailed_signal :	a string containing the signal name
... :	additional parameters
Returns :	a PyObject*

The `emit()` method causes the object to emit the signal specified by *detailed_signal*. The additional parameters must match the number and type of the required signal handler parameters. In most cases no additional parameters are needed. for example:

```
  button.emit("clicked")
```

is all that is required to emit the "clicked" signal for a button. The most common case requiring additional parameters occurs when emitting an event signal; for example:

```
  button.emit("button_press_event", event)
```

gobject.GObject.stop_emission

```
  def stop_emission(detailed_signal)
```

detailed_signal :	a string containing the signal name

The `stop_emission()` method stops the current emission of the signal specified by *detailed_signal*. Any signal handlers in the list still to be run will not be invoked.

gobject.GObject.emit_stop_by_name

```
  def emit_stop_by_name(detailed_signal)
```

detailed_signal :	a string containing the signal name

The `emit_stop_by_name()` method stops the current emission of the signal specified by *detailed_signal*. Any signal handlers in the list still to be run will not be invoked.

gobject.GObject.chain

```
  def chain(...)
```

... :	additional parameters
Returns :	a Python object

The chain() method does something.

Prev Up Next
The gobject Class Reference Home gobject.GBoxed

gobject.GBoxed
Prev **The gobject Class Reference** Next

gobject.GBoxed

gobject.GBoxed — an object containing an opaque chunk of data

Synopsis

```
class gobject.GBoxed:
    def copy()
```

Ancestry

```
+-- gobject.GBoxed
```

Description

`gobject.GBoxed` is an abstract base class that encapsulates an opaque chunk of data to provide an object–oriented interface and a type that is registered with the `GLIB` type system. A boxed type is registered with functions that provide for the copying and freeing of the underlying data structure – this allows PyGTK to encapsulate these as Python objects.

Methods

gobject.GBoxed.copy

```
   def copy()
```

Returns :	a copy of the `gobject.GBoxed` object

The `copy()` method makes and returns a copy of the boxed object.

Prev	Up	Next
gobject.GObject	Home	gobject.GPointer

gobject.GInterface

Prev **The gobject Class Reference** Next

gobject.GInterface

gobject.GInterface an object representing a GInterface

Synopsis

```
class gobject.GInterface:
```

Ancestry

```
+-- gobject.GInterface
```

Description

`gobject.GInterface` is an abstract base class that encapsulates a GInterface.

Prev Up Next

gobject.GPointer

gobject.GPointer — an object containing a completely opaque chunk of data

Synopsis

```
class gobject.GPointer:
```

Ancestry

```
+-- gobject.GPointer
```

Description

`gobject.GPointer` is an abstract base class that encapsulates an opaque chunk of data and registers it with the `GLIB` type system. A pointer type has no methods and generic ways of copying and freeing the data. It shouldn't be used in PyGTK.

Prev Up Next

gobject.GBoxed Home gobject.GInterface

gobject.MainContext

gobject.MainContext — an object representing a set of event sources to be handled in a `gobject.MainLoop`.

Synopsis

```
class gobject.MainContext:
    gobject.MainContext()
    def iteration(may_block)
    def pending()
```

Ancestry

```
+-- gobject.MainContext
```

Description

A `gobject.MainContext` represents a set of event sources that can be run in a single thread. File descriptors (plain files, pipes or sockets) and timeouts are the standard event sources for `GTK` and `PyGTK` though others can be added. Each event source is assigned a priority. The default priority, `gobject.PRIORITY_DEFAULT`, is 0. Values less than 0 denote higher priorities. Values greater than 0 denote lower priorities. Events from high priority sources are always processed before events from lower priority sources. Single iterations of a `gobject.MainContext` can be run with the `iteration()` method.

Constructor

```
gobject.MainContext()
```

Returns :	a new `gobject.MainContext` object.

Creates a new `gobject.MainContext` object.

Methods

gobject.MainContext.iteration

```
def iteration()
```

`may_block` :	if `TRUE` the call may block waiting for an event.
Returns :	`TRUE` if events were dispatched.

The `iteration()` method runs a single iteration. This involves:

- checking to see if any associated event sources are ready to be processed;
- then if no events sources are ready and *`may_block`* is `TRUE`, waiting for a source to become ready;
- and finally, dispatching the highest priority events sources that are ready

Note that even when *`may_block`* is `TRUE`, it is still possible for `iteration()` to return `FALSE`, since the the wait may be interrupted for other reasons than an event source becoming ready.

gobject.MainContext.pending

```
def pending()
```

Returns :	`TRUE` if events are pending.

The `pending()` method checks if any associated sources have pending events.

Prev	Up	Next
gobject.GInterface	Home	gobject.MainLoop

gobject.MainLoop

Prev **The gobject Class Reference** Next

gobject.MainLoop

gobject.MainLoop — an object representing the main event loop of a PyGTK application.

Synopsis

```
class gobject.MainLoop:
    gobject.MainLoop(context=None, is_running=0)
    def get_context()
    def is_running()
    def quit()
    def run()
```

Ancestry

```
+-- gobject.MainLoop
```

Description

`gobject.MainLoop` represents a main event loop. A `gobject.MainLoop` is created with the gobject.MainLoop() constructor. After adding the initial event sources, the `run()` method is called. This continuously checks for new events from each of the event sources and dispatches them. Finally, the processing of an event from one of the sources leads to a call to the `quit()` method to exit the main loop, and the `run()` method returns.

It is possible to create new instances of `gobject.MainLoop` recursively. This is often used in `PyGTK` applications when showing modal dialog boxes. Note that event sources are associated with a particular `gobject.MainContext`, and will be checked and dispatched for all main loops associated with that `gobject.MainContext`.

`PyGTK` contains wrappers of some of these functions, e.g. the `gtk.main()`, `gtk.main_quit()` and `gtk.events_pending()` functions.

Constructor

```
    gobject.MainLoop(context=None, is_running=None)
```

context :	a `gobject.MainContext` or `None` to use the default context.
is_running :	if `TRUE` indicates that the loop is running. This is not very important since calling the `run()` method will set this to `TRUE` anyway.
Returns :	a new `gobject.MainLoop` object.

Creates a new `gobject.MainLoop` object.

Methods

gobject.MainLoop.get_context

```
    def get_context()
```

Returns :	the `gobject.MainContext` the mainloop is associated with

The `get_context()` method returns the `gobject.MainContext` that the mainloop was created with.

gobject.MainLoop.is_running

```
    def is_running()
```

Returns :	`TRUE` if the mainloop is currently being run.

The `is_running`() method checks to see if the mainloop is currently being run via the `run()` method.

gobject.MainLoop.quit

```
    def quit()
```

The `quit`() method stops the mainloop from running. Any subsequent calls to the `run()` method will return immediately.

gobject.MainLoop.run

```
    def run()
```

The `run`() method runs a mainloop until the `quit()` method is called. If this is called for the thread of the loop's `gobject.MainContext`, it will process events from the loop, otherwise it will simply wait.

Prev	Up	Next
gobject.MainContext	Home	gobject Functions

gtk.AboutDialog

Prev	**The gtk Class Reference**	Next

gtk.AboutDialog

gtk.AboutDialog   popup window displaying information about an application (new in PyGTK 2.6)

Synopsis

```
class gtk.AboutDialog(gtk.Dialog):
    gtk.AboutDialog()
    def get_name()
    def set_name(name)
    def get_version()
    def set_version(version)
    def get_copyright()
    def set_copyright(copyright)
    def get_comments()
    def set_comments(comments)
    def get_license()
    def set_license(license)
    def get_website()
    def set_website(website)
    def get_website_label()
    def set_website_label(website_label)
    def get_authors()
    def set_authors(authors)
    def get_documenters()
    def set_documenters(documenters)
    def get_artists()
    def set_artists(artists)
    def get_translator_credits()
    def set_translator_credits(translator_credits)
    def get_logo()
    def set_logo(logo)
    def get_logo_icon_name()
    def set_logo_icon_name(icon_name)
Functions

    def gtk.about_dialog_set_email_hook(func, data)
    def gtk.about_dialog_set_url_hook(func, data)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Window
            +-- gtk.Dialog
              +--gtk.AboutDialog
```

Properties

"artists"	Read−Write	The list of people who have contributed artwork to the program.
"authors"	Read−Write	The list of authors of the program.
"comments"	Read−Write	A string containing comments about the program.
"copyright"	Read−Write	A string containing copyright information for the program.
"documenters"	Read−Write	The list of people documenting the program.
"license"	Read−Write	A string containing the license of the program.
"logo"	Read−Write	A logo for the about box. If this is not set, it defaults to the default window icon list..
"logo−icon−name"	Read−Write	The name of an icon to use as the logo for the about box.
"name"	Read−Write	The name of the program. If this is not set, it defaults to g_get_application_name().
"translator−credits"	Read−Write	Credits to the translators. This string should be marked as translatable.
"version"	Read−Write	A string containing the version of the program.
"website"	Read−Write	The URL for the link to the website of the program.
"website−label"	Read−Write	The label for the link to the website of the program. If this is not set, it defaults to the URL.

Style Properties

"link−color"	Read	The color of hyperlinks.

Description

Note

This widget is available in PyGTK 2.6 and above.

The `gtk.AboutDialog` offers a simple way to display information about a program like its logo, name, copyright, website and license. It is also possible to give credits to the authors, documenters, translators and artists who have worked on the program. An about dialog is typically opened when the user selects the HelpAbout menu. All parts of the dialog are optional.

Constructor

```
gtk.AboutDialog()
```

Returns :	a new `gtk.AboutDialog`

Note

This constructor is available in PyGTK 2.6 and above.

Creates a new gtk.AboutDialog with default property values.

Methods

gtk.AboutDialog.get_name

```
def get_name()
```

Returns :	The program name or `None`.

Note

This method is available in PyGTK 2.6 and above.

The `get_name`() method returns the value of the "name" property which is the program name displayed in the about dialog. `get_name`() returns `None` if the name is not set.

gtk.AboutDialog.set_name

```
def set_name(name)
```

`name` :	a string containing the program name or `None` to use the default name.

Note

This method is available in PyGTK 2.6 and above.

The `set_name`() method sets the "name" property to the string contained in *`name`*. The "name" property is used as the program name in the about dialog. If *`name`* is `None`, it defaults to g_get_application_name().

gtk.AboutDialog.get_version

```
def get_version()
```

Returns :	The version string or `None`.

Note

This method is available in PyGTK 2.6 and above.

The `get_version()` method returns the string contained in the "version" property.

gtk.AboutDialog.set_version

```
def set_version(version)
```

`version` :	the version string or `None`

Note

This method is available in PyGTK 2.6 and above.

The `set_version()` method sets the "version" property to the string in *`version`*.

gtk.AboutDialog.get_copyright

```
def get_copyright()
```

Returns :	the copyright string or `None`.

Note

This method is available in PyGTK 2.6 and above.

The `get_copyright()` method returns the string contained in the "copyright" property.

gtk.AboutDialog.set_copyright

```
def set_copyright(copyright)
```

`copyright` :	a string containing the copyright notice or `None`.

Note

This method is available in PyGTK 2.6 and above.

The `set_copyright()` method sets the "copyright" property to the string contained in *`copyright`*. If *`copyright`* is `None`, the copyright notice is hidden.

gtk.AboutDialog.get_comments

```
def get_comments()
```

Returns :	a string containing the comments or `None`

Note

This method is available in PyGTK 2.6 and above.

The `get_comments()` method returns the string contained in the "comments" property.

gtk.AboutDialog.set_comments

```
    def set_comments(comments)
```

comments :	a string containing the comments or `None`.

Note

This method is available in PyGTK 2.6 and above.

The `set_comments()` method sets the "comments" property to the string contained in *comments*. If *comments* is `None` the comments label is hidden.

gtk.AboutDialog.get_license

```
    def get_license()
```

Returns :	a string containing the license information or `None`

Note

This method is available in PyGTK 2.6 and above.

The `get_license()` method returns the string contained in the "license" property.

gtk.AboutDialog.set_license

```
    def set_license(license)
```

license :	

Note

This method is available in PyGTK 2.6 and above.

The `set_license()` method sets the "license" property to the string contained in *license*. If *license* is `None` the license button is hidden.

gtk.AboutDialog.get_website

```
    def get_website()
```

Returns :	a string containing the website URL or `None`

Note

This method is available in PyGTK 2.6 and above.

The `get_website()` method returns the string contained in the "website" property. The website should be a URL.

gtk.AboutDialog.set_website

```
    def set_website(website)
```

website :	a string containing the URL of the program's website or `None`

Note

This method is available in PyGTK 2.6 and above.

The `set_website()` method sets the "website" property to the string contained in *`website`*. The string should be a valid URL.

gtk.AboutDialog.get_website_label

```
def get_website_label()
```

Returns :	a string containing the website link label

Note

This method is available in PyGTK 2.6 and above.

The `get_website_label()` method returns the contents of the "website_label" property. The website label is used if the url hook has been set using the `gtk.about_dialog_set_url_hook()` function.

gtk.AboutDialog.set_website_label

```
def set_website_label(website_label)
```

`website_label` :	

Note

This method is available in PyGTK 2.6 and above.

The `set_website_label()` method sets the "website_label" property to the string contained in *`website_label`* if the url hook has been set using the `gtk.about_dialog_set_url_hook()` function.

gtk.AboutDialog.get_authors

```
def get_authors()
```

Returns :	a list containing the names of the program authors

Note

This method is available in PyGTK 2.6 and above.

The `get_authors()` method returns the contents of the "authors" property. The "authors" property contains a list of the names of the authors of the program.

gtk.AboutDialog.set_authors

```
def set_authors(authors)
```

`authors` :	a list containing the names of the program authors.

Note

This method is available in PyGTK 2.6 and above.

The `set_authors()` method sets the "authors" property to the list contained in *`authors`*. The author names are displayed in the authors tab of the secondary credits dialog. `set_authors()` method will show the Credits button if it is not displayed.

gtk.AboutDialog.get_documenters

```
def get_documenters()
```

Returns :	a list of the program documenters

Note

This method is available in PyGTK 2.6 and above.

The `get_documenters()` method returns the contents of the "documenters" property which contains a list of the names of the program documenters.

gtk.AboutDialog.set_documenters

```
def set_documenters(documenters)
```

`documenters` :	a list of the names of the program documenters

Note

This method is available in PyGTK 2.6 and above.

The `set_documenters()` method sets the "documenters" property to the contents of *`documenters`*. The documenter names are displayed in the documenters tab of the secondary credits dialog. `set_documenters()` method will show the Credits button if it is not displayed.

gtk.AboutDialog.get_artists

```
def get_artists()
```

Returns :	a list of the names of the program artists

Note

This method is available in PyGTK 2.6 and above.

The `get_artists()` method returns the contents of the "artists" property which contains a list of the names of the program artists.

gtk.AboutDialog.set_artists

```
def set_artists(artists)
```

`artists` :	a list of the names of the program artists

Note

This method is available in PyGTK 2.6 and above.

The `set_artists()` method sets the "artists" property to the contents of *`artists`*. The artist names are displayed in the artists tab of the secondary credits dialog. `set_artists()` method will show the Credits button if it is not displayed.

gtk.AboutDialog.get_translator_credits

```
def get_translator_credits()
```

Returns :	a string containing the translator credits

Note

This method is available in PyGTK 2.6 and above.

The `get_translator_credits()` method returns the contents of the "translator–credits" property which contains the credits for the current translation.

gtk.AboutDialog.set_translator_credits

```
def set_translator_credits(translator_credits)
```

`translator_credits` :	a string containing the current translation credits

Note

This method is available in PyGTK 2.6 and above.

The `set_translator_credits()` method sets the "translator_credits" property to the value of *`translator_credits`*. The translator credits are displayed in the translators tab of the secondary credits dialog.

The intended use for this string is to display the translator of the language which is currently used in the user interface. Using the `gettext()` function, a simple way to achieve that is to mark the string for translation.

It is a good idea to use the customary msgid "translator–credits" for this purpose, since translators will already know the purpose of that msgid, and since `gtk.AboutDialog` will detect if "translator–credits" is untranslated and hide the tab.

gtk.AboutDialog.get_logo

```
def get_logo()
```

Returns :	the `gtk.gdk.Pixbuf` used as the logo

Note

This method is available in PyGTK 2.6 and above.

The `get_logo()` method returns the contents of the "logo" property which contains a `gtk.gdk.Pixbuf` that is used as the logo.

gtk.AboutDialog.set_logo

```
    def set_logo(logo)
```

logo :	a gtk.gdk.Pixbuf to be used as the logo

Note

This method is available in PyGTK 2.6 and above.

The `set_logo`() method sets the "logo" property to the gtk.gdk.Pixbuf contained in *logo*. If *logo* is `None`, the default window icon set with the gtk.window_set_default_icon_list() function will be used.

gtk.AboutDialog.get_logo_icon_name

```
    def get_logo_icon_name()
```

Returns :	the name of the icon used as the logo

Note

This method is available in PyGTK 2.6 and above.

The `get_logo_name`() method returns the contents of the "logo–icon–name" property which contains the name of the icon used as the logo.

gtk.AboutDialog.set_logo_icon_name

```
    def set_logo_icon_name(icon_name)
```

icon_name :	the name of an icon or `None`

Note

This method is available in PyGTK 2.6 and above.

The `set_logo_icon_name`() method sets the "logo–icon–name" property to the value of *icon_name*. If *icon_name* is `None`, the default window icon set with the gtk.window_set_default_icon_list() function will be used.

Functions

gtk.about_dialog_set_email_hook

```
    def gtk.about_dialog_set_email_hook(func, data)
```

func :	a function to call when an email link is activated.
data :	data to pass to *func*

Note

This function is available in PyGTK 2.6 and above.

The `gtk.about_dialog_set_email_hook` function installs a global function (specified by *`func`*) to be called whenever the user activates an email link in an about dialog.

gtk.about_dialog_set_url_hook

```
    def gtk.about_dialog_set_url_hook(func, data)
```

`func` :	a function to call when a URL link is activated.
`data` :	data to pass to *`func`*

Note

This function is available in PyGTK 2.6 and above.

The `gtk.about_dialog_set_url_hook` function installs a global function (specified by *`func`*) to be called whenever the user activates a URL link in an about dialog.

Prev	Up	Next
The gtk Class Reference	Home	gtk.AccelGroup

gtk.AccelGroup

gtk.AccelGroup — a group of accelerators for a Window hierarchy

Synopsis

```
class gtk.AccelGroup(gobject.GObject):
    gtk.AccelGroup()
    def lock()
    def unlock()
    def connect(accel_key, accel_mods, accel_flags, callback)
    def connect_group(accel_key, accel_mods, accel_flags, callback)
    def connect_by_path(accel_path, callback)
    def disconnect_key(accel_key, accel_mods)
Functions

    def gtk.accelerator_valid(keyval, modifiers)
    def gtk.accelerator_parse(accelerator)
    def gtk.accelerator_name(accelerator_key, accelerator_mods)
    def gtk.accelerator_set_default_mod_mask(default_mod_mask)
    def gtk.accelerator_get_default_mod_mask()
    def gtk.accelerator_get_label(accelerator_key, accelerator_mods)
    def gtk.accel_map_add_entry(accel_path, accel_key, accel_mods)
    def gtk.accel_map_lookup_entry(accel_path)
    def gtk.accel_map_change_entry(accel_path, accel_key, accel_mods, replace)
    def gtk.accel_map_load(file_name)
    def gtk.accel_map_save(file_name)
    def gtk.accel_map_load_fd(fd)
    def gtk.accel_map_save_fd(fd)
    def gtk.accel_map_lock_path(accel_path)
    def gtk.accel_map_unlock_path(accel_path)
    def gtk.accel_map_add_filter(filter_pattern)
    def gtk.accel_groups_from_object(object)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.AccelGroup
```

Signal Prototypes

"accel–activate"	`def callback(`*`accelgroup, acceleratable, accel_key, accel_mods, user_param1, ...`*`)`
"accel–changed"	`def callback(`*`accelgroup, accel_key, accel_mods, closure, user_param1, ...`*`)`

Description

A `gtk.AccelGroup` object groups all the accelerators for the associated window hierarchy (either `gtk.Window` (or a descendant) or `gtk.MenuShell` (or a descendant)). Once the `gtk.AccelGroup` is associated with a window or menu (using `gtk.Window.add_accel_group()` or `gtk.Menu.set_accel_group()`), accelerators can be added to the widget or one of its children by using `gtk.Widget.add_accelerator()` . Accelerators can also be added by using a `gtk.ItemFactory`.

Note that accelerators are different from mnemonics. Accelerators are shortcuts for activating a menu item; they appear alongside the menu item they're a shortcut for. For example **Ctrl+Q** might appear alongside the Files–>Quit menu item. Mnemonics are shortcuts for GUI elements such as text entries or buttons; they appear as underlined characters. Of course, menu items can have both accelerators and mnemonics.

Constructor

```
  gtk.AccelGroup()
```

Returns :	an AccelGroup object

Creates a new `gtk.AccelGroup` object.

Methods

gtk.AccelGroup.lock

```
  def lock()
```

The `lock`() method locks the accelerator group. preventing its accelerators from being changed during runtime. Refer to `gtk.accel_map_change_entry()` about runtime accelerator changes.

If called more than once, the accelerator group remains locked until `gtk.AccelGroup.unlock()` has been called an equivalent number of times.

gtk.AccelGroup.unlock

```
  def unlock()
```

The `unlock`() method undoes the last call to `gtk.AccelGroup.lock()` for this accelerator group.

gtk.AccelGroup.connect_group

```
def connect(accel_key, accel_mods, accel_flags, callback)
```

accel_key :	key value of the accelerator
accel_mods :	modifier combination of the accelerator
accel_flags :	a flag mask to configure this accelerator
callback :	a function or method to be executed upon accelerator activation

Note

This method is available in PyGTK 2.2 as *connect*() and was changed in PyGTK 2.4 and above to *connect_group*() to avoid conflict with the gobject.GObject.connect() method.

The connect_group() method installs an accelerator in the accelerator group. When the accelerator group is being activated, the function (or method) specified by *callback* will be invoked if the accelerator key and modifier key match those specified by *accel_key* and *accel_mods*.

The value of *modifier* is a combination of the GDK Modifier Constants. *accel_flags* is a combination of gtk.ACCEL_VISIBLE and gtk.ACCEL_LOCKED.

The *callback* function is defined as:

```
def callback(accel_group, acceleratable, keyval, modifier)
```

where *accel_group* is the accelerator group, *acceleratable* is the object that the *accel_group* is attached to (e.g. a gtk.Window), *keyval* is the accelerator key and *modifier* is the key modifier. *callback* returns TRUE if the accelerator was handled by *callback*.

Note

Due to implementation details, a single function or method can only be connected to one accelerator group.

gtk.AccelGroup.connect_by_path

```
def connect_by_path(accel_path, callback)
```

accel_path :	path used for determining key and modifiers.
callback :	function or method to be executed upon accelerator activation

Note

This method is available in PyGTK 2.4 and above

The connect_by_path() method installs an accelerator in the accelerator group, using an accelerator path to look up the appropriate key and modifiers (see the function gtk.accel_map_add_entry()). When the accelerator group is being activated, the function (or method) specified by *callback* will be invoked if the *accel_key* and *accel_mods* that cause the activation match the key and modifiers for the accelerator path specified by *accel_path*.

The *callback* function is defined as:

```
def callback(accel_group, acceleratable, keyval, modifier)
```

where *accel_group* is the accelerator group, *acceleratable* is the object that the *accel_group* is attached to (e.g. a gtk.Window), *keyval* is the accelerator key and *modifier* is the key modifier. *callback* returns TRUE if the accelerator was handled by *callback*.

gtk.AccelGroup.disconnect_key

```
def disconnect_key(accel_key, accel_mods)
```

accel_key :	key value of the accelerator
accel_mods :	modifier combination of the accelerator
Returns :	`TRUE` if there was an accelerator which was removed, `FALSE` otherwise

The `disconnect()` method removes a previously installed accelerator specified by *`accel_key`* and *`accel_mods`* from the accelerator group.

Functions

gtk.accelerator_valid

```
def gtk.accelerator_valid(keyval, modifiers)
```

keyval :	a key value
modifiers :	a modifier mask
Returns :	`TRUE` if the accelerator is valid

The `gtk.accelerator_valid()` function returns `TRUE` if the specified *`keyval`* and *`modifiers`* constitute a valid keyboard accelerator. For example, the `ord('a')` keyval plus `gtk.gdk.CONTROL_MASK` is valid – this is a **Control+a** accelerator. The value of *`modifiers`* is a combination of the GDK Modifier Constants.

gtk.accelerator_parse

```
def gtk.accelerator_parse(accelerator)
```

accelerator :	a string representing an accelerator
Returns :	a 2–tuple containing the keyval and modifier mask of the accelerator

The `gtk.accelerator_parse()` function parses the specified *`accelerator`* string and returns a 2–tuple containing the keyval and modifier mask corresponding to *`accelerator`*. The format looks like "<Control>a" or "<Shift><Alt>F1" or "<Release>z" (the last one is for key release). The parser is fairly liberal and allows lower or upper case, and also abbreviations such as "<Ctl>" and "<Ctrl>". If the parse fails, the tuple values will both be 0 (zero). See the `gtk.accelerator_valid()` function for more details.

gtk.accelerator_name

```
def gtk.accelerator_name()
```

accelerator_key :	a key value
accelerator_mods :	a modifier mask
Returns :	a string representing the accelerator or `None` if not a valid accelerator

The `gtk.accelerator_name()` function converts the accelerator keyval and modifier mask (specified by *`accelerator_key`* and *`accelerator_mods`*) into a string parseable by the `gtk.accelerator_parse()` function. For example, if you pass in `ord('q')` and `gtk.gdk.CONTROL_MASK`, this function returns "<Control>q".

gtk.accelerator_set_default_mod_mask

```
    def gtk.accelerator_set_default_mod_mask(default_mod_mask)
```

`default_mod_mask` :	the new default accelerator modifier mask

The `gtk.accelerator_set_default_mod_mask()` function sets the modifiers (specified by *`default_mod_mask`*) that will be considered significant for keyboard accelerators. The default mod mask is `gtk.gdk.CONTROL_MASK` | `gtk.gdk.SHIFT_MASK` | `gtk.gdk.MOD1_MASK`, that is, **Control**, **Shift**, and **Alt**. Other modifiers will by default be ignored by `gtk.AccelGroup`. You must include at least the three default modifiers in any value you pass to this function. The default mod mask should be changed on application startup, before using any accelerator groups. The value of *`default_mod_mask`* is a combination of the GDK Modifier Constants.

gtk.accelerator_get_default_mod_mask

```
    def gtk.accelerator_get_default_mod_mask()
```

Returns :	the default accelerator modifier mask

The `gtk.accelerator_get_default_mod_mask()` function returns the default accelerator modifier mask as set by the `gtk.accelerator_set_default_mod_mask()` function. See the `gtk.accelerator_set_default_mod_mask()` function for more detail on modifier masks.

gtk.accelerator_get_label

```
    def gtk.accelerator_get_label(accelerator_key, accelerator_mods)
```

`accelerator_key` :	a key value
`accelerator_mods` :	a modifier mask
Returns :	a string representing the accelerator

Note

This function is available in PyGTK 2.6 and above.

The `gtk.accelerator_get_label()` function converts the accelerator keyval and modifier mask specified by *`accelerator_key`* and *`accelerator_mods`* respectively into a string which can be used to represent the accelerator to the user. The value of *`accelerator_mods`* is a combination of the GDK Modifier Constants.

gtk.accel_map_add_entry

```
    def gtk.accel_map_add_entry(accel_path, accel_key, accel_mods)
```

`accel_path` :	a valid accelerator path
`accel_key` :	the accelerator key
`accel_mods` :	the accelerator modifiers

Note

This function is available in PyGTK 2.4 and above.

The `gtk.accel_map_add_entry()` function registers a new accelerator specified by *`accel_key`* and *`accel_mods`* with the global accelerator map. The accelerator will be associated with the accelerator path specified by *`accel_path`*. This function should only be called once per *`accel_path`* with the canonical *accel_key* and *accel_mods* for this path. To change the accelerator during runtime programatically, use the gtk.accel_map_change_entry() function. The accelerator path must consist of "<WINDOWTYPE>/Category1/Category2/.../Action", where <WINDOWTYPE> should be a unique application−specific identifier, that corresponds to the kind of window the accelerator is being used in, e.g. "Gimp−Image", "Abiword−Document" or "Gnumeric−Settings". The Category1/.../Action portion is most appropriately chosen by the action the accelerator triggers, i.e. for accelerators on menu items, choose the item's menu path, e.g. "File/Save As", "Image/View/Zoom" or "Edit/Select All". So a full valid accelerator path may look like: "<Gimp−Toolbox>/File/Dialogs/Tool Options...".

gtk.accel_map_lookup_entry

```
def gtk.accel_map_lookup_entry(accel_path)
```

accel_path :	a valid accelerator path
Returns :	a 2−tuple containing the keyval and modifier mask corresponding to *accel_path* or `None` if not valid

The `gtk.accel_map_lookup_entry()` function returns a 2−tuple containing the keyval and modifier mask corresponding to the accelerator path specified by *accel_path* or `None` if *accel_path* is not a valid accelerator path.

The accelerator path must consist of "<WINDOWTYPE>/Category1/Category2/.../Action", where <WINDOWTYPE> should be a unique application−specific identifier, that corresponds to the kind of window the accelerator is being used in, e.g. "Gimp−Image", "Abiword−Document" or "Gnumeric−Settings". The Category1/.../Action portion is most appropriately chosen by the action the accelerator triggers, i.e. for accelerators on menu items, choose the item's menu path, e.g. "File/Save As", "Image/View/Zoom" or "Edit/Select All". So a full valid accelerator path may look like: "<Gimp−Toolbox>/File/Dialogs/Tool Options...".

gtk.accel_map_change_entry

```
def gtk.accel_map_change_entry(accel_path, accel_key, accel_mods, replace)
```

accel_path :	a valid accelerator path
accel_key :	the new accelerator key
accel_mods :	the new accelerator modifiers
replace :	if `TRUE` other accelerators may be deleted if conflicting
Returns :	`TRUE` if the accelerator could be changed

The `gtk.accel_map_change_entry()` function changes the keyval and modifier mask currently associated with the accelerator path specified by *accel_path* to the values specified by *accel_key* and *accel_mods* respectively. Due to conflicts with other accelerators, a change may not always be possible. If *replace* is `TRUE` the other accelerators may be deleted to resolve such conflicts. A change will only occur if all conflicts could be resolved (which might not be the case if conflicting accelerators are locked). Successful changes are indicated by a `TRUE` return value.

gtk.accel_map_load

```
def gtk.accel_map_load(file_name)
```

file_name :	the file containing accelerator specifications

The `gtk.accel_map_load()` function parses the file (specified by *file_name*) previously saved with the gtk.accel_map_save() function for accelerator specifications, and propagates them accordingly.

gtk.accel_map_save

```
    def gtk.accel_map_save(file_name)
```

file_name :	the file to save the accelerator specifications in

The `gtk.accel_map_save()` function saves current accelerator specifications (accelerator path, key and modifiers) to the file specified by *`file_name`*. The file is written in a format suitable to be read back in by the `gtk.accel_map_load()` function.

gtk.accel_map_load_fd

```
    def gtk.accel_map_load_fd(fd)
```

fd :	a Python file object or an integer file descriptor

The `gtk.accel_map_load_fd()` function loads the accelerator map from the open Python file object specified by *`fd`*. *`fd`* may also be an integer file descriptor. See the `gtk.accel_map_load()` function.

gtk.accel_map_save_fd

```
    def gtk.accel_map_save_fd(fd)
```

fd :	a Python file object or an integer file descriptor

The `gtk.accel_map_save_fd()` function saves the accelerator map into the open Python file object specified by fd. *`fd`* may also be an integer file descriptor. See the `gtk.accel_map_save()` function.

gtk.accel_map_lock_path

```
    def gtk.accel_map_lock_path(accel_path)
```

accel_path :	a valid accelerator path

Note

This function is available in PyGTK 2.4 and above.

The `gtk.accel_map_lock_path()` function locks the accelerator path specified by *`accel_path`*. If the accelerator map doesn't yet contain an entry for *`accel_path`*, a new one is created.

Locking an accelerator path prevents its accelerator from being changed during runtime. A locked accelerator path can be unlocked by the `gtk.accel_map_unlock_path()` function. Refer to the `gtk.accel_map_change_entry()` function for information about runtime accelerator changes.

If called more than once, *`accel_path`* remains locked until the `gtk.accel_map_unlock_path()` function has been called an equivalent number of times.

Note that locking of individual accelerator paths is independent from locking the `gtk.AccelGroup` containing them. For runtime accelerator changes to be possible both the accelerator path and its `gtk.AccelGroup` have to be unlocked.

gtk.accel_map_unlock_path

```
    def gtk.accel_map_unlock_path(accel_path)
```

accel_path :	a valid accelerator path

Note

This function is available in PyGTK 2.4 and above.

The `gtk.accel_map_unlock_path()` function undoes the last call to the `gtk.accel_map_lock_path()` function on the accelerator path specified by *`accel_path`*. Refer to the `gtk.accel_map_lock_path()` function for information about accelerator path locking.

gtk.accel_map_add_filter

```
def gtk.accel_map_add_filter(filter_pattern)
```

`filter_pattern` :	a glob–style pattern
Returns :	

The `gtk.accel_map_add_filter()` function adds the filter pattern specified by *`filter_pattern`* to the global list of accel path filters. The pattern specified by *`filter_pattern`* contain '*' and '?' wildcards with similar semantics as the Python `glob.py` and `fnmatch.py` modules: '*' matches an arbitrary, possibly empty, string, '?' matches an arbitrary character. Note that in contrast to `glob.py`, the '/' character can be matched by the wildcards, there are no '[...]' character ranges and '*' and '?' can not be escaped to include them literally in a pattern. This function is intended for `PyGTK` modules that create their own menus, but don't want them to be saved into the applications accelerator map dump.

gtk.accel_groups_from_object

```
def gtk.accel_groups_from_object(object)
```

`object` :	a GObject usually a `gtk.Window`

Note

This function is available in PyGTK 2.4 and above.

The `gtk.accel_groups_from_object()` function returns a list of all the `gtk.AccelGroup` objects attached to the object specified by *`object`*.

Signals

The "accel–activate" gtk.AccelGroup Signal

```
def callback(accelgroup, acceleratable, accel_key, accel_mods, user_param1, ...)
```

`accelgroup` :	the accelgroup that received the signal
`acceleratable` :	the object that the accelerator is associated with
`accel_key` :	the accelerator key value
`accel_mods` :	the accelerator modifiers
`user_param1` :	the first user parameter (if any) specified with the `gobject.GObject.connect()` method
`...` :	additional user parameters (if any)
Returns :	`TRUE` if the accelerator was handled

The "accel–activate" signal is emitted when an accelerator is activated.

The "accel−changed" gtk.AccelGroup Signal

```
    def callback(accelgroup, accel_key, accel_mods, closure, user_param1, ...)
```

accelgroup :	the accelgroup that received the signal
accel_key :	the key value of the accelerator
accel_mods :	the modifiers of the accelerator
closure :	the closure of the accelerator
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "accel−changed" signal is emitted when an accelerator is added or removed from an accelerator group.

Prev Up Next
gtk.AboutDialog Home gtk.AccelLabel

gtk.AccelLabel
Prev **The gtk Class Reference** Next

gtk.AccelLabel

gtk.AccelLabel — a label which displays accelerator info to the right of the text

Synopsis

```
class gtk.AccelLabel(gtk.Label):
    gtk.AccelLabel(string)
    def accelerator_width()
    def get_accel_widget()
    def get_accel_width()
    def set_accel_widget(accel_widget)
    def refetch()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Misc
        +-- gtk.Label
          +-- gtk.AccelLabel
```

Properties

"accel−closure"	Read−Write	The closure to be monitored for accelerator changes
"accel−widget"	Read−Write	The widget associated with the accelerator label

Description

The `gtk.AccelLabel` widget is a subclass of `gtk.Label` that displays an accelerator string to the right of the label text, e.g. **Ctrl+S**. It is commonly used in menus to show the keyboard shortcuts for commands. The accelerator string to display is not set explicitly; instead, the `gtk.AccelLabel` displays the accelerators which have been added to the associated widget. A widget is associated with the accelerator label by calling `set_accel_widget()`.

For example, a `gtk.MenuItem` widget may have an accelerator added to emit the "activate" signal when the **Ctrl+S** key combination is pressed. A `gtk.AccelLabel` is created and added to the `gtk.MenuItem`, and `set_accel_widget()` is called with the `gtk.MenuItem` as the first argument. The `gtk.AccelLabel` will now display **Ctrl+S** after its label.

Creating a `gtk.MenuItem` with the `gtk.MenuItem()` function (or one of the similar functions `gtk.CheckMenuItem()` and `gtk.RadioMenuItem()`) and specifying a label, automatically adds a `gtk.AccelLabel` to the `gtk.MenuItem` and calls `set_accel_widget()` to set it up for you.

A `gtk.AccelLabel` will only display accelerators which have `gtk.ACCEL_VISIBLE` set. A `gtk.AccelLabel` can display multiple accelerators and even signal names, though it is almost always used to display just one accelerator.

The following code fragment creates a simple menu item with an accelerator and enables the display of the accelerator key string in the menu item label:

```
    # Create an accelgroup and add it to the window
    accel_group = gtk.AccelGroup()
    window.add_accel_group(accel_group)

    # Create the menu item
    save_item = gtk.MenuItem("Save")
    save_item.show()
    menu.add(save_item)

    # Now add the accelerator to the menu item. Note that since we created
    # the menu item with a label the AccelLabel is automatically setup to
    # display the accelerators.
    save_item.add_accelerator("activate", accel_group, ord("S"),
                              gtk.gdk.CONTROL_MASK, gtk.ACCEL_VISIBLE)
```

Constructor

```
  gtk.AccelLabel(string)
```

`string` :	the label string
Returns :	a new `gtk.AccelLabel` object

`gtk.AccelLabel()` creates a new `gtk.AccelLabel` object. The *`string`* parameter specifies the text to be displayed by the label. The accelerator text is automatically added by the associated widget.

Methods

gtk.AccelLabel.accelerator_width

```
  def accelerator_width()
```

Returns :	the width in pixels needed

The `accelerator_width()` method returns the width in pixels needed to display the accelerator(s). It is used by menus to align all of the `gtk.MenuItem` widgets, and isn't usually needed by applications.

Note

`accelerator_width`() is deprecated – use the `get_accel_width()` method instead.

gtk.AccelLabel.get_accel_widget

```
def get_accel_widget()
```

Returns :	the widget associated with the accelerator label, or `None`.

The `get_accel_widget`() method retrieves the widget associated with this accelerator label. See `gtk.AccelLabel.set_accel_widget()`.

gtk.AccelLabel.get_accel_width

```
def get_accel_width()
```

Returns :	the width in pixels needed

The `get_accel_width`() method returns the width in pixels needed to display the accelerator(s). It is used by menus to align all of the `gtk.MenuItem` widgets, and isn't usually needed by applications.

gtk.AccelLabel.set_accel_widget

```
def set_accel_widget(accel_widget)
```

`accel_widget` :	the widget to be associated.

The `set_accel_widget`() method associates the accelerator label with the widget specified by *`accel_widget`*.

gtk.AccelLabel.refetch

```
def refetch()
```

Returns :	FALSE

The `refetch`() method recreates the accelerator label string holding the accelerator information when the accelerator is changed. The size of the string is also recalculated.

This method is not usually needed by applications since the accelerator label string is automatically updated whenever accelerators are added or removed from the associated widget.

gtk.Accessible

gtk.Accessible — accessibility support for widgets.

Synopsis

```
class gtk.Accessible(atk.Object):
    def connect_widget_destroyed()
```

Ancestry

```
+-- gobject.GObject
  +-- atk.Object
    +-- gtk.Accessible
```

Description

The gtk.Accessible class is an abstract base class.

Methods

gtk.Accessible.connect_widget_destroyed

```
   def connect_widget_destroyed()
```

This method specifies the callback function to be called when the widget corresponding to a `gtk.Accessible` is destroyed.

Prev Up Next
gtk.AccelLabel Home gtk.Action

gtk.Action
Prev The gtk Class Reference Next

gtk.Action

gtk.Action an action which can be triggered by a menu or toolbar item (new in PyGTK 2.4)

Synopsis

```
class gtk.Action(gobject.GObject):
    gtk.Action(name, label, tooltip, stock_id)
    def get_name()
    def is_sensitive()
    def get_sensitive()
    def is_visible()
    def get_visible()
    def activate()
    def create_icon(icon_size)
    def create_menu_item()
    def create_tool_item()
    def connect_proxy(proxy)
    def disconnect_proxy(proxy)
    def get_proxies()
    def connect_accelerator()
    def disconnect_accelerator()
    def block_activate_from(proxy)
    def unblock_activate_from(proxy)
    def set_accel_path(accel_path)
    def set_accel_group(accel_group)
    def set_sensitive(sensitive)
    def set_visible(visible)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Action
```

Properties

"action–group"	Read–Write	The `gtk.ActionGroup` this `gtk.Action` is associated with, or `None` (for internal use).
"hide–if–empty"	Read–Write	If `TRUE`, empty menu proxies for this action are hidden.
"is–important"	Read–Write	If `TRUE`, `gtk.ToolItem` proxies for this action show text in `gtk.TOOLBAR_BOTH_HORIZ` mode.
"label"	Read–Write	The label used for menu items and buttons that activate this action.
"name"	Read–Write–Construct Only	A unique name for the action.
"sensitive"	Read–Write	If `TRUE`, the action is enabled.
"short–label"	Read–Write	A shorter label that may be used on toolbar buttons.
"stock–id"	Read–Write	The stock icon displayed in widgets representing this action.
"tooltip"	Read–Write	A tooltip for this action.
"visible"	Read–Write	If `TRUE`, the action is visible.
"visible–horizontal"	Read–Write	If `TRUE`, the toolbar item is visible when the toolbar is in a horizontal orientation.
"visible–vertical"	Read–Write	If `TRUE`, the toolbar item is visible when the toolbar is in a vertical orientation.

Signal Prototypes

"activate"	`def callback(`*`action, user_param1, ...`*`)`

Description

Note

This widget is available in PyGTK 2.4 and above.

A `gtk.Action` represents operations that the user can perform, along with some information how it should be presented in the interface. Each `gtk.Action` provides methods to create icons, menu items and toolbar items representing itself.

As well as the callback that is called when the action gets activated, the following also gets associated with the action:

- a name (not translated, for path lookup)

- a label (translated, for display)
- an accelerator
- whether the label indicates a stock id
- a tooltip (optional, translated)
- a toolbar label (optional, shorter than label)

The action will also have some state information:

- visible (shown/hidden)
- sensitive (enabled/disabled)

Apart from regular actions, there are toggle actions, which can be toggled between two states and radio actions, where only one in a group can be in the "active" state. Other actions can be implemented as `gtk.Action` subclasses.

Each `gtk.Action` can have one or more proxy menu items, toolbar buttons or other proxy widgets. Proxies mirror the state of the action (text label, tooltip, icon, visible, sensitive, etc), and should change when the action's state changes. When the proxy is activated, it should activate its action.

Constructor

```
gtk.Action(name, label, tooltip, stock_id)
```

`name` :	a unique name for the `gtk.Action`
`label` :	the label displayed in menu items and on buttons
`tooltip` :	a tooltip for the action
`stock_id` :	the stock icon to display in widgets representing the action
Returns :	a new `gtk.Action`

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new `gtk.Action` object. To add the action to a `gtk.ActionGroup` and set the accelerator for the action, call the `gtk.ActionGroup.add_action_with_accel()`.

Methods

gtk.Action.get_name

```
def get_name()
```

Returns :	the name of the action.

Note

This method is available in PyGTK 2.4 and above.

The `get_name()` method returns the value of the "name" property containing the name of the action.

gtk.Action.is_sensitive

```
    def is_sensitive()
```

Returns :	`TRUE` if the action and its associated action group are both sensitive.

Note

This method is available in PyGTK 2.4 and above.

The `is_sensitive`() method returns `TRUE` if the action is effectively sensitive i.e. both the `gtk.Action` and its associated `gtk.ActionGroup` are sensitive.

gtk.Action.get_sensitive

```
    def get_sensitive()
```

Returns :	`TRUE` if the action itself is sensitive.

Note

This method is available in PyGTK 2.4 and above.

The `get_sensitive`() method returns the value of the "sensitive" property. If "sensitive" is `TRUE` the action itself is sensitive. Note that this doesn't necessarily mean effective sensitivity. See the `is_sensitive()` method for more information.

gtk.Action.is_visible

```
    def is_visible()
```

Returns :	`TRUE` if the action and its associated action group are both visible.

Note

This method is available in PyGTK 2.4 and above.

The `is_visible`() method returns `TRUE` if the action is effectively visible i.e. both the `gtk.Action` and its associated `gtk.ActionGroup` as visible.

gtk.Action.get_visible

```
    def get_visible()
```

Returns :	`TRUE` if the action itself is visible.

Note

This method is available in PyGTK 2.4 and above.

The `get_visible`() method returns the value of the "visible" property. If "visible" is `TRUE` the `gtk.Action` itself is visible. Note that this doesn't necessarily mean effective visibility. See the `is_visible()` method for more information.

gtk.Action.activate

```
def activate()
```

Note

This method is available in PyGTK 2.4 and above.

The `activate()` method emits the "activate" signal on the `gtk.Action`, if it isn't insensitive. This gets called by the proxy widgets when they get activated. It can also be used to manually activate an action.

gtk.Action.create_icon

```
def create_icon(icon_size)
```

`icon_size` :	the size of the icon that should be created.
Returns :	a widget that displays the icon for this action.

Note

This method is available in PyGTK 2.4 and above.

The `create_icon()` method creates and returns a `gtk.Image` with the size specified by *`size`* from the icon contained in the "stock–id" property if it exists. The value of *`size`* must be one of:

- `gtk.ICON_SIZE_MENU`
- `gtk.ICON_SIZE_SMALL_TOOLBAR`
- `gtk.ICON_SIZE_LARGE_TOOLBAR`
- `gtk.ICON_SIZE_BUTTON`
- `gtk.ICON_SIZE_DND`
- `gtk.ICON_SIZE_DIALOG`

This method is intended for use by `gtk.Action` implementations to create icons displayed in the proxy widgets.

gtk.Action.create_menu_item

```
def create_menu_item()
```

Returns :	a menu item connected to the action.

Note

This method is available in PyGTK 2.4 and above.

The `create_menu_item()` method creates and returns a menu item widget that proxies for the `gtk.Action`.

gtk.Action.create_tool_item

```
def create_tool_item()
```

Returns :	a tool item connected to the action.

Note

This method is available in PyGTK 2.4 and above.

The `create_tool_item()` method creates and returns a tool item widget that proxies for the `gtk.Action`.

gtk.Action.connect_proxy

```
    def connect_proxy(proxy)
```

`proxy` :	the proxy widget

Note

This method is available in PyGTK 2.4 and above.

The `connect_proxy()` method connects the widget specified by *`proxy`* to the `gtk.Action` object as a proxy. This method synchronizes various properties of the `gtk.Action` with the widget (such as label text, icon, tooltip, etc), and attaches a callback so that the `gtk.Action` is activated when *`proxy`* is.

If *`proxy`* is already connected to another `gtk.Action`, it is disconnected first. The `gtk.Action` should be added to a `gtk.ActionGroup` before calling this method.

gtk.Action.disconnect_proxy

```
    def disconnect_proxy(proxy)
```

`proxy` :	the proxy widget

Note

This method is available in PyGTK 2.4 and above.

The `disconnect_proxy()` method disconnects the widget specified by *`proxy`* from the `gtk.Action`. This method does *not* destroy the widget. The `gtk.Action` should be added to a `gtk.ActionGroup` before calling this method.

gtk.Action.get_proxies

```
    def get_proxies()
```

Returns :	a list of proxy widgets.

Note

This method is available in PyGTK 2.4 and above.

The `get_proxies()` method returns a list containing the proxy widgets associated with the `gtk.Action`.

gtk.Action.connect_accelerator

```
    def connect_accelerator()
```

Note

This method is available in PyGTK 2.4 and above.

The `connect_accelerator()` method installs the accelerator for the `gtk.Action` if it has an associated accel path and `gtk.AccelGroup`. See the `set_accel_path()` and the `set_accel_group()` methods.

Since multiple proxies may independently trigger the installation of the accelerator, the `gtk.Action` counts the number of times this method has been called and doesn't remove the accelerator until `disconnect_accelerator()` has been called as many times.

gtk.Action.disconnect_accelerator

```
    def disconnect_accelerator()
```

Note

This method is available in PyGTK 2.4 and above.

The `disconnect_accelerator()` method undoes the effect of one call to the `connect_accelerator()` method.

gtk.Action.block_activate_from

```
    def block_activate_from(proxy)
```

proxy :	a proxy widget

Note

This method is available in PyGTK 2.4 and above.

The `block_activate_from()` method disables calls to the `activate()` method by signals on the widget specified by *proxy*. This is used to break notification loops for things like check or radio actions.

This method is intended for use by `gtk.Action` implementations.

gtk.Action.unblock_activate_from

```
    def unblock_activate_from(proxy)
```

proxy :	a proxy widget

Note

This method is available in PyGTK 2.4 and above.

The `unblock_activate_from()` method re−enables calls to the `activate()` method by signals on the widget specified by *proxy*. This undoes the blocking done by the `block_activate_from()` method.

This method is intended for use by `gtk.Action` implementations.

gtk.Action.set_accel_path

```
def set_accel_path(accel_path)
```

`accel_path` :	the accelerator path

Note

This method is available in PyGTK 2.4 and above.

The `sel_accel_path()` method sets the accel path for the action to the value of *`accel_path`*. All proxy widgets associated with the action will have this accel path, so that their accelerators are consistent.

gtk.Action.set_accel_group

```
def set_accel_group(accel_group)
```

`accel_group` :	a `gtk.AccelGroup` or `None`

Note

This method is available in PyGTK 2.4 and above.

The `set_accel_group()` method sets the `gtk.AccelGroup` specified by *`accel_group`* as the accelerator group for the `gtk.Action`.

gtk.Action.set_sensitive

```
def set_sensitive(sensitive)
```

`sensitive` :	if `TRUE` make the action sensitive

Note

This method is available in PyGTK 2.6 and above.

The `set_sensitive()` method sets the "sensitive" property to the value of *`sensitive`*. Note that this doesn't necessarily set the effective sensitivity. See the `is_sensitive()` method for more information.

gtk.Action.set_visible

```
def set_visible(visible)
```

`visible` :	if `TRUE` make the action visible

Note

This method is available in PyGTK 2.6 and above.

The `set_visible()` method sets the "visible" property to the value of *`visible`*. Note that this doesn't necessarily set the effective visibility. See the `is_visible()` method for more information.

Signals

The "activate" gtk.Action Signal

```
    def callback(action, user_param1, ...)
```

action :	the gtk.Action that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

Note

This signal is available in PyGTK 2.4 and above.

The "activate" signal is emitted when *action* is activated.

Prev Up Next
gtk.Accessible Home gtk.ActionGroup
gtk.ActionGroup
Prev The gtk Class Reference Next

gtk.ActionGroup

gtk.ActionGroup a group of actions (new in PyGTK 2.4)

Synopsis

```
class gtk.ActionGroup(gobject.GObject):
    gtk.ActionGroup(name)
    def get_name()
    def get_sensitive()
    def set_sensitive(sensitive)
    def get_visible()
    def set_visible(visible)
    def get_action(action_name)
    def list_actions()
    def add_action(action)
    def add_action_with_accel(action, accelerator)
    def remove_action(action)
    def add_actions(entries, user_data=None)
    def add_toggle_actions(entries, user_data=None)
    def add_radio_actions(entries, value=0, on_change=None, user_data=None)
    def set_translation_domain(domain)
    def translate_string(string)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.ActionGroup
```

Properties

"name"	Read−Write	A name for the action group.
"sensitive"	Read−Write	If TRUE, the action group is enabled.
"visible"	Read−Write	If TRUE, the action group is visible.

Signal Prototypes

"connect–proxy"	`def callback(actiongroup, action, proxy, user_param1, ...)`
"disconnect–proxy"	`def callback(actiongroup, action, proxy, user_param1, ...)`
"post–activate"	`def callback(actiongroup, action, user_param1, ...)`
"pre–activate"	`def callback(actiongroup, action, user_param1, ...)`

Description

Note

This object is available in PyGTK 2.4 and above.

`gtk.Action` objects are organized into `gtk.ActionGroup` objects. An action group is basically a map from names to `gtk.Action` objects.

All actions that would make sense to use in a particular context should be in a single action group. Multiple action groups may be used for a particular user interface. In fact, it is expected that most nontrivial applications will make use of multiple groups. For example, in an application that can edit multiple documents, there could be one group holding global actions (e.g. quit, about, new), and one group per document holding actions that act on that document (e.g. save, cut/copy/paste, etc). Each window's menus would be constructed from a combination of the two action groups.

Accelerators are handled by the GTK+ accelerator map. All actions are assigned an accelerator path (which normally has the form "<Actions>/group–name/action–name") and a shortcut is associated with this accelerator path. All menuitems and toolitems take on this accelerator path. The GTK+ accelerator map code makes sure that the correct shortcut is displayed next to the menu item.

Constructor

```
    gtk.ActionGroup(name)
```

name :	the name of the action group.
Returns :	the new `gtk.ActionGroup`

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new `gtk.ActionGroup` object with the name specified by *name*. The name of the action group is used when associating keybindings with the actions.

Methods

gtk.ActionGroup.get_name

```
    def get_name()
```

Returns :	the name of the action group.

Note

This method is available in PyGTK 2.4 and above.

The `get_name()` method returns the value of the "name" property that contains the name of the action group.

gtk.ActionGroup.get_sensitive

```
    def get_sensitive()
```

Returns :	`TRUE` if the group is sensitive.

Note

This method is available in PyGTK 2.4 and above.

The `get_sensitive()` method returns the value of the "sensitive" property. If "sensitive" is `TRUE` the group is enabled. The constituent actions can only be logically sensitive (see the `gtk.Action.is_sensitive()` method) if they are sensitive (see the `gtk.Action.get_sensitive()` method) and their group is sensitive.

gtk.ActionGroup.set_sensitive

```
    def set_sensitive(sensitive)
```

`sensitive` :	if `TRUE`, the group is enabled

Note

This method is available in PyGTK 2.4 and above.

The `set_sensitive()` method sets the "sensitive" property to the value of *`sensitive`*. If *`sensitive`* is `TRUE`, the `gtk.ActionGroup` is enabled.

gtk.ActionGroup.get_visible

```
    def get_visible()
```

Returns :	`TRUE` if the group is visible.

Note

This method is available in PyGTK 2.4 and above.

The `get_visible()` method returns the value of the "visible" property. If "visible" is `TRUE`, the group is visible. The constituent actions can only be logically visible (see the `gtk.Action.is_visible()` method) if they are visible (see the `gtk.Action.get_visible()` method) and their group is visible.

gtk.ActionGroup.set_visible

```
    def set_visible(visible)
```

`visible` :	if `TRUE`, the group will be visible

Note

This method is available in PyGTK 2.4 and above.

The `set_visible()` method sets the "visible" property to the value of *`visible`*. If *`visible`* is `TRUE` the `gtk.ActionGroup` will be visible.

gtk.ActionGroup.get_action

```
    def get_action(action_name)
```

`action_name` :	the name of the action
Returns :	the action, or `None` if no action with that name exists.

Note

This method is available in PyGTK 2.4 and above.

The `get_action()` method retrieves the action in the action group with the name specified by *`action_name`*.

gtk.ActionGroup.list_actions

```
    def list_actions()
```

Returns :	a list of the action objects in the action group

Note

This method is available in PyGTK 2.4 and above.

The `list_action()` method returns a list containing the `gtk.Action` objects in the action group.

gtk.ActionGroup.add_action

```
    def add_action(action)
```

`action` :	an action

Note

This method is available in PyGTK 2.4 and above.

The `add_action()` method adds the `gtk.Action` specified by *`action`* to the action group.

gtk.ActionGroup.add_action_with_accel

```
    def add_action_with_accel(action, accelerator)
```

`action` :	the action to add
`accelerator` :	the accelerator for the action, in the format understood by the `gtk.accelerator_parse()` function, or `None` to use the stock accelerator

Note

This method is available in PyGTK 2.4 and above.

The `add_action_with_accel()` method adds a `gtk.Action` specified by *`action`* to the action group and sets up the accelerator specified by *`accelerator`*.

If *`accelerator`* is `None`, this method attempts to use the accelerator associated with the "stock_id" property of the `gtk.Action`.

Accel paths are set to `<Actions>/`*`group-name`*`/`*`action-name`*.

gtk.ActionGroup.remove_action

```
    def remove_action(action)
```

`action` :	a `gtk.Action`

Note

This method is available in PyGTK 2.4 and above.

The `remove_action()` method removes the `gtk.Action` specified by *`action`* from the action group.

gtk.ActionGroup.add_actions

```
    def add_actions(entries, user_data=None)
```

`entries` :	a list or tuple of action descriptions
`user_data` :	data to pass to the action callbacks

Note

This method is available in PyGTK 2.4 and above.

The `add_actions()` method is a convenience method that creates a number of `gtk.Action` objects based on the information in the list of action entry tuples contained in *`entries`* and adds them to the action group. The entry tuples can vary in size from one to six items with the following information:

- The name of the action. Must be specified.
- The stock id for the action. Optional with a default value of `None` if a label is specified.
- The label for the action. This field should typically be marked for translation, see the `set_translation_domain()` method. Optional with a default value of `None` if a stock id is specified.
- The accelerator for the action, in the format understood by the `gtk.accelerator_parse()` function. Optional with a default value of `None`.
- The tooltip for the action. This field should typically be marked for translation, see the `set_translation_domain()` method. Optional with a default value of `None`.
- The callback function invoked when the action is activated. Optional with a default value of `None`.

The "activate" signals of the actions are connected to the callbacks and their accel paths are set to `<Actions>/`*`group-name`*`/`*`action-name`*.

gtk.ActionGroup.add_toggle_actions

```
    def add_toggle_actions(entries, user_data=None)
```

entries :	a list or tuple of toggle action entry tuples
user_data :	data to pass to the action callbacks

Note

This method is available in PyGTK 2.4 and above.

The `add_toggle_actions()` method is a convenience method that creates a number of `gtk.ToggleAction` objects based on the information in the list of action entry tuples contained in *`entries`* and adds them to the action group. The toggle action entry tuples can vary in size from one to six items with the following information:

- The name of the action. Must be specified.
- The stock id for the action. Optional with a default value of `None` if a label is specified.
- The label for the action. This field should typically be marked for translation, see the `set_translation_domain()` method. Optional with a default value of `None` if a stock id is specified.
- The accelerator for the action, in the format understood by the `gtk.accelerator_parse()` function. Optional with a default value of `None`.
- The tooltip for the action. This field should typically be marked for translation, see the `set_translation_domain()` method. Optional with a default value of `None`.
- The callback function invoked when the action is activated. Optional with a default value of `None`.
- A flag indicating whether the toggle action is active. Optional with a default value of `FALSE`.

The "activate" signals of the actions are connected to the callbacks and their accel paths are set to `<Actions>/`*`group-name`*`/`*`action-name`*.

gtk.ActionGroup.add_radio_actions

```
    def add_radio_actions(entries, value=0, on_change=None, user_data=None)
```

entries :	a list or tuple of radio action entry tuples
value :	the value of the radio action to set active
on_change :	a callback to connect to the "changed" signal of the first radio action
user_data :	data to pass to the *`on_change`* callback

Note

This method is available in PyGTK 2.4 and above.

The `add_radio_actions()` method is a convenience method that creates a number of `gtk.RadioAction` objects based on the information in the list of action entry tuples contained in *`entries`* and adds them to the action group. The entry tuples can vary in size from one to six items with the following information:

- The name of the action. Must be specified.
- The stock id for the action. Optional with a default value of `None` if a label is specified.
- The label for the action. This field should typically be marked for translation, see the `set_translation_domain()` method. Optional with a default value of `None` if a stock id is specified.

- The accelerator for the action, in the format understood by the `gtk.accelerator_parse()` function. Optional with a default value of `None`.
- The tooltip for the action. This field should typically be marked for translation, see the `set_translation_domain()` method. Optional with a default value of `None`.
- The value to set on the radio action. Optional with a default value of `0`. Should be specified in applications.

The *`value`* parameter specifies the radio action that should be set active. The "changed" signal of the first radio action is connected to the *`on_change`* callback (if specified and not `None`) and the accel paths of the actions are set to `<Actions>/`*`group-name`*`/`*`action-name`*.

gtk.ActionGroup.set_translation_domain

```
    def set_translation_domain(domain)
```

`domain` :	the translation domain to use for `dgettext()` calls

Note

This method is available in PyGTK 2.4 and above.

The `set_translation_domain()` method sets the translation domain to the string specified by *`domain`* and uses `dgettext()` for translating the *`label`* and *`tooltip`* strings of the actions added by the `add_actions()`, `add_toggle_actions()` and `add_radio_actions()` methods.

gtk.ActionGroup.translate_string

```
    def translate_string(string)
```

`string` :	the string to be translated
Returns :	the translation of *`string`*

Note

This method is available in PyGTK 2.6 and above.

The `translate_string()` method translates the string specified by *`string`* using the specified `translate_func()`. This is mainly intended for language bindings.

Signals

The "connect–proxy" gtk.ActionGroup Signal

```
    def callback(actiongroup, action, proxy, user_param1, ...)
```

`actiongroup` :	the actiongroup that received the signal
`action` :	the action that is associated with *`proxy`*
`proxy` :	the proxy widget associated with *`action`*
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

Note

This signal is available in PyGTK 2.4 and above.

The "connect−proxy" signal is emitted when the widget specified by *proxy* is connected to the gtk.Action specified by *action*.

The "disconnect−proxy" gtk.ActionGroup Signal

```
def callback(actiongroup, action, proxy, user_param1, ...)
```

actiongroup :	the actiongroup that received the signal
action :	the action that is associated with *proxy*
proxy :	the proxy widget associated with *action*
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

Note

This signal is available in PyGTK 2.4 and above.

The "disconnect−proxy" signal is emitted when the widget specified by *proxy* is disconnected from the gtk.Action specified by *action*.

The "post−activate" gtk.ActionGroup Signal

```
def callback(actiongroup, action, user_param1, ...)
```

actiongroup :	the actiongroup that received the signal
action :	the action that is being activated
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

Note

This signal is available in PyGTK 2.4 and above.

The "post−activate" signal is emitted after the gtk.Action specified by *action* has been activated.

The "pre−activate" gtk.ActionGroup Signal

```
def callback(actiongroup, action, user_param1, ...)
```

actiongroup :	the actiongroup that received the signal
action :	the action that is being activated
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

Note

This signal is available in PyGTK 2.4 and above.

The "pre−activate" signal is emitted before the gtk.Action specified by *action* is activated.

gtk.Adjustment

gtk.Adjustment — an object representing an adjustable bounded value

Synopsis

```
class gtk.Adjustment(gtk.Object):
    gtk.Adjustment(value=0, lower=0, upper=0, step_incr=0, page_incr=0, page_size=0)
    def set_all(value, lower, upper, step_increment, page_increment, page_size)
    def changed()
    def value_changed()
    def clamp_page(lower, upper)
    def get_value()
    def set_value(value)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Adjustment
```

Signal Prototypes

"changed"	`def callback(adjustment, user_param1, ...)`
"value–changed"	`def callback(adjustment, user_param1, ...)`

Properties

Note

These properties are only available in GTK+ 2.4 and above. The GTK+ version is contained in the 3–tuple gtk.gtk_version.

"lower"	Read–Write	The minimum value of the adjustment. (float)
"page–increment"	Read–Write	The page increment of the adjustment. (float)
"page–size"	Read–Write	The page size of the adjustment. (float)
"step–increment"	Read–Write	The step increment of the adjustment. (float)
"upper"	Read–Write	The maximum value of the adjustment. (float)
"value"	Read–Write	The value of the adjustment. (float)

Attributes

"value"	Read–Write	the current value (float)
"lower"	Read–Write	the minimum value (float)
"upper"	Read–Write	the maximum value (float)
"step_increment"	Read–Write	the increment to use to make minor changes to the value. In a `gtk.Scrollbar` this increment is used when the mouse is clicked on the arrows at the top and bottom of the scrollbar, to scroll by a small amount. (float)
"page_increment"	Read–Write	the increment to use to make major changes to the value. In a `gtk.Scrollbar` this increment is used when the mouse is clicked in the trough, to scroll by a large amount. (float)
"page_size"	Read–Write	a widget specific size. In a `gtk.Scrollbar` this is the size of the area which is currently visible. (float)

Description

The `gtk.Adjustment` object contains a value which has an associated lower and upper bound, together with step and page increments, and a page size. It is used in conjunction with several PyGTK widgets, including `gtk.SpinButton`, `gtk.Viewport`, and `gtk.Range` (which is a base class for `gtk.HScrollbar`, `gtk.VScrollbar`, `gtk.HScale`, and `gtk.VScale`).

A `gtk.Adjustment` can be shared by multiple widgets. The `gtk.Adjustment` object does not update the value itself. Instead it is left up to the associated widget(s) that use the `gtk.Adjustment` to control the value.

The widget using the `gtk.Adjustment` typically calls the `value_changed()` or `changed()` methods after changing the value or its bounds. This results in the emission of the "value_changed" or "changed" signal respectively.

A `gtk.Adjustment` object contains several attributes that provide access to its value and bounds:

- value
- lower
- upper
- step_increment
- page_increment
- page_size

The attribute values can be retrieved and set similar to:

```
  adjustment.upper = 25.0
  lower = adjustment.lower
```

Constructor

```
    gtk.Adjustment(value=0, lower=0, upper=0, step_incr=0, page_incr=0, page_size=0)
```

`value` :	the initial value.
`lower` :	the minimum value.
`upper` :	the maximum value.
`step_incr` :	the step increment.

page_incr :	the page increment.
page_size :	the page size.
Returns :	a new gtk.Adjustment object

`gtk.Adjustment`() creates a new `gtk.Adjustment` object with the specified attributes. Any attributes not specified are set to 0.0 by default.

Methods

gtk.Adjustment.set_all

```
    def set_all(value, lower, upper, step_increment, page_increment, page_size)
```

value :	the new value.
lower :	the new minimum value.
upper :	the new maximum value.
step_increment :	the new step increment.
page_increment :	the new page increment.
page_size :	the new page size.

The `set_all`() method sets the attributes of the adjustment to the specified values.

gtk.Adjustment.changed

```
    def changed()
```

The `changed`() method emits a "changed" signal from the adjustment. This must typically be called if any of the adjustment attributes other than value has changed so that the widget(s) using the adjustment can reflect the changes. Applications usually will not need to use this method since setting the attributes directly will automatically invoke this method.

gtk.Adjustment.value_changed

```
    def value_changed()
```

The `value_changed`() method emits a "value_changed" signal from the adjustment. This must typically be called after the value attribute of the adjustment has changed. Applications usually will not need to use this method since setting the attribute directly will automatically invoke this method as will using the `set_value()` method.

gtk.Adjustment.clamp_page

```
    def clamp_page(lower, upper)
```

lower :	the lower value
upper :	the upper value

The `clamp_page`() method updates the adjustment value to ensure that the range between *`lower`* and *`upper`* is in the current page (i.e. between value and value + page_size). If the range is larger than the page size, then only the start of it will be in the current page. A "changed" signal will be emitted if the value is changed.

gtk.Adjustment.get_value

```
    def get_value()
```

Returns :	The current value of the adjustment.

The `get_value`() method gets the current value of the adjustment.

gtk.Adjustment.set_value

```
    def set_value(value)
```

`value` :	the new value

The `set_value`() method sets the value of the adjustment to the specified *`value`*.

Signals

The "changed" gtk.Adjustment Signal

```
    def callback(adjustment, user_param1, ...)
```

`adjustment` :	the object that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "changed" signal is emitted when one (or more) of the adjustment attributes (except the value attribute) has changed.

The "value−changed" gtk.Adjustment Signal

```
    def callback(adjustment, user_param1, ...)
```

`adjustment` :	the object that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "value−changed" signal is emitted when the adjustment value attribute has changed.

Prev Up Next
gtk.ActionGroup Home gtk.Alignment

gtk.Alignment
Prev **The gtk Class Reference** Next

gtk.Alignment

gtk.Alignment — a widget that controls the alignment and size of its child

Synopsis

```
class gtk.Alignment(gtk.Bin):
    gtk.Alignment(xalign=0.0, yalign=0.0, xscale=0.0, yscale=0.0)
    def set(xalign, yalign, xscale, yscale)
    def set_padding(padding_top, padding_bottom, padding_left, padding_right)
    def get_padding()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Alignment
```

Properties

"xalign"	Read/Write	The fraction of horizontal *free* space to the left of the child. 0.0 means no *free* space to the left, 1.0 means all *free* space to the left.
"yalign"	Read/Write	The fraction of vertical *free* space above the child. 0.0 means no *free* space above, 1.0 means all *free* space above.
"xscale"	Read/Write	The fraction of horizontal *free* space (beyond that needed by the child) that the child will absorb. 0.0 means the child will absorb none, 1.0 means the child will absorb all
"yscale"	Read/Write	The fraction of vertical *free* space (beyond that needed by the child) that the child will absorb. 0.0 means the child will absorb none, 1.0 means the child will absorb all
"bottom−padding"	Read/Write	The padding to insert at the bottom of the widget. GTK+ 2.4 and above.
"top−padding"	Read/Write	The padding to insert at the top of the widget. GTK+ 2.4 and above.
"left−padding"	Read/Write	The padding to insert at the left of the widget. GTK+ 2.4 and above.
"right−padding"	Read/Write	The padding to insert at the right of the widget. GTK+ 2.4 and above.

Description

The `gtk.Alignment` widget controls the alignment and size of its child widget. It has four properties: xscale, yscale, xalign, and yalign. The scale properties are used to specify how much of the *free* (extra) space allocated to the `gtk.Alignment` should be absorbed by the child widget. The values can range from 0.0 (meaning the child absorbs none) to 1.0 (meaning the child absorbs all of the *free* space). If the value is 0.5, the child widget absorbs half the *free* space. The align properties are used to place the child widget within the available area by specifying the fraction of *free* space that will be placed above or to the left of the child widget. The values range from 0.0 (no *free* space above or to the left of the child) to 1.0 (all *free* space above or to the left of the child). Of course, if the scale properties are both set to 1.0, the alignment properties have no effect.

An example may make this clearer. A `gtk.Button` widget (32 pixels wide by 32 pixels high) is placed inside a `gtk.Alignment` widget (256 pixels wide by 128 pixels high) which has xalign of 0.25, yalign of 0.25, xscale of 0.25 and yscale of 0.25. The horizontal *free* space is 256−32=224 pixels and the vertical *free* space is 128−32=96 pixels. The button will absorb 0.25x224=56 pixels horizontally and 0.25x96=24 pixels vertically since the xscale and yscale are 0.25 thus becoming 32+56=88 pixels wide by 32+24=56 pixels high. This will leave 256−88=168 pixels of horizontal *free* space and 128−56=72 pixels of vertical *free* space. Since the xalign value is 0.25 the horizontal *free* space will be allocated as 0.25x168=42 pixels to the left of the button and 0.72x168=126 pixels to the right. Likewise since the yalign is 0.25 the vertical *free* space is allocated as 0.25x72=18 pixels above the button and 0.75*72=54 pixels below.

Constructor

```
gtk.Alignment(xalign=0.0, yalign=0.0, xscale=0.0, yscale=0.0)
```

xalign :	the fraction of horizontal *free* space to the left of the child widget. Ranges from 0.0 to 1.0
yalign :	the fraction of vertical *free* space above the child widget. Ranges from 0.0 to 1.0
xscale :	the fraction of horizontal *free* space that the child widget absorbs, from 0.0 to 1.0
yscale :	the fraction of vertical *free* space that the child widget absorbs, from 0.0 to 1.0
Returns :	a new alignment object

Creates a new alignment widget with the specified properties. If the scale and alignment parameters are not specified they default to 0.0.

Methods

gtk.Alignment.set

```
def set(xalign, yalign, xscale, yscale)
```

xalign :	the fraction of horizontal *free* space to the left of the child widget. Ranges from 0.0 to 1.0
yalign :	the fraction of vertical *free* space above the child widget. Ranges from 0.0 to 1.0
xscale :	the fraction of horizontal *free* space that the child widget absorbs, from 0.0 to 1.0
yscale :	the fraction of vertical *free* space that the child widget absorbs, from 0.0 to 1.0

The `set`() method sets the properties of the alignment widget to the specified values.

gtk.Alignment.set_padding

```
def set_padding(padding_top, padding_bottom, padding_left, padding_right)
```

padding_top :	the padding at the top of the widget
padding_bottom :	the padding at the bottom of the widget
padding_left :	the padding at the left of the widget
padding_right :	the padding at the right of the widget.

Note

This method is available in PyGTK 2.4 and above.

The `set_padding`() method sets the padding around the sides of the alignment widget to the values specified by *`padding_top`*, *`padding_bottom`*, *`padding_left`* and *`padding_right`*. The padding adds blank space to the sides of the widget. For instance, this can be used to indent the child widget toward the right by adding padding on the left.

gtk.Alignment.get_padding

```
def get_padding()
```

Returns :	a 4–tuple containing the padding set on the top, bottom, left and right sides of the widget

Note

This method is available in PyGTK 2.4 and above.

The `get_padding()` method returns a 4–tuple containing the padding set on the sides of the widget.

gtk.Arrow

gtk.Arrow — produces an arrow pointing in one of the four cardinal directions.

Synopsis

```
class gtk.Arrow(gtk.Misc):
    gtk.Arrow(arrow_type, shadow_type)
    def set(arrow_type, shadow_type)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Misc
        +-- gtk.Arrow
```

Properties

"arrow–type"	Read/Write	The direction the arrow should point (gtk.ARROW_UP, gtk.ARROW_DOWN, gtk.ARROW_LEFT, or gtk.ARROW_RIGHT)
"shadow–type"	Read/Write	Appearance of the shadow surrounding the arrow (gtk.SHADOW IN, gtk.SHADOW OUT, gtk.ETCHED IN, or gtk.ETCHED OUT)

Description

The `gtk.Arrow` is used to draw simple arrows that point in one of the four cardinal directions (gtk.ARROW_UP, gtk.ARROW_DOWN, gtk.ARROW_LEFT, or gtk.ARROW_RIGHT). The shadow style of the arrow can be one of gtk.SHADOW IN, gtk.SHADOW OUT, gtk.ETCHED IN, or gtk.ETCHED OUT.

A `gtk.Arrow` will fill any space alloted to it, but since it is inherited from `gtk.Misc`, it can be padded and/or aligned, to fill exactly the space desired.

Arrows are created with a call to `gtk.Arrow()`. The direction or style of an arrow can be changed after creation by using `arrow_set()`.

Constructor

```
    gtk.Arrow(arrow_type, shadow_type)
```

arrow_type :	one of the GTK Arrow Type Constants
shadow_type :	one of the GTK Shadow Type Constants
Returns :	a `gtk.Arrow` widget

Creates a new arrow widget with the specified *arrow_type* and *arrow_shadow*.

Methods

gtk.Arrow.set

```
    def set(arrow_type, shadow_type)
```

arrow_type :	a GtkArrowType
shadow_type :	one of the GTK Shadow Type Constants

The `set`() method sets the *arrow_type* and *shadow_type* of the arrow widget.

Prev Up Next
gtk.Alignment Home gtk.AspectFrame

gtk.AspectFrame
Prev **The gtk Class Reference** Next

gtk.AspectFrame

gtk.AspectFrame — A frame that constrains its child to a particular aspect ratio.

Synopsis

```
class gtk.AspectFrame(gtk.Frame):
    gtk.AspectFrame(label=None, xalign=0.5, yalign=0.5, ratio=1.0, obey_child=TRUE)
    def set(xalign=0.0, yalign=0.0, ratio=1.0, obey_child=TRUE)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Frame
            +-- gtk.AspectFrame
```

Properties

"xalign"	Read/Write	The fraction of horizontal *free* space to the left of the child. 0.0 means no *free* space to the left, 1.0 means all *free* space to the left.
"yalign"	Read/Write	The fraction of vertical *free* space above the child. 0.0 means no *free* space above, 1.0 means all *free* space above.
"ratio"	Read/Write	The aspect ratio of the widget in the range of 0.0001 to 10000.0 if "obey−child" is FALSE.
"obey−child"	Read/Write	Make aspect ratio match that of the child widget

Description

The `gtk.AspectFrame` packs a widget so that it can resize but always retains the same aspect ratio. For instance, one might be drawing a small preview of a larger image. `gtk.AspectFrame` derives from `gtk.Frame`, so it can draw a label and a frame around the child. The frame will be "shrink−wrapped" to the size of the child.

The aspect frame "ratio" property determines the widget width:height ratio. An aspect ratio of 0.5 means the width is one half the height; an aspect ratio of 2.0 means the width is twice the height. The default value for the "ratio" property is 1.0.

The align properties are used to place the child widget within the available area by specifying the fraction of *free* space (space in the aspect frame that is not used by the child widget) that is placed above or to the left of the child widget. The values range from 0.0 (meaning no *free* space above or to the left of the child) to 1.0 (meaning all *free* space above or to the left of the child) . The default value for both align properties is 0.5.

If the "obey−child" property is TRUE (the default value), the "ratio" property is ignored and the aspect ratio is set by the child widget.

Constructor

```
    gtk.AspectFrame(label=None, xalign=0.5, yalign=0.5, ratio=1.0, obey_child=TRUE)
```

`label` :	a string used to set the aspect frame label
`xalign` :	The fraction of horizontal *free* space to the left of the child. 0.0 means no *free* space to the left, 1.0 means all *free* space to the left.
`yalign` :	The fraction of vertical *free* space above the child. 0.0 means no *free* space above, 1.0 means all *free* space above.
`ratio` :	the ratio of the child width to height (in the range 0.0001 to 10000.0) if *`obey_child`* is FALSE
`obey_child` :	if TRUE, *`ratio`* is ignored, and the aspect ratio is taken from the requisition of the child.
Returns :	a new aspect frame object

Creates a new aspect frame object with the specified *`label`*, *`xalign`* and *`yalign`* values. The default values are: *`label`*, None; *`xalign`*, 0.5; and, *`yalign`*, 0.5. If *`obey_child`* is TRUE the *`ratio`* value is ignored. If *`obey_child`* is FALSE, *`ratio`* sets the aspect ratio for the child widget. The default value for *`ratio`* is 1.0. The default value for *`obey_child`* is TRUE.

Methods

gtk.AspectFrame.set

```
  def set(xalign=0.0, yalign=0.0, ratio=1.0, obey_child=TRUE)
```

`xalign` :	The fraction of horizontal *free* space to the left of the child. 0.0 means no *free* space to the left, 1.0 means all *free* space to the left.
`yalign` :	The fraction of vertical *free* space above the child. 0.0 means no *free* space above, 1.0 means all *free* space above.
`ratio` :	the ratio of the child width to height (in the range 0.0001 to 10000.0) if *`obey_child`* is FALSE
`obey_child` :	if TRUE, *`ratio`* is ignored, and the aspect ratio is taken from the requisition of the child

The `set()` method changes the aspect frame properties to the values specified by *`xalign`*, *`yalign`*, *`ratio`* and *`obey_child`*. The default values are: *`xalign`*, 0; *`yalign`*, 0.0; *`ratio`*, 1.0; and, *`obey_child`*, TRUE.

Prev	Up	Next
gtk.Arrow	Home	gtk.Bin

gtk.Bin

Prev **The gtk Class Reference** Next

gtk.Bin

gtk.Bin an abstract base class defining a container with just one child.

Synopsis

```
class gtk.Bin(gtk.Container):
    def get_child()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
```

Attributes

"child"	Read	The child widget or `None` if there is no child.

Description

`gtk.Bin` is an abstract base class defining a widget that is a container with just one child. It is useful for deriving subclasses, since it provides the common code needed for handling a single child widget. Many PyGTK widgets are subclasses of `gtk.Bin`, including `gtk.Window`, `gtk.Button`, `gtk.Frame`, `gtk.HandleBox`, and `gtk.ScrolledWindow`.

Methods

gtk.Bin.get_child

```
def get_child()
```

Returns :	a reference to the child widget

The `get_child`() method returns a reference to the child of the bin, or `None` if the bin contains no child widget.

Prev	Up	Next
gtk.AspectFrame	Home	gtk.Border

gtk.Border

Prev **The gtk Class Reference** Next

gtk.Border

gtk.Border an object containing data for a border (new in PyGTK 2.4)

Synopsis

```
class gtk.Border(gobject.GBoxed):
    def copy()
    def free()
```

Description

Note

This object is available in PyGTK 2.4 and above.

A `gtk.Border` object contains the integer values for the left, right, top and bottom values of a border. `gtk.Border` is used in `gtk.Style` specifications. `gtk.Border` has two methods: `copy()` and `free()`.

Methods

gtk.Border.copy

```
def copy()
```

Returns :	a copy of the border.

Note

This method is available in PyGTK 2.4 and above.

The `copy`() method returns a copy of the border object.

gtk.Border.free

```
    def free()
```

Note

This method is available in PyGTK 2.4 and above.

The `free()` method frees the memory used by the border.

Prev	Up	Next
gtk.Bin	Home	gtk.Box

gtk.Box

Prev **The gtk Class Reference** Next

gtk.Box

gtk.Box an abstract base class for box containers

Synopsis

```
class gtk.Box(gtk.Container):
    def pack_start(child, expand=TRUE, fill=TRUE, padding=0)
    def pack_end(child, expand=TRUE, fill=TRUE, padding=0)
    def pack_start_defaults(widget)
    def pack_end_defaults(widget)
    def set_homogeneous(homogeneous)
    def get_homogeneous()
    def set_spacing(spacing)
    def get_spacing()
    def reorder_child(child, position)
    def query_child_packing(child)
    def set_child_packing(child, expand, fill, padding, pack_type)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Box
```

Properties

"spacing"	Read/Write	The amount of space between children.
"homogeneous"	Read/Write	If TRUE the children should all be the same size.

Child Properties

"expand"	Read/Write	If TRUE the child should receive extra space when the parent grows
"fill"	Read/Write	If TRUE extra space given to the child should be allocated to the child; if FALSE extra space given to the child should be used as padding
"padding"	Read/Write	The amount of extra space to put between the child and its neighbors, in pixels
"pack−type"	Read/Write	Indicates whether the child is packed with reference to the start (gtk.PACK_START) or end (gtk.PACK_END) of the parent
"position"	Read/Write	The index of the child in the parent

Description

The `gtk.Box` class is an abstract base class defining a widget that encapsulates functionality for a particular kind of container, one that organizes a variable number of widgets into a rectangular area. `gtk.Box` currently has two derived classes, `gtk.HBox` and `gtk.VBox`.

The rectangular area of a `gtk.Box` is organized into either a single row or a single column of child widgets depending upon whether the box is of type `gtk.HBox` or `gtk.VBox`, respectively. Thus, all children of a `gtk.Box` are allocated one dimension in common, which is the height of a row, or the width of a column.

`gtk.Box` uses a notion of packing. Packing refers to adding widgets with reference to a particular position in a `gtk.Container`. For a `gtk.Box`, there are two reference positions: the start and the end of the box. For a `gtk.VBox`, the start is defined as the top of the box and the end is defined as the bottom. For a `gtk.HBox` the start is defined as the left side and the end is defined as the right side.

Repeated calls to `pack_start()` pack widgets into a `gtk.Box` from start to end. The `pack_end()` method adds widgets from end to start. You may intersperse these calls and add widgets from both ends of the same `gtk.Box`.

Because `gtk.Box` is a `gtk.Container`, you may also use `add()` to insert widgets into the box, and they will be packed as if with the `pack_start()` method. Use `remove()` to remove widgets from the `gtk.Box`.

The `set_homogeneous()` method specifies whether or not all children of the `gtk.Box` are forced to get the same amount of space.

The `set_spacing()` method determines how much space will be minimally placed between all children in the `gtk.Box`.

The `reorder_child()` method moves a `gtk.Box` child to a different place in the box.

The `set_child_packing()` method resets the expand, fill, and padding attributes of any `gtk.Box` child. Use the `query_child_packing()` to query these properties.

Methods

gtk.Box.pack_start

```
def pack_start(child, expand=TRUE, fill=TRUE, padding=0)
```

`child` :	the widget to be added to the box
`expand` :	TRUE if *`child`* is to be given extra space allocated to box. The extra space will be divided evenly between all children of box that use this option.

fill :	TRUE if space given to *child* by the *expand* option is actually allocated to *child*, rather than just padding it. This parameter has no effect if *expand* is set to FALSE. A child is always allocated the full height of a gtk.HBox and the full width of a gtk.VBox. This option affects the other dimension.
padding :	extra space in pixels to put between *child* and its neighbors, over and above the global amount specified by *spacing* in gtk.Box. If *child* is a widget at one of the reference ends of box, then *padding* pixels are also put between *child* and the reference edge of box.

The `pack_start()` method adds *child* to the box, packed with reference to the start of box. The *child* is packed after any other child packed with reference to the start of box.

gtk.Box.pack_end

```
def pack_end(child, expand=TRUE, fill=TRUE, padding=0)
```

child :	the widget to be added to the box
expand :	TRUE if *child* is to be given extra space allocated to box. The extra space will be divided evenly between all children of box that use this option.
fill :	TRUE if space given to *child* by the *expand* option is actually allocated to *child*, rather than just padding it. This parameter has no effect if *expand* is set to FALSE. A child is always allocated the full height of a gtk.HBox and the full width of a gtk.VBox. This option affects the other dimension.
padding :	extra space in pixels to put between *child* and its neighbors, over and above the global amount specified by *spacing* in gtk.Box. If *child* is a widget at one of the reference ends of box, then *padding* pixels are also put between *child* and the reference edge of box.

The `pack_end()` method adds *child* to the box, packed with reference to the end of the box. The *child* is packed after (away from end of) any other child packed with reference to the end of the box.

gtk.Box.pack_start_defaults

```
def pack_start_defaults(widget)
```

widget :	the widget to be added to the box

Warning

This method is deprecated in PyGTK 2.4 and above.

The `pack_start_defaults()` method adds *widget* to the box, packed with reference to the start of the box. The *widget* is packed after any other child widget packed with reference to the start of box. The parameters for packing the child widget: expand, fill, and padding are given their default values, TRUE, TRUE, and 0, respectively.

gtk.Box.pack_end_defaults

```
def pack_end_defaults(widget)
```

widget :	the widget to be added to the box

Warning

This method is deprecated in PyGTK 2.4 and above.

The `pack_end_defaults()` method adds *widget* to the box, packed with reference to the end of the box. The *widget* is packed after (away from the end of) any other child widget packed with reference to the end of the box. The parameters for packing the child widget: expand, fill, and padding are given their default values, TRUE, TRUE, and 0, respectively.

gtk.Box.set_homogeneous

```
    def set_homogeneous(homogeneous)
```

`homogeneous` :	If `TRUE` the box is homogeneous i.e. all children are allocated the same space otherwise the allocations vary for each child.

The `set_homogeneous()` method sets the homogeneous (all children are allocated the same space) property of the box.

gtk.Box.get_homogeneous

```
    def get_homogeneous()
```

Returns :	`TRUE` if the box is homogeneous.

The `get_homogeneous()` method returns whether the box is homogeneous (all children are allocated the same space). See `gtk.Box.set_homogeneous()`.

gtk.Box.set_spacing

```
    def set_spacing(spacing)
```

`spacing` :	the number of pixels to put between children.

The `set_spacing()` method sets the number of pixels to place between children of the box.

gtk.Box.get_spacing

```
    def get_spacing()
```

Returns :	the spacing in pixels between children

The `get_spacing()` method returns the number of pixels used as padding between children as set by the `set_spacing()`.

gtk.Box.reorder_child

```
    def reorder_child(child, position)
```

`child` :	the child widget to move
`position` :	the new position for child in the children list of the box starting from 0. If negative, indicates the end of the list.

Moves child to a new position in the list of the box children. The list contains both widgets packed gtk.PACK_START as well as widgets packed gtk.PACK_END, in the order that these widgets were added to box.

A widget's position in the box children list determines where the widget is packed into box. A child widget at some position in the list will be packed just after all other widgets of the same packing type that appear earlier in the list.

gtk.Box.query_child_packing

```
    def query_child_packing(child)
```

child :	the child widget to be queried for its packing information
expand :	the child's expand value
fill :	the child's fill value
padding :	the child's padding value
pack_type :	the child's pack_type value

The `query_child_packing()` method returns a tuple containing information about how *child* is packed into the box. The tuple members are: (expand, fill, padding, pack_type) where: expand and fill are 0 or 1 (corresponding to FALSE or TRUE); padding is the number of pixels of padding; and pack_type is gtk.PACK_START or gtk.PACK_END.

gtk.Box.set_child_packing

```
    def set_child_packing(child, expand, fill, padding, pack_type)
```

child :	the child widget to be queried for its packing information
expand :	the child's new expand value
fill :	the child's new fill value
padding :	the child's new padding value
pack_type :	the child's new pack_type value

The `set_child_packing()` method sets the way child is packed into the box.

Prev	Up	Next
gtk.Border	Home	gtk.Button

gtk.Button

Prev **The gtk Class Reference** Next

gtk.Button

gtk.Button A pushbutton widget that issues a signal when clicked.

Synopsis

```
class gtk.Button(gtk.Bin):
    gtk.Button(label=None, stock=None, use_underline=TRUE)
    def pressed()
    def released()
    def clicked()
    def enter()
    def leave()
    def set_relief(newstyle)
    def get_relief()
    def set_label(label)
    def get_label()
    def set_use_underline(use_underline)
    def get_use_underline()
    def set_use_stock(use_stock)
    def get_use_stock()
    def set_focus_on_click(focus_on_click)
    def get_focus_on_click()
    def set_alignment(xalign, yalign)
    def get_alignment()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Button
```

Properties

"focus–on–click"	Read/Write	If `TRUE` the button grabs focus when it is clicked with the mouse. This property is available in GTK+ 2.4 and above.
"label"	Read/Write	Text of the label widget inside the button, if the button contains a label widget.
"relief"	Read/Write	The border relief style. One of: `gtk.RELIEF_NORMAL`, `gtk.RELIEF_HALF` or `gtk.RELIEF_NONE`
"use–underline"	Read/Write	If `TRUE`, an underscore in the text indicates the next character should be underlined and used for the mnemonic accelerator key if it is the first character so marked
"use–stock"	Read/Write	If `TRUE`, the label text is used to pick a stock item instead of being displayed
"xalign"	Read/Write	If the child of the button is a `gtk.Misc` or `gtk.Alignment`, this property can be used to control it's horizontal alignment. The value ranges from 0.0 to 1.0 representing the fraction of freespace to the left of the widget. This property is available in GTK+ 2.4 and above.
"yalign"	Read/Write	If the child of the button is a `gtk.Misc` or `gtk.Alignment`, this property can be used to control it's vertical alignment. The value ranges from 0.0 to 1.0 representing the fraction of freespace above the widget. This property is available in GTK+ 2.4 and above.

Style Properties

"child–displacement–y"	Read/Write	The number of pixels in the y direction to move the child when the button is depressed
"child–displacement–x"	Read/Write	The number of pixels in the x direction to move the child when the button is depressed
"default–border"	Read/Write	The extra space to add for `gtk.CAN_DEFAULT` buttons
"default–outside–border"	Read/Write	The extra space to add for `gtk.CAN_DEFAULT` buttons always drawn outside the border

Signal Prototypes

"activate"	`def callback(button, user_param1, ...)`

"clicked"	`def callback(button, user_param1, ...)`
"enter"	`def callback(button, user_param1, ...)`
"leave"	`def callback(button, user_param1, ...)`
"pressed"	`def callback(button, user_param1, ...)`
"released"	`def callback(button, user_param1, ...)`

Description

The `gtk.Button` widget is usually displayed as a pushbutton with a text label (`gtk.Label`) though it can contain any valid widget. The `gtk.Button` is generally used to attach a callback function or method that is called when the button is clicked. Buttons generate signals that indicate:

- "clicked" – the user pressed and released a mouse button over the button
- "pressed" – the user pressed a mouse button over the button
- "released" – the user released a mouse button over the button
- "enter" – the pointer entered the button
- "leave" – the pointer left the button

The "clicked" signal is usually the only signal that an application needs to handle.

If a label is being used by the button its text (the "label" property) is retrieved using the `get_label()` method. The label text is changed using the `set_label()` method.

The property ("use_underline") that tells a button to use the first underscore to indicate a mnemonic key is changed using the `set_use_underline()`. method. It can be retrieved using the `get_use_underline()` method.

The button's relief style (the "relief" property) is retrieved using the method `get_relief()`. The relief style is set to one of `gtk.RELIEF_NONE`, `gtk.RELIEF_HALF` or `gtk.RELIEF_NORMAL` using the method `set_relief()`.

Constructor

```
gtk.Button(label=None, stock=None, use_underline=TRUE)
```

`label` :	the text to be displayed by the button label including an underscore to indicate the mnemonic character if desired or `None` if no label is required.
`stock` :	the stock id identifying the stock image and text to be used in the button or `None` if no stock id is to be used.
`use_underline` :	if `TRUE`, an underscore in the text indicates the next character should be underlined and used for the mnemonic accelerator key if it is the first character so marked. Available in PyGTK 2.4 and above.
Returns :	a new button object

Creates a new button widget with the content depending on the parameters: *`label`*, *`stock`* and *`use_underline`*. The default values for *`stock`* and *`label`* are `None` and, in PyGTK 2.4 and above, *`use_underline`* is available and defaults to `TRUE`. If both *`label`* and *`stock`* are specified *`stock`* takes precedence. If neither is specified the button is created with no child widget. A child widget can be added later with the `add()` method.

If *`stock`* is specified the "use–stock" property will be set to `TRUE`.

If *label* is specified the "label" property will be set with the text of the label, the "use_underline" property will be set to TRUE and any characters that are preceded by an underscore are underlined (use two underscores to insert an underscore in a label). The first underscored character will become the mnemonic character used as the keyboard accelerator for the button when pressed simultaneously with the **Alt** key. In PyGTK 2.4 and above the "use–underline" property can be set by using the optional *use_underline* parameter.

Methods

gtk.Button.pressed

```
def pressed()
```

The pressed() method emits the "pressed" signal to the button.

gtk.Button.released

```
def released()
```

The released() method emits the "released" signal to the button.

gtk.Button.clicked

```
def clicked()
```

The clicked() method emits the "clicked" signal to the button.

gtk.Button.enter

```
def enter()
```

The enter() method emits the "enter" signal to the button.

gtk.Button.leave

```
def leave()
```

The leave() method emits the "leave" signal to the button.

gtk.Button.set_relief

```
def set_relief(newstyle)
```

newstyle :	one of gtk.RELIEF_NONE, gtk.RELIEF_NORMAL or gtk.RELIEF_HALF

The set_relief() method sets the relief style of the edges of the button. Three styles exist, gtk.RELIEF_NORMAL, gtk.RELIEF_HALF, gtk.RELIEF_NONE. The default style is, as one can guess, gtk.RELIEF_NORMAL.

gtk.Button.get_relief

```
    def get_relief()
```

Returns :	the relief style

The `get_relief()` method retrieves the current relief style (the "relief" property) set for the button.

gtk.Button.set_label

```
    def set_label(label)
```

`label` :	a string to be set as the text in the button label

The `set_label()` method sets the text of the button label to *`label`* (also sets the "label" property). This string is also used to select the stock item if the "use_stock" property is `TRUE` and the string references a stock item. Any previously set labels will be cleared.

gtk.Button.get_label

```
    def get_label()
```

Returns :	the text of the label widget.

The `get_label()` method retrieves the text from the label of the button, as set by `set_label()` or by the `gtk.Button()` constructor. This string is owned by the widget and must not be modified or freed. If the label text has not been set the return value will be `None`. This will be the case if you create an empty button with `gtk.Button()` to use as a container.

gtk.Button.set_use_underline

```
    def set_use_underline(use_underline)
```

`use_underline` :	`TRUE` if underscores in the text indicate mnemonics

The `set_use_underline()` method sets the "use_underline" property to the value of *`use_underline`*. If *`use_underline`* is `TRUE`, an underscore in the text of the button label indicates that the next character should be underlined and used for the mnemonic accelerator key if it is also the first underlined character.

gtk.Button.get_use_underline

```
    def get_use_underline()
```

Returns :	`TRUE` if an underscore in the button label indicates the mnemonic accelerator keys.

The `get_use_underline()` method returns whether the "use_underline" property is `TRUE` meaning that an underscore in the button label indicates a mnemonic. See `set_use_underline()`.

gtk.Button.set_use_stock

```
    def set_use_stock(use_stock)
```

`use_stock` :	If `TRUE` the button should use a stock item

The `set_use_stock()` method sets the "use_stock" property to the value of *`use_stock`*. If *`use_stock`* is `TRUE`, the label set on the button is used as a stock id to select the stock item for the button.

gtk.Button.get_use_stock

```
def get_use_stock()
```

Returns :	the value of the "use_stock" property.

The `get_use_stock()` method returns the value of the "use_stock" property. If `TRUE` the button label is used to select a stock item instead of being used directly as the label text.

gtk.Button.set_focus_on_click

```
def set_focus_on_click(focus_on_click)
```

`focus_on_click` :	If `TRUE` the button grabs focus when clicked with the mouse.

Note

This method is available in PyGTK 2.4 and above.

The `set_focus_on_click()` method sets the "focus–on–click" property to the value of *`focus_on_click`*. If *`focus_on_click`* is `TRUE`, the button grabs focus when it is clicked by the mouse.

gtk.Button.get_focus_on_click

```
def get_focus_on_click()
```

Returns :	the value of the "focus–on–click" property.

Note

This method is available in PyGTK 2.4 and above.

The `get_focus_on_click()` method returns the value of the "focus–on–click" property. If `TRUE` the button grabs focus when it is clicked by the mouse .

gtk.Button.set_alignment

```
def set_alignment(xalign, yalign)
```

`xalign` :	the horizontal alignment of the child widget. The value ranges from 0.0 to 1.0 and represents the fraction of freespace to the left of the child widget.
`yalign` :	the vertical alignment of the child widget. The value ranges from 0.0 to 1.0 and represents the fraction of freespace above the child widget.

Note

This method is available in PyGTK 2.4 and above.

The `set_alignment()` method sets the "xalign" and "yalign" properties to the value of *`xalign`* and *`yalign`* respectively. This property has no effect unless the child is a `gtk.Misc` or a `gtk.Alignment`.

gtk.Button.get_alignment

```
def get_alignment()
```

Returns :	a 2–tuple containing the values of the "xalign" and "yalign" properties.

Note

This method is available in PyGTK 2.4 and above.

The `get_alignment()` method returns the values of the "xalign" and "yalign" properties. See the `set_alignment()` method for more information.

Signals

The "activate" gtk.Button Signal

```
  def callback(button, user_param1, ...)
```

button :	the button that received the "activate" signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "activate" signal is emitted when the `gtk.Widget.activate()` method is called. For a button it causes the "clicked" signal to be emitted.

The "clicked" gtk.Button Signal

```
  def callback(button, user_param1, ...)
```

button :	the button that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "clicked" signal is emitted when the mouse button is pressed and released while the pointer is over the button.

The "enter" gtk.Button Signal

```
  def callback(button, user_param1, ...)
```

button :	the button that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "enter" signal is emitted when the pointer enters the button.

The "leave" gtk.Button Signal

```
  def callback(button, user_param1, ...)
```

button :	the button that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "leave" signal is emitted when the pointer leaves the button.

The "pressed" gtk.Button Signal

```
    def callback(button, user_param1, ...)
```

button :	the button that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "pressed" signal is emitted when the mouse button is pressed while the pointer is over the button.

The "released" gtk.Button Signal

```
    def callback(button, user_param1, ...)
```

button :	the button that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "released" signal is emitted when the button is released no matter where the pointer is.

Prev Up Next

gtk.Box Home gtk.ButtonBox

gtk.ButtonBox

Prev **The gtk Class Reference** Next

gtk.ButtonBox

gtk.ButtonBox — the base class for widgets that contain multiple buttons

Synopsis

```
class gtk.ButtonBox(gtk.Box):
    def get_layout()
    def set_layout(layout_style)
    def get_child_secondary(child)
    def set_child_secondary(child, is_secondary)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Box
          +-- gtk.ButtonBox
```

Properties

"layout-style"	Read/Write	the style used to layout the buttons in the box. The layout styles are: `gtk.BUTTONBOX_SPREAD`, `gtk.BUTTONBOX_EDGE`, `gtk.BUTTONBOX_START` and `gtk.BUTTONBOX_END`.

Child Properties

"secondary"	Read/Write	If `TRUE`, the child appears in a secondary group of children, suitable for, e.g., help buttons.

Style Properties

"child−min−width"	Read	The minimum width of buttons inside the box.
"child−min−height"	Read	The minimum height of buttons inside the box.
"child−internal−pad−y"	Read	The amount of padding that increases a child's size on the top and bottom
"child−internal−pad−x"	Read	The amount of padding that increases a child's size on the top and bottom

Description

A button box is used to provide a consistent layout of buttons throughout an application. The `gtk.ButtonBox` is subclassed by the `gtk.HButtonBox` and `gtk.VButtonBox` classes to provide horizontal and vertical button layouts respectively. The same effect could be achieved by using a `gtk.Box` but less conveniently.

A buttonbox provides one default layout and a default spacing value that are persistent across all widgets. The layout/spacing can then be altered by the programmer, or if desired, by the user to alter the 'feel' of a program to a small degree.

The `gtk.Box.get_spacing()` method and the `gtk.Box.set_spacing()` methods retrieve and change the default number of pixels between buttons. The `get_layout()` and `set_layout()` methods retrieve and alter the style used to spread the buttons in a button box across the container. The layout styles are: `gtk.BUTTONBOX_SPREAD`, `gtk.BUTTONBOX_EDGE`, `gtk.BUTTONBOX_START` and `gtk.BUTTONBOX_END`.

Methods

gtk.ButtonBox.get_layout

```
    def get_layout()
```

Returns :	the layout style used by the buttonbox

The `get_layout()` method returns the current value of the "layout−style" property. The possible values are: `gtk.BUTTONBOX_SPREAD`, `gtk.BUTTONBOX_EDGE`, `gtk.BUTTONBOX_START` and `gtk.BUTTONBOX_END`.

gtk.ButtonBox.set_layout

```
    def set_layout(layout_style)
```

`layout_style` :	the new layout style

The `set_layout()` method sets the "layout−style" property to the value in *`layout_style`*. The possible values are: `gtk.BUTTONBOX_SPREAD`, `gtk.BUTTONBOX_EDGE`, `gtk.BUTTONBOX_START` and `gtk.BUTTONBOX_END`.

gtk.ButtonBox.get_child_secondary

```
    def get_child_secondary(child)
```

child :	a child button of the buttonbox
Returns :	if TRUE, the *child* appears in a secondary group of the button box.

Note

This method is available in PyGTK 2.4 and above.

The `get_child_secondary()` method returns `TRUE` if *child* should appear in a secondary group of children. See the set_child_secondary() method for more information.

gtk.ButtonBox.set_child_secondary

```
    def set_child_secondary(child, is_secondary)
```

child :	a child button of the buttonbox
is_secondary :	if TRUE, the *child* appears in a secondary group of the button box.

The `set_child_secondary()` method sets whether *child* should appear in a secondary group of children. The typical use of a secondary child is the help button in a dialog that is displayed away from the main group of buttons e.g. right aligned.

The secondary group appears after the other children if the style is `gtk.BUTTONBOX_START`, `gtk.BUTTONBOX_SPREAD` or `gtk.BUTTONBOX_EDGE`, and before the the other children if the style is `gtk.BUTTONBOX_END`. For horizontal button boxes, the definition of before/after depends on direction of the widget (see `gtk.Widget.set_direction()`). If the style is `gtk.BUTTONBOX_START` or `gtk.BUTTONBOX_END`, then the secondary children are aligned at the other end of the button box from the main children. For the other styles, they appear immediately next to the main children.

Prev Up Next
gtk.Button Home gtk.Calendar

gtk.Calendar
Prev **The gtk Class Reference** Next

gtk.Calendar

gtk.Calendar — a widget that displays a calendar and allows the user to select a date.

Synopsis

```
class gtk.Calendar(gtk.Widget):
    gtk.Calendar()
    def select_month(month, year)
    def select_day(day)
    def mark_day(day)
    def unmark_day(day)
    def clear_marks()
    def get_display_options()
    def set_display_options(flags)
    def display_options(flags)
    def get_date()
    def freeze()
    def thaw()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Calendar
```

Properties

Note

These properties are available in GTK+ 2.4 and above.

"day"	Read–Write	The selected day (as a number between 1 and 31, or 0 to unselect the currently selected day).
"month"	Read–Write	The selected month (as a number between 0 and 11).
"no–month–change"	Read–Write	If `TRUE` the selected month cannot be changed.
"show–day–names"	Read–Write	If `TRUE`, day names are displayed.
"show–heading"	Read–Write	If `TRUE`, a heading is displayed.
"show–week–numbers"	Read–Write	If `TRUE`, week numbers are displayed.
"year"	Read–Write	The selected year.

Signal Prototypes

"day–selected"	`def callback(calendar, user_param1, ...)`
"day–selected–double–click"	`def callback(calendar, user_param1, ...)`
"month–changed"	`def callback(calendar, user_param1, ...)`
"next–month"	`def callback(calendar, user_param1, ...)`
"next–year"	`def callback(calendar, user_param1, ...)`
"prev–month"	`def callback(calendar, user_param1, ...)`
"prev–year"	`def callback(calendar, user_param1, ...)`

Description

The `gtk.Calendar` is a widget that displays a simple calendar, one month at a time. In addition, the calendar can display the days of the week and navigation controls that allow a user to change the month and year displayed by calling the `display_options()` (`set_display_options()` in PyGTK 2.4 and above) method. The possible display options are:

`gtk.CALENDAR_SHOW_HEADING`	Specifies that the month and year should be displayed.
`gtk.CALENDAR_SHOW_DAY_NAMES`	Specifies that three letter day descriptions should be present.
`gtk.CALENDAR_NO_MONTH_CHANGE`	Prevents the user from switching months with the calendar.
`gtk.CALENDAR_SHOW_WEEK_NUMBERS`	Displays each week numbers of the current year, down the left side of the calendar.
`gtk.CALENDAR_WEEK_START_MONDAY`	Starts the calendar week on Monday, instead of the default Sunday.

The month and year currently displayed are programatically changed by calling the `select_month()` method. The exact day is selected from the displayed month using the `select_day()` method.

To place a visual marker on a particular day, use the `mark_day()` method and to remove the marker, the `unmark_day()` method. All marks are cleared by calling the `clear_marks()` method.

The selected date can be retrieved from a `gtk.Calendar` using the `get_date()` method. If performing many 'mark' operations, the calendar can be frozen to prevent flicker, using the `freeze()` method, and 'thawed' again using the `thaw()` method.

Constructor

```
gtk.Calendar()
```

Returns :	a calendar object

Creates a calendar object that displays the current month and year with the current day selected. The default calendar display style is: gtk.CALENDAR_SHOW_HEADING | gtk.CALENDAR_SHOW_DAY_NAMES that shows the days of the week and the month and year heading with navigation controls.

Methods

gtk.Calendar.select_month

```
def select_month(month, year)
```

month :	the new month number between 0 and 11
year :	the new year number
Returns :	TRUE if the month is set

The `select_month()` method changes the calendar display to the specified *month* and *year*.

gtk.Calendar.select_day

```
def select_day(day)
```

day :	the new day number between 1 and 31 – 0 removes the current selection

The `select_day()` method selects the specified *day* on the calendar when *day* has a value between 1 and 31. If *day* is 0 then the current day selection is removed.

gtk.Calendar.mark_day

```
def mark_day(day)
```

day :	the number of the day to be marked
Returns :	TRUE

The `mark_day()` method marks the specified month *day* with a visual marker (typically by making the number bold). If the calendar month and year are changed the marked days remain marked.

gtk.Calendar.unmark_day

```
    def unmark_day(day)
```

day :	the number of the day to be unmarked
Returns :	TRUE

The `unmark_day`() method unmarks the specified month *`day`*.

gtk.Calendar.clear_marks

```
    def clear_marks()
```

The `clear_marks`() method clears all marked days.

gtk.Calendar.get_display_options

```
    def get_display_options()
```

Returns :	the calendar display options

Note

This method is available in PyGTK 2.4 and above.

The `get_display_options`() method returns the current calendar display options. See the `set_display_options()` method for more information.

gtk.Calendar.set_display_options

```
    def set_display_options(flags)
```

flags :	the new calendar display options

Note

This method is available in PyGTK 2.4 and above.

The `set_display_options`() method sets the calendar display options to the value specified by *`flags`*. The possible display options are a combination of:

`gtk.CALENDAR_SHOW_HEADING`	Specifies that the month and year should be displayed.
`gtk.CALENDAR_SHOW_DAY_NAMES`	Specifies that three letter day descriptions should be present.
`gtk.CALENDAR_NO_MONTH_CHANGE`	Prevents the user from switching months with the calendar.
`gtk.CALENDAR_SHOW_WEEK_NUMBERS`	Displays each week numbers of the current year, down the left side of the calendar.
`gtk.CALENDAR_WEEK_START_MONDAY`	Starts the calendar week on Monday, instead of the default Sunday.

The display options can be removed by passing 0 as the value of *`flags`*.

gtk.Calendar.display_options

```
    def display_options(flags)
```

flags :	the new calendar display options

Warning

This method is deprecated in PyGTK 2.4 and should be replaced by the `set_display_options()` method.

The `display_options()` method sets the calendar display options to the value specified by *`flags`*. The possible display options are a combination of:

`gtk.CALENDAR_SHOW_HEADING`	Specifies that the month and year should be displayed.
`gtk.CALENDAR_SHOW_DAY_NAMES`	Specifies that three letter day descriptions should be present.
`gtk.CALENDAR_NO_MONTH_CHANGE`	Prevents the user from switching months with the calendar.
`gtk.CALENDAR_SHOW_WEEK_NUMBERS`	Displays each week numbers of the current year, down the left side of the calendar.
`gtk.CALENDAR_WEEK_START_MONDAY`	Starts the calendar week on Monday, instead of the default Sunday.

The display options can be removed by passing 0 as the value of *`flags`*.

gtk.Calendar.get_date

```
    def get_date()
```

Returns :	a tuple containing the year, month and day

The `get_date()` method retrieves the calendar's current year, month and selected day numbers as a tuple (year, month, day).

gtk.Calendar.freeze

```
    def freeze()
```

The `freeze()` method stops the update of the calendar display until the `thaw()` method is called. This method is used to reduce calendar flicker when doing a large number of updates to the calendar.

gtk.Calendar.thaw

```
    def thaw()
```

The `thaw()` method reenables the update of the calendar after a `freeze()` method is called. All changes made since the last `freeze()` are displayed

Signals

The "day−selected" gtk.Calendar Signal

```
    def callback(calendar, user_param1, ...)
```

`calendar` :	the calendar that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "day−selected" signal is emitted when a day is selected either by the user or programatically.

The "day−selected−double−click" gtk.Calendar Signal

```
    def callback(calendar, user_param1, ...)
```

calendar :	the calendar that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "day−selected−double−click" signal is emitted when a calendar day is doubled−clicked by the user.

The "month−changed" gtk.Calendar Signal

```
    def callback(calendar, user_param1, ...)
```

calendar :	the calendar that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "month−changed" signal is emitted when the calendar month is changed programatically or by the user.

The "next−month" gtk.Calendar Signal

```
    def callback(calendar, user_param1, ...)
```

calendar :	the calendar that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "next−month" signal is emitted when the user clicks the "next−month" navigation control in the calendar header.

The "next−year" gtk.Calendar Signal

```
    def callback(calendar, user_param1, ...)
```

calendar :	the calendar that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "next−year" signal is emitted when the user clicks the "next−year" navigation control in the calendar header.

The "prev−month" gtk.Calendar Signal

```
    def callback(calendar, user_param1, ...)
```

calendar :	the calendar that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "prev−month" signal is emitted when the user clicks the "prev−month" navigation control in the calendar header.

The "day−selected−double−click" gtk.Calendar Signal 218

The "prev−year" gtk.Calendar Signal

```
  def callback(calendar, user_param1, ...)
```

calendar :	the calendar that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "prev−year" signal is emitted when the user clicks the "prev−year" navigation control in the calendar header.

gtk.CellEditable

gtk.CellEditable — an interface for editing a TreeView cell

Synopsis

```
class gtk.CellEditable(gobject.GInterface):
    def start_editing(event)
    def editing_done()
    def remove_widget()
```

Signal Prototypes

"editing−done"	def callback(*celleditable*, *user_param1*, *...*)
"remove−widget"	def callback(*celleditable*, *user_param1*, *...*)

Description

The `gtk.CellEditable` is an interface that provides editing of a cell in a `gtk.TreeView` cell.

Methods

gtk.CellEditable.start_editing

```
  def start_editing(event)
```

event :	A `gtk.gdk.Event`, or `None`

The `start_editing()` method begins the editing on a *cell_editable* widget that has been reparented over the treeview cell. *event* is the `gtk.gdk.Event` that began the editing process. If the editing was initiated through programmatic means, *event* may be `None`, .

gtk.CellEditable.editing_done

```
def editing_done()
```

The `editing_done()` method emits the "editing_done" signal that notifies the cell renderer to update it's value from the cell.

gtk.CellEditable.remove_widget

```
def remove_widget()
```

The remove_widget() method emits the "remove_widget" signal that indicates that the cell is finished editing, and the celleditable widget may now be destroyed.

Signals

The "editing−done" gtk.CellEditable Signal

```
def callback(celleditable, user_param1, ...)
```

celleditable :	the celleditable that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "editing−done" signal is emitted when the `editing_done()` method is called.

The "remove−widget" gtk.CellEditable Signal

```
def callback(celleditable, user_param1, ...)
```

celleditable :	the celleditable that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "remove−widget" signal is emitted when the cellrenderer for the treeview cell has retrieved the edited information and the celleditable widget can be destroyed.

Prev	Up	Next
gtk.Calendar	Home	gtk.CellLayout

gtk.CellLayout

gtk.CellLayout — an interface for packing cells

Synopsis

```
class gtk.CellLayout(gobject.GInterface):
    def pack_start(cell, expand=TRUE)
    def pack_end(cell, expand=TRUE)
    def clear()
    def set_attributes(cell, ...)
    def add_attribute(cell, attribute, column)
    def set_cell_data_func(cell, func, func_data)
    def clear_attributes(cell)
    def reorder(cell, position)
```

Description

Note

This interface is available in PyGTK 2.4 and above.

`gtk.CellLayout` is an interface to be implemented by all objects that want to provide a `gtk.TreeViewColumn`–like API for packing cells, setting attributes and data funcs. The `gtk.CellLayout` interface is implemented by the `gtk.ComboBoxEntry`, `gtk.ComboBox`, `gtk.EntryCompletion` and `gtk.TreeViewColumn` widgets.

Methods

gtk.CellLayout.pack_start

```
    def pack_start(cell, expand=TRUE)
```

cell :	A `gtk.CellRenderer`.
expand :	if `TRUE`, the *cell* is to be given extra space that is allocated to the cell layout.

Note

This method is available in PyGTK 2.4 and above.

The `pack_start()` method packs the `gtk.CellRenderer` specified by *cell* into the beginning of the cell layout. If the optional parameter *expand* is `FALSE`, then *cell* is allocated no more space than it needs. Any unused space is divided evenly between cells for which *expand* is `TRUE`.

gtk.CellLayout.pack_end

```
    def pack_end(cell, expand=TRUE)
```

cell :	A `gtk.CellRenderer`.
expand :	if `TRUE`, the *cell* is to be given extra space that is allocated to the cell layout.

Note

This method is available in PyGTK 2.4 and above.

The `pack_end()` method adds the `gtk.CellRenderer` specified by *cell* to the end of the cell layout. If the optional parameter *expand* is `FALSE`, then the *cell* is allocated no more space than it needs. Any unused space is divided evenly between cells for which *expand* is `TRUE`.

gtk.CellLayout.clear

```
def clear()
```

Note

This method is available in PyGTK 2.4 and above.

The `clear`() method unsets all the attribute mappings on all cell renderers in the cell layout.

gtk.CellLayout.set_attributes

```
def set_attributes(cell, ...)
```

`cell` :	A `gtk.CellRenderer`.
`...` :	Zero or more keyword−value arguments in the format attribute=column.

Note

This method is available in PyGTK 2.4 and above.

The `set_attributes`() method sets the attributes provided as a keyword argument list as the attributes of the `gtk.CellRenderer` specified by *`cell`*. The attributes should be supplied as keyword−value arguments in the format: attribute=column (e.g. text=0, background=1). All existing attributes are removed, and replaced with the new attributes.

gtk.CellLayout.add_attribute

```
def add_attribute(cell, attribute, column)
```

`cell` :	A `gtk.CellRenderer`.
`attribute` :	An attribute on the renderer.
`column` :	The column number in the model to get the attribute from.

Note

This method is available in PyGTK 2.4 and above.

The `add_attribute`() method adds an attribute mapping to the list in the cell layout. The *`column`* parameter is the column of the model to get a value from, and the *`attribute`* parameter is the attribute of *`cell`* to be set from the value. So for example if column 2 of the model contains strings, you could have the "text" attribute of a `gtk.CellRendererText` get its values from column 2.

gtk.CellLayout.set_cell_data_func

```
def set_cell_data_func(cell, func, func_data)
```

`cell` :	A `gtk.CellRenderer`.
`func` :	The function to use.
`func_data` :	The user data for *`func`*.

Note

This method is available in PyGTK 2.4 and above.

The `set_cell_data_func()` method sets the function (or method) specified by *`func`* to be used for setting the column value of the `gtk.CellRenderer` specified by *`cell`* instead of using the standard attribute mapping method. *`func`* may be `None` to remove the current function. The signature of *`func`* is:

```
    def celldatafunction(celllayout, cell, model, iter, user_data)

    def celldatamethod(self, celllayout, cell, model, iter, user_data)
```

where *`celllayout`* is the `gtk.CellLayout`, *`cell`* is the `gtk.CellRenderer` for *`celllayout`*, *`model`* is the `gtk.TreeModel` and *`iter`* is the `gtk.TreeIter` pointing at the row.

gtk.CellLayout.clear_attributes

```
  def clear_attributes(cell)
```

`cell` :	A `gtk.CellRenderer` to clear the attribute mapping on.

Note

This method is available in PyGTK 2.4 and above.

The `clear_attributes()` method clears all existing attribute mappings from the `gtk.CellRenderer` specified by *`cell`* previously set with the `set_attributes()` or `add_attribute()` methods.

gtk.CellLayout.reorder

```
  def reorder(cell, position)
```

`cell` :	A `gtk.CellRenderer` to reorder.
`position` :	New position to insert *`cell`* at.

Note

This method is available in PyGTK 2.4 and above.

The `reorder()` method re–inserts the `gtk.CellRenderer` specified by *`cell`* at *`position`*. Note that *`cell`* has to already be packed into *`cell_layout`* for this to function properly.

Prev Up Next
gtk.CellEditable Home gtk.CellRenderer

gtk.CellRenderer
Prev **The gtk Class Reference** Next

gtk.CellRenderer

gtk.CellRenderer — a base class for objects that render into Treeview cells

Synopsis

```
class gtk.CellRenderer(gtk.Object):
    def get_size(widget, cell_area=None)
    def render(window, widget, background_area, cell_area, expose_area, flags)
    def activate(event, widget, path, background_area, cell_area, flags)
    def start_editing(event, widget, path, background_area, cell_area, flags)
    def editing_canceled()
    def stop_editing(canceled)
    def set_fixed_size(width, height)
    def get_fixed_size()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.CellRenderer
```

Properties

"mode"	Read/Write	The editable mode of the cellrenderer. One of: `gtk.CELL_RENDERER_MODE_INERT`, `gtk.CELL_RENDERER_MODE_ACTIVATABLE` or `gtk.CELL_RENDERER_MODE_EDITABLE`
"visible"	Read/Write	If TRUE the cell is displayed
"xalign"	Read/Write	The fraction of *free* space to the left of the cell in the range 0.0 to 1.0.
"yalign"	Read/Write	The fraction of *free* space above the cell in the range 0.0 to 1.0.
"xpad"	Read/Write	The amount of padding to the left and right of the cell.
"ypad"	Read/Write	The amount of padding above and below cell.
"width"	Read/Write	The fixed width of the cell.
"height"	Read/Write	The fixed height of the cell.
"is–expander"	Read/Write	If TRUE the row has children
"is–expanded"	Read/Write	If TRUE the row has children and it is expanded to show the children.
"cell–background"	Write	The background color of the cell as a string.
"cell–background–gdk"	Read/Write	The background color of the cell as a `gtk.gdk.Color`.
"cell–background–set"	Read/Write	If TRUE the cell background color is set by this cellrenderer

Signal Prototypes

"editing–canceled"	`def callback(cellrenderer, user_param1, ...)`
"editing–started"	`def callback(cellrenderer, editable, path, user_param1, ...)`

Description

The `gtk.CellRenderer` is a base class for a set of objects used for rendering a cell to a `gtk.gdk.Drawable`. The `gtk.CellRenderer` provides the common attributes and methods for its subclasses (`gtk.CellRendererPixbuf`, `gtk.CellRendererText` and `gtk.CellRendererToggle`).

The primary use of a `gtk.CellRenderer` is for drawing a certain graphical elements on a `gtk.gdk.Drawable`. Typically, one cell renderer is used to draw many cells on the screen. To this extent, it isn't expected that a CellRenderer keep any permanent state around. Instead, any state is set just prior to use using the GObjects property system. Then, the cell is measured using the `get_size()` method. Finally, the cell is rendered in the correct location using the `render()` method.

Methods

gtk.CellRenderer.get_size

```
def get_size(widget, cell_area)
```

`widget` :	the widget the renderer is rendering to
`cell_area` :	The area a cell will be allocated, or `None`
Returns :	a tuple containing the xoffset, yoffset, width and height

The `get_size`() method obtains the width and height needed to render the cell. These values are returned as part of a tuple containing the x_offset, y_offset, width and height. `get_size`() is used by view widgets to determine the appropriate size for the *`cell_area`* to be passed to the `gtk.CellRenderer.render()` method. If *`cell_area`* is not `None`, the x and y offsets of the cell relative to this location are returned. Please note that the values set in the returned width and height, as well as those in x_offset and y_offset are inclusive of the xpad and ypad properties.

gtk.CellRenderer.render

```
def render(window, widget, background_area, cell_area, expose_area, flags)
```

`window` :	a `gtk.gdk.Drawable` to draw to
`widget` :	the widget owning *`window`*
`background_area` :	entire cell area (including tree expanders and maybe padding on the sides)
`cell_area` :	area normally rendered by a cell renderer
`expose_area` :	area that actually needs updating
`flags` :	flags that affect rendering

The `render`() method invokes the virtual render function of the `gtk.CellRenderer`. The three passed–in rectangles are areas of *`window`*. Most renderers will draw within *`cell_area`*; the xalign, yalign, xpad, and ypad properties of the `gtk.CellRenderer` should be honored with respect to *`cell_area`*. *`background_area`* includes the blank space around the cell, and also the area containing the tree expander; so the *`background_area`* rectangles for all cells tile to cover the entire *`window`*. *`expose_area`* is a clip rectangle.

The *`flags`* value is one of: `gtk.CELL_RENDERER_SELECTED`, `gtk.CELL_RENDERER_PRELIT`, `gtk.CELL_RENDERER_INSENSITIVE` or `gtk.CELL_RENDERER_SORTED`

gtk.CellRenderer.activate

```
def activate(event, widget, path, background_area, cell_area, flags)
```

`event` :	a `gtk.gdk.Event`
`widget` :	widget that received the event
`path` :	widget–dependent string representation of the event location; e.g. for `gtk.TreeView`, a string representation of `gtk.TreePath`

background_area :	background area as passed to render()
cell_area :	cell area as passed to render()
flags :	render flags
Returns :	`TRUE` if the event was consumed/handled

The `activate`() method passes an activate event to the cell renderer for possible processing. Some cell renderers may use events; for example, `gtk.CellRendererToggle` toggles when it gets a mouse click.

The *flags* value is one of: `gtk.CELL_RENDERER_SELECTED`, `gtk.CELL_RENDERER_PRELIT`, `gtk.CELL_RENDERER_INSENSITIVE` or `gtk.CELL_RENDERER_SORTED`

gtk.CellRenderer.start_editing

```
    def start_editing(event, widget, path, background_area, cell_area, flags)
```

event :	a `gtk.gdk.Event`
widget :	the widget that received the event
path :	a widget–dependent string representation of the event location; e.g. for `gtk.TreeView`, a string representation of `gtk.TreePath`
background_area :	background area as passed to render().
cell_area :	cell area as passed to render()
flags :	render flags
Returns :	A new `gtk.CellEditable`, or `None`

The `start_editing`() method initiates the editing of a cell.

gtk.CellRenderer.editing_canceled

```
    def editing_canceled()
```

Note

This method is available in PyGTK 2.4 and above.

Warning

This method is deprecated in PyGTK 2.6 and above. Use the stop_editing() method instead.

The `editing_canceled`() method causes the cell renderer to emit the "editing–canceled" signal. This method is for use only by implementations of cell renderers that need to notify the client program that an editing process was canceled and the changes were not committed.

gtk.CellRenderer.stop_editing

```
    def stop_editing(canceled)
```

canceled :	if `TRUE` the editing has been canceled

Note

This method is available in PyGTK 2.6 and above.

The `stop_editing`() method informs the cell renderer that the editing is stopped. If *canceled* is `TRUE`, the cell renderer will emit the "editing–canceled" signal. This method should be called by cell renderer implementations in response to the "editing–done" signal of gtk.CellEditable.

gtk.CellRenderer.set_fixed_size

```
def set_fixed_size(width, height)
```

width :	the width of the cell renderer, or –1
height :	the height of the cell renderer, or –1

The set_fixed_size() method sets the renderer size to the specified *width* and *height*, independent of the properties set.

gtk.CellRenderer.get_fixed_size

```
def get_fixed_size()
```

Returns :	a tuple containing the width and height of the cell

The get_fixed_size() method retrieves a tuple containing the fixed *width* and *height* of the cell.

Signals

The "editing–canceled" gtk.CellRenderer Signal

```
def callback(cellrenderer, user_param1, ...)
```

cellrenderer :	the cellrenderer that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.4 and above.

The "editing–canceled" signal is emitted when the user cancels the process of editing a cell. For example, an editable cell renderer could be written to cancel editing when the user presses Escape. Also see the editing_canceled() method.

The "editing–started" gtk.CellRenderer Signal

```
def callback(cellrenderer, editable, path, user_param1, ...)
```

cellrenderer :	the cellrenderer that received the signal
editable :	the gtk.CellEditable
path :	he path identifying the edited cell
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.6 and above.

The "editing−started" signal is emitted when a cell starts to be edited. The intended use of this signal is to do special setup on *editable*, e.g. adding a `gtk.EntryCompletion` or setting up additional columns in a `gtk.ComboBox`.

Note that GTK+ doesn't guarantee that cell renderers will continue to use the same kind of widget for editing in future releases, therefore you should check the type of editable before doing any specific setup, as in the following example:

```
def text_editing_started(cell, editable, path, data):
  if isinstance(editable, gtk.Entry):
      # ... create a GtkEntryCompletion
      completion = gtk.EntryCompletion()
      editable.set_completion(completion)
      ...
  ...
```

Prev	Up	Next
gtk.CellLayout	Home	gtk.CellRendererCombo

gtk.CellRendererCombo

Prev The gtk Class Reference Next

gtk.CellRendererCombo

gtk.CellRendererCombo — an object that renders a `gtk.ComboBoxEntry` into a `gtk.TreeView` cell (new in PyGTK 2.6)

Synopsis

```
class gtk.CellRendererCombo(gtk.CellRendererText):
    gtk.CellRendererCombo()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.CellRenderer
      +-- gtk.CellRendererText
        +-- gtk.CellRendererCombo
```

Properties

"has−entry"	Read/Write	If `FALSE`, don't allow entering strings other than the given ones. Default value: `TRUE`. Available in GTK+ 2.6 and above.
"model"	Read/Write	The tree model containing the possible values for the combo box entry. Available in GTK+ 2.6 and above.
"text−column"	Read/Write	A column in the data source model to get the strings from. Allowed values: >= −1. Default value: −1. Available in GTK+ 2.6 and above.

Description

Note

This object is available in PyGTK 2.6 and above.

The `gtk.CellRendererCombo` manages the rendering of a `gtk.ComboBoxEntry` into a `gtk.TreeView` cell.

Constructor

```
  gtk.CellRendererCombo()
```

Returns :	the new cell renderer

Note

This constructor is available in PyGTK 2.6 and above.

Creates a new `gtk.CellRendererCombo`. Rendering parameters are adjusted using the object properties. The object properties can be set globally (with `set_property()`). Also, with `gtk.TreeViewColumn`, you can bind a property to a value in a `gtk.TreeModel`. For example, you can bind the "text" property on the cell renderer to a string value in the model, thus rendering a different string in each row of the `gtk.TreeView`.

Prev	Up	Next
gtk.CellRenderer	Home	gtk.CellRendererPixbuf

gtk.CellRendererPixbuf

Prev **The gtk Class Reference** Next

gtk.CellRendererPixbuf

gtk.CellRendererPixbuf — an object that renders a pixbuf into a `gtk.TreeView` cell

Synopsis

```
class gtk.CellRendererPixbuf(gtk.CellRenderer):
    gtk.CellRendererPixbuf()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.CellRenderer
      +-- gtk.CellRendererPixbuf
```

Properties

"pixbuf"	Read/Write	The pixbuf to render.
"pixbuf–expander–open"	Read/Write	Pixbuf for open expander.
"pixbuf–expander–closed"	Read/Write	Pixbuf for closed expander.
"stock–id"	Read/Write	The stock ID of the stock icon to render
"stock–size"	Read/Write	The size of the rendered icon
"stock–detail"	Read/Write	Render detail to pass to the theme engine

Description

The `gtk.CellRendererPixbuf` manages the rendering of a pixbuf into a `gtk.TreeView` cell.

Constructor

```
    gtk.CellRendererPixbuf()
```

Returns :	the new cell renderer

Creates a new `gtk.CellRendererPixbuf`. Rendering parameters are adjusted using the object properties. The object properties can be set globally (with `set_property()`). Also, with `gtk.TreeViewColumn`, you can bind a property to a value in a `gtk.TreeModel`. For example, you can bind the "pixbuf" property on the cell renderer to a pixbuf value in the model, thus rendering a different image in each row of the `gtk.TreeView`.

gtk.CellRendererProgress

gtk.CellRendererProgress   an object that renders numbers as progress bars in a `gtk.TreeView` (new in PyGTK 2.6)

Synopsis

```
class gtk.CellRendererProgress(gtk.CellRenderer):
    gtk.CellRendererProgress()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.CellRenderer
      +-- gtk.CellRendererProgress
```

Properties

"text"	Read/Write	The text in the label that will be drawn over the progress bar. Setting this property to `None` causes the default label to be displayed. Setting this property to an empty string

		causes no label to be displayed. Default value: `None`. Available in GTK+ 2.6 and above.
"value"	Read/Write	The percentage that the progress bar is "filled in". Available in GTK+ 2.6 and above.

Description

Note

This object is available in PyGTK 2.6 and above.

The `gtk.CellRendererProgress` manages the rendering of a number as a progress bar in a `gtk.TreeView` cell.

Constructor

```
gtk.CellRendererProgress()
```

Returns :	the new cell renderer

Note

This constructor is available in PyGTK 2.6 and above.

Creates a new `gtk.CellRendererProgress`. Rendering parameters are adjusted using the object properties. The object properties can be set globally (with `set_property()`). Also, with `gtk.TreeViewColumn`, you can bind a property to a value in a `gtk.TreeModel`. For example, you can bind the "text" property on the cell renderer to a string value in the model, thus rendering a different string in each row of the `gtk.TreeView`.

gtk.CellRendererText

gtk.CellRendererText — an object that renders text into a `gtk.TreeView` cell

Synopsis

```
class gtk.CellRendererText(gtk.CellRenderer):
    gtk.CellRendererText()
    def set_fixed_height_from_font(number_of_rows)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.CellRenderer
      +-- gtk.CellRendererText
```

Properties

"attributes"	Read/Write	A list of style attributes to apply to the text of the renderer.
"background"	Write	Background color as a string. Default: `None`
"background−gdk"	Read/Write	Background color as a gtk.gdk.Color
"background−set"	Read/Write	If `TRUE` this tag affects the background color. Default: `FALSE`
"editable"	Read/Write	If `TRUE` the text can be modified by the user. GTK+ 2.4 and above. Default: `FALSE`
"editable−set"	Read/Write	If `TRUE` this tag affects the text editability. GTK+ 2.4 and above. Default: `FALSE`
"ellipsize"	Read/Write	The preferred place to ellipsize the string, if the cell renderer does not have enough room to display the entire string, if at all. GTK+ 2.6 and above. Default: `pango.ELLIPSIZE_NONE`
"ellipsize−set"	Read/Write	If `TRUE` this tag affects the text editability. Default: `FALSE`
"family"	Read/Write	Name of the font family, e.g. Sans, Helvetica, Times, Monospace. Default: `None`
"family−set"	Read/Write	If `TRUE` this tag affects the font family. Default: `FALSE`
"font"	Read/Write	Font description as a string. Default: `None`
"font−desc"	Read/Write	Font description as a `pango.FontDescription`
"foreground"	Write	Foreground color as a string. Default: `None`
"foreground−gdk"	Read/Write	Foreground color as a `gtk.gdk.Color`
"foreground−set"	Read/Write	If `TRUE` this tag affects the foreground color. Default: `FALSE`
"language"	Read/Write	The language this text is in, as an ISO code. Pango can use this as a hint when rendering the text. If you don't understand this parameter, you probably don't need it. GTK+ 2.4 and above. Default: `None`
"language−set"	Read/Write	If `TRUE` this tag affects the language used to render the text. GTK+ 2.4 and above. Default: `FALSE`
"markup"	Write	Marked up text to render. Default: `None`
"rise"	Read/Write	Offset of text above the baseline (below the baseline if rise is negative). Default: 0
"rise−set"	Read/Write	If `TRUE` this tag affects the rise. Default: `FALSE`
"scale"	Read/Write	Font scaling factor. Allowed values >= 0. Default: 1
"scale−set"	Read/Write	If `TRUE` this tag scales the font. Default: `FALSE`
"single−paragraph−mode"	Read/Write	If `TRUE`, keep all text in a single paragraph. GTK+ 2.4 and above. Default: `FALSE`
"size"	Read/Write	Font size. Allowed values >= 0. Default: 0
"size−points"	Read/Write	Font size in points. Allowed values >= 0. Default: 0
"size−set"	Read/Write	If `TRUE` this tag affects the font size. Default: `FALSE`
"stretch"	Read/Write	Font stretch. Default: `pango.STRETCH_NORMAL`
"stretch−set"	Read/Write	If `TRUE` this tag affects the font stretch. Default: `FALSE`
"strikethrough"	Read/Write	If `TRUE` strike through the text. Default: `FALSE`
"strikethrough−set"	Read/Write	If `TRUE` this tag affects the strikethrough. Default: `FALSE`
"style"	Read/Write	Font style. Default: `pango.STYLE_NORMAL`
"style−set"	Read/Write	If `TRUE` this tag affects the font style. Default: `FALSE`
"text"	Read/Write	Text to render. Default: `None`
"underline"	Read/Write	Style of underline for this text. Default:

		`pango.UNDERLINE_NONE`
"underline−set"	Read/Write	If `TRUE` this tag affects the text underlining. Default: `FALSE`
"variant"	Read/Write	Font variant. Default: `pango.VARIANT_NORMAL`
"variant−set"	Read/Write	If `TRUE` this tag affects the font variant. Default: `FALSE`
"weight"	Read/Write	Font weight. Allowed values >= 0. Default value: 400
"weight−set"	Read/Write	If `TRUE` this tag affects the font weight. Default: `FALSE`
"width−chars"	Read/Write	The desired width of the cell, in characters. If this property is set to −1, the width will be calculated automatically, otherwise the cell will request either 3 characters or the property value, whichever is greater. GTK+ 2.6 and above. Allowed values >= −1. Default value: −1

Signal Prototypes

"edited" `def callback(`*`cellrenderertext, path, new_text, user_param1, ...`*`)`

Description

The `gtk.CellRendererText` manages the rendering of text into a `gtk.TreeView` cell.

Constructor

```
gtk.CellRendererText()
```

Returns :	the new cell renderer

Creates a new `gtk.CellRendererText`. The way that text is drawn is changed using object properties. The object properties can be set globally (with `set_property()`). Also, with `gtk.TreeViewColumn`, you can bind a property to a value in a `gtk.TreeModel`. For example, you can bind the "text" property on the cell renderer to a string value in the model, thus rendering a different string in each row of the `gtk.TreeView`.

Methods

gtk.CellRendererText.set_fixed_height_from_font

```
def set_fixed_height_from_font(number_of_rows)
```

`number_of_rows` :	Number of rows of text each cell renderer is allocated, or −1

The `set_fixed_height_from_font()` sets the height of a renderer to explicitly be determined by the "font" and "ypad" properties set on it. This method must be called each time these properties are changed to affect the height. This function is inflexible, and should really only be used if calculating the size of a cell is too slow (i.e. a massive number of cells displayed). If *`number_of_rows`* is −1, then the fixed height is unset, and the height is determined by the properties again.

Signals

The "edited" gtk.CellRendererText Signal

```
    def callback(cellrenderertext, path, new_text, user_param1, ...)
```

cellrenderertext :	the cellrenderertext that received the "edited" signal
path :	the path string of the cellrenderertext
new_text :	the new text of the cellrenderertext
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "edited" signal is emitted when the text in the cell has been edited.

Prev Up Next
gtk.CellRendererProgress Home gtk.CellRendererToggle

gtk.CellRendererToggle
Prev **The gtk Class Reference** Next

gtk.CellRendererToggle

gtk.CellRendererToggle — an object that renders a toggle button into a TreeView cell

Synopsis

```
class gtk.CellRendererToggle(gtk.CellRenderer):
    gtk.CellRendererToggle()
    def get_radio()
    def set_radio(radio)
    def get_active()
    def set_active(setting)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.CellRenderer
      +-- gtk.CellRendererToggle
```

Properties

"activatable"	Read/Write	If `TRUE`, the toggle button can be activated
"active"	Read/Write	If `TRUE`, the button is active.
"radio"	Read/Write	If `TRUE`, draw the toggle button as a radio button
"inconsistent"	Read/Write	If `TRUE`, the button is in an inconsistent state. GTK+ 2.2 and above.

Signal Prototypes

"toggled"	`def callback(cellrenderertoggle, path, user_param1, ...)`

Description

The gtk.CellRendererToggle manages the rendering of toggle button into a gtk.TreeView cell.

Constructor

```
gtk.CellRendererToggle()
```

Returns : the new cell renderer

Creates a new gtk.CellRendererToggle. The toggle button rendering parameters are adjusted using the object properties. The object properties can be set globally (with set_property()). Also, with gtk.TreeViewColumn, you can bind a property to a value in a gtk.TreeModel. For example, you can bind the "active" property on the cell renderer to a boolean value in the model, thus causing the check button to reflect the state of the model.

Methods

gtk.CellRendererToggle.get_radio

```
def get_radio()
```

Returns : `TRUE` if we're rendering radio toggles rather than checkboxes

The `get_radio`() method returns `TRUE` if radio toggles rather than checkboxes are being rendered.

gtk.CellRendererToggle.set_radio

```
def set_radio(radio)
```

`radio` : If `TRUE` make the toggle look like a radio button

The `set_radio`() method sets the style of the toggle button. If *`radio`* is `TRUE`, the cell renderer renders a radio toggle (i.e. a toggle in a group of mutually–exclusive toggles). If `FALSE`, it renders a check toggle (a standalone boolean option). This can be set globally for the cell renderer, or changed just before rendering each cell in the model (for gtk.TreeView, you set up a per–row setting using gtk.TreeViewColumn to associate model columns with cell renderer properties).

gtk.CellRendererToggle.get_active

```
def get_active()
```

Returns : `TRUE` if the cell renderer is active.

The `get_active`() method returns `TRUE` if the cell renderer is active. See gtk.CellRendererToggle.set_active().

gtk.CellRendererToggle.set_active

```
def set_active(setting)
```

`setting` : the value to set.

The `set_active`() method activates a cell renderer if *`setting`* is `TRUE` and or deactivates a cell renderer if *`setting`* is `FALSE`.

Signals

The "toggled" gtk.CellRendererToggle Signal

```
    def callback(cellrenderertoggle, path, user_param1, ...)
```

cellrenderertoggle :	the cellrenderertoggle that received the "toggled" signal
path :	the path of the cellrenderertoggle
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "toggled" signal is emitted when the toggle button in the cell changes state.

Prev	Up	Next
gtk.CellRendererText	Home	gtk.CellView

gtk.CellView

Prev **The gtk Class Reference** Next

gtk.CellView

gtk.CellView a widget displaying a single row of a `gtk.TreeModel` (new in PyGTK 2.6).

Synopsis

```
class gtk.CellView(gtk.Widget):
    gtk.CellView()
    def set_model(model)
    def set_displayed_row(path)
    def get_displayed_row()
    def get_size_of_row(path, requisition)
    def set_background_color(color)
    def get_cell_renderers()
Functions

    def gtk.cell_view_new_with_text(text)
    def gtk.cell_view_new_with_markup(markup)
    def gtk.cell_view_new_with_pixbuf(pixbuf)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.CellView
```

Properties

"background"	Write	The background color as a string. Default value: `None`
"background–gdk"	Read–Write	The background color as a `gtk.gdk.Color`.
"background–set"	Read–Write	If `TRUE`, use the "background–gdk" property to set the background color.

Description

Note

This widget is available in PyGTK 2.6 and above.

The `gtk.CellView` is a widget that displays one row of a `gtk.TreeModel`. `gtk.CellView` implements the `gtk.CellLayout` interface that provides for the addition and management of `gtk.CellRenderer` objects.

Constructor

```
    gtk.CellView()
```

Returns :	a new `gtk.CellView`

Note

This constructor is available in PyGTK 2.6 and above.

Creates a new `gtk.CellView` widget.

Methods

gtk.CellView.set_model

```
    def set_model(model)
```

`model` :	a `gtk.TreeModel` or `None`

Note

This method is available in PyGTK 2.6 and above.

The `set_model`() method sets the `gtk.TreeModel` used by the cell view to that specified by *`model`*. If *`model`* is `None` the cell view's model will be unset.

gtk.CellView.set_displayed_row

```
    def set_displayed_row(path)
```

`path` :	a tree path or `None` to unset the row

Note

This method is available in PyGTK 2.6 and above.

The `set_displayed_row`() method sets the row of the model that is currently displayed to the path specified by *`path`*. If *`path`* is `None` the display row will be unset, and the contents of the cell view "stick" at their last value. This is not normally a desired result, but may be a needed intermediate state if say, the model for the cell view becomes temporarily empty.

gtk.CellView.get_displayed_row

```
    def get_displayed_row()
```

Returns :	the path of the currently displayed row in the cell view's model

Note

This method is available in PyGTK 2.6 and above.

The `get_displayed_row`() method returns the path of the currently displayed row as set by the `set-displayed_row()` method.

gtk.CellView.get_size_of_row

```
    def get_size_of_row(path)
```

`path` :	the path of a row
Returns :	a `gtk.Requisition` containing the required size data

Note

This method is available in PyGTK 2.6 and above.

The `get_size_of_row`() method returns a `gtk.Requisition` containing the size required for displaying the row with the tree path specified by *`path`*.

gtk.CellView.set_background_color

```
    def set_background_color(color)
```

`color` :	

Note

This method is available in PyGTK 2.6 and above.

The `set_background_color`() method sets the background color of the cell view to the `gtk.gdk.Color` specified by *`color`*.

gtk.CellView.get_cell_renderers

```
    def get_cell_renderers()
```

Returns :	a list of the `gtk.CellRenderer` objects of the cell view.

Note

This method is available in PyGTK 2.6 and above.

The `get_cell_renderers`() method returns a list containing the `gtk.CellRenderer` objects used by the cell view.

Functions

gtk.cell_view_new_with_text

```
    def gtk.cell_view_new_with_text(text)
```

text :	a string
Returns :	a new gtk.CellView

Note

This function is available in PyGTK 2.6 and above.

The `gtk.cell_view_new_with_text()` function creates a new gtk.CellView with a gtk.CellRendererText displaying the string specified by *text*.

gtk.cell_view_new_with_markup

```
    def gtk.cell_view_new_with_markup(markup)
```

markup :	a string containing Pango markup to be displayed.
Returns :	a new gtk.CellView

Note

This function is available in PyGTK 2.6 and above.

The `gtk.cell_view_new_with_markup()` function creates a new gtk.CellView with a gtk.CellRendererText displaying the Pango markup specified by *markup*.

gtk.cell_view_new_with_pixbuf

```
    def gtk.cell_view_new_with_pixbuf(pixbuf)
```

pixbuf :	a gtk.gdk.Pixbuf
Returns :	a new gtk.CellView

Note

This function is available in PyGTK 2.6 and above.

The `gtk.cell_view_new_with_pixbuf()` function creates a new gtk.CellView with a gtk.CellRendererPixbuf displaying the gtk.gdk.Pixbuf specified by *pixbuf*.

Prev Up Next
gtk.CellRendererToggle Home gtk.CheckButton

gtk.CheckButton

Prev **The gtk Class Reference** Next

gtk.CheckButton

gtk.CheckButton — a toggle button widget styled as a checkbox and label

Synopsis

```
class gtk.CheckButton(gtk.ToggleButton):
    gtk.CheckButton(label=None, use_underline=TRUE)
```

Ancestry

```
+-- gobject.GObject
   +-- gtk.Object
      +-- gtk.Widget
         +-- gtk.Container
            +-- gtk.Bin
               +-- gtk.Button
                  +-- gtk.ToggleButton
                     +-- gtk.CheckButton
```

Style Properties

"indicator–spacing"	Read/Write	The spacing around the check or radio indicator
"indicator–size"	Read/Write	The size of the check or radio indicator

Description

A `gtk.CheckButton` places a discrete `gtk.ToggleButton` next to a widget, (usually a `gtk.Label`). See the section on `gtk.ToggleButton` widgets for more information about toggle and check buttons. The signal ('toggled') is also inherited from `gtk.ToggleButton`.

Constructor

```
    gtk.CheckButton(label=None, use_underline=TRUE)
```

label :	a string to be used as the label text or None
use_underline :	if `TRUE`, an underscore in the label text indicates the next character should be underlined and used for the mnemonic accelerator key if it is the first character so marked. Available in PyGTK 2.4 and above.
Returns :	a new checkbutton object

Creates a new checkbutton with a text label specified by *`label`*. If *`label`* is `None` or not specified then no label is created. If *`label`* contains underscore characters then the character following the underscore will be underlined and the character following the first underscore will be used as the mnemonic keyboard accelerator.

In PyGTK 2.4 and above the *`use_underline`* parameter is available and defaults to `TRUE`. If *`use_underline`* is set to `FALSE` the label text will not be parsed for mnemonic characters.

gtk.CheckMenuItem

gtk.CheckMenuItem a menu item with a check box.

Synopsis

```
class gtk.CheckMenuItem(gtk.MenuItem):
    gtk.CheckMenuItem(label=None, use_underline=TRUE)
    def set_active(is_active)
    def get_active()
    def toggled()
    def set_inconsistent(setting)
    def get_inconsistent()
    def set_draw_as_radio(draw_as_radio)
    def get_draw_as_radio()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Item
            +-- gtk.MenuItem
              +-- gtk.CheckMenuItem
```

Properties

"active"	Read/Write	If `TRUE`, the menu item is checked.
"inconsistent"	Read/Write	If `TRUE`, display an "inconsistent" state.
"draw–as–radio"	Read/Write	If `TRUE`, display like a radio menu item. GTK+ 2.4 and above.

Style Properties

"indicator–size"	Read	The size of the check or radio indicator.

Attributes

"active"	Read	If `TRUE`, the menu item is checked.

Signal Prototypes

"toggled"	`def callback(checkmenuitem, user_param1, ...)`

Description

A gtk.CheckMenuItem is a menu item that maintains the state of a boolean value in addition to a gtk.MenuItem's usual role in activating application code. A check box indicating the state of the boolean value is displayed at the left side of the gtk.MenuItem. Activating the gtk.MenuItem toggles the value.

Constructor

```
    gtk.CheckMenuItem(label=None, use_underline=TRUE)
```

label :	a string to be used as the label text or None
use_underline :	if TRUE, an underscore in the label text indicates the next character should be underlined and used for the mnemonic accelerator key if it is the first character so marked. Available in PyGTK 2.4 and above.
Returns :	a new checkmenuitem

Creates a new checkmenuitem with a text label specified by *label*. If *label* is None or not specified then no label is created. If *label* contains underscore characters then the character following the underscore will be underlined and the character following the first underscore will be used as the mnemonic keyboard accelerator.

In PyGTK 2.4 and above the *use_underline* parameter is available and defaults to TRUE. If *use_underline* is set to FALSE the label text will not be parsed for mnemonic characters.

Methods

gtk.CheckMenuItem.set_active

```
    def set_active(is_active)
```

is_active :	If TRUE set the check box active

The set_active() method sets the active state of the menu item's check box according to the value of *is_active*.

gtk.CheckMenuItem.get_active

```
    def get_active()
```

Returns :	TRUE if the menu item is checked (check box is active).

The get_active() method returns whether the check menu item is active. See gtk.CheckMenuItem.set_active().

gtk.CheckMenuItem.toggled

```
    def toggled()
```

The toggle() method emits the "toggled" signal on the checkmenuitem

gtk.CheckMenuItem.set_inconsistent

```
    def set_inconsistent(setting)
```

`setting` :	If `TRUE` display an "inconsistent" third state check

The `set_inconsistent()` method sets the check box to display an "inconsistent" state if the value of *`setting`* is `TRUE`. The "inconsistent" state display is removed if *`setting`* is `FALSE`.

An application may want to display an "inconsistent" state if the user has selected a range of elements (such as some text or spreadsheet cells) that are affected by a boolean setting, and the current values for those elements cannot be represented by a single checkmenuitem state. The `set_inconsistent()` method only affects visual appearance, it doesn't affect the semantics of the widget.

gtk.CheckMenuItem.get_inconsistent

```
    def get_inconsistent()
```

Returns :	`TRUE` if the checkmenuitem displays the "inconsistent" state

The `get_inconsistent()` method retrieves the value set by the `set_inconsistent()` method.

gtk.CheckMenuItem.set_draw_as_radio

```
    def set_draw_as_radio(draw_as_radio)
```

`draw_as_radio` :	If `TRUE` display the check menu item like a radio menu item

Note

This method is available in PyGTK 2.4 and above.

The `set_draw_as_radio()` method displays the check menu item like a radio menu item if the value of *`draw_as_radio`* is `TRUE`. If *`draw_as_radio`* is `FALSE` the check menu item is displayed as normal.

gtk.CheckMenuItem.get_draw_as_radio

```
    def get_draw_as_radio()
```

Returns :	`TRUE` if the checkmenuitem should be displayed like a radio menu item.

Note

This method is available in PyGTK 2.4 and above.

The `get_draw_as_radio()` method retrieves the value set by the `set_draw_as_radio()` method.

Signals

The "toggled" gtk.CheckMenuItem Signal

```
    def callback(checkmenuitem, user_param1, ...)
```

`checkmenuitem` :	the checkmenuitem that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method

`...` :	additional user parameters (if any)

The "toggled" signal is emitted when the checkmenuitem changes state.

gtk.Clipboard

gtk.Clipboard — an object to store data to and retrieve data from (new in PyGTK 2.2)

Synopsis

```
class gtk.Clipboard(gobject.GObject):
    gtk.Clipboard(display=gtk.gdk.display_get_default(), selection="CLIPBOARD")
    def get_display()
    def set_with_data(targets, get_func, clear_func, user_data)
    def get_owner()
    def clear()
    def set_text(text, len=-1)
    def request_contents(target, callback, user_data=None)
    def request_text(callback, user_data=None)
    def request_targets(callback, user_data=None)
    def wait_for_contents(target)
    def wait_for_text()
    def wait_is_text_available()
    def wait_for_targets()
    def wait_is_target_available(target)
    def set_can_store(targets)
    def store()
```

```
Functions

    def gtk.clipboard_get(selection="CLIPBOARD")
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Clipboard
```

Description

Note

This object is available in PyGTK 2.2 and above.

The `gtk.Clipboard` object represents a clipboard of data shared between different processes or between different widgets in the same process. Each clipboard is identified by a name encoded as a `gtk.gdk.Atom`. The `gtk.Clipboard` is basically a higher−level interface to the lower−level `gtk.SelectionData` and selection interface. The built−in atoms are:

- "PRIMARY"
- "SECONDARY"

- "CLIPBOARD"
- "BITMAP"
- "COLORMAP"
- "DRAWABLE"
- "PIXMAP"
- "STRING"
- "WINDOW"

Creating a `gtk.gdk.Atom` from strings can be done with the gtk.gdk.atom_intern() constructor function though `PyGTK` will usually do the conversion under the covers as needed. The name of a `gtk.gdk.Atom` can be retrieved using the Python `str`() function:

```
  name = str(atom)
```

The default clipboard corresponds to the "CLIPBOARD" atom; another commonly used clipboard is the "PRIMARY" clipboard, which, in X, traditionally contains the currently selected text.

To simultaneously support different formats on the clipboard, the clipboard mechanism allows you to provide callbacks instead of the actual data. When you set the contents of the clipboard, you can either supply the data directly (via a method like `set_text()`), or you can supply a callback to be called when the data is needed (via the `set_with_data()` method.) Providing a callback also avoids making unnecessary copies of the data.

Along with the methods to get the clipboard contents as an arbitrary data chunk, there is a method to retrieve it as text, the `wait_for_text()` method. This method takes care of determining which formats are advertised by the clipboard provider, asking for the clipboard in the best available format and converting the results into the UTF−8 encoding. (The standard form for representing strings in `GTK+`.)

Constructor

```
    gtk.Clipboard(display=gtk.gdk.display_get_default(), selection="CLIPBOARD")
```

`display` :	the `gtk.gdk.Display` for which the clipboard is to be retrieved or created.
`selection` :	a string that identifies the clipboard to use.
Returns :	the appropriate clipboard object or if no clipboard already exists, a new one will be created. Once a clipboard object has been created, it is persistent for all time and cannot be freed.

Note

This constructor is available in PyGTK 2.2 and above.

Returns the clipboard object for the `gtk.gdk.Display` specified by *`display`* and the selection specified by the string in *`selection`*. Cut/copy/paste menu items and keyboard shortcuts should use the default clipboard, returned by passing `"CLIPBOARD"` for *`selection`*. The currently−selected object or text should be provided on the clipboard identified by `"PRIMARY"`. Cut/copy/paste menu items conceptually copy the contents of the `"PRIMARY"` clipboard to the default clipboard, i.e. they copy the selection to what the user sees as the clipboard.

See http://www.freedesktop.org/standards/clipboards−spec/clipboards.txt for a detailed discussion of the "CLIPBOARD" vs. "PRIMARY" selections under the X window system. On Win32 the `"PRIMARY"` clipboard is essentially ignored.

It's possible to have arbitrarily named clipboards. If you do invent new clipboards, you should prefix the selection name with an underscore (because the ICCCM requires that nonstandard atoms are underscore−prefixed), and namespace it as well. For example, if your application called "Foo" has a special–purpose clipboard, you might call it "_FOO_SPECIAL_CLIPBOARD".

In PyGTK 2.4 and above, the *display* argument is optional and defaults to the default display returned from the `gtk.gdk.display_get_default()` function.

In PyGTK 2.4 and above, the *selection* argument is optional and defaults to "CLIPBOARD".

Methods

gtk.Clipboard.get_display

```
def get_display()
```

Returns :	the `gtk.gdk.Display` associated with the clipboard

Note

This method is available in PyGTK 2.2 and above.

The `get_display()` method returns the `gtk.gdk.Display` associated with the clipboard.

gtk.Clipboard.set_with_data

```
def set_with_data(targets, get_func, clear_func, user_data)
```

`targets` :	a list of 3–tuples containing information about the available forms for the clipboard data
`get_func` :	a function to call to get the actual clipboard data
`clear_func` :	when the clipboard contents are set again, this function will be called, and *get_func* will not be subsequently called.
`user_data` :	the user data to pass to *get_func* and *clear_func*.
Returns :	`TRUE` if setting the clipboard data succeeded. If setting the clipboard data failed the provided callback functions will be ignored.

Note

This method is available in PyGTK 2.2 and above.

The `set_with_data()` method virtually sets the contents of the specified clipboard by providing a list of supported formats (specified by *targets*) for the clipboard data and a function (specified by *get_func*) to call to get the actual data when it is requested. *clear_func* is a function that is called when the contents of the clipboard are being changed to provide cleanup operations on *user_data*. *user_data* is passed to *get_func* and *clear_func* when they are invoked. The 3–tuples listed in *targets* contain the following items:

- a string representing a target supported by the clipboard
- a flags value used for drag and drop – a combination of: `gtk.TARGET_SAME_APP` and `gtk.TARGET_SAME_WIDGET`
- an application assigned integer that is passed as a signal parameter to help identify the target type

The signature of *get_func* is:

```
def get_func(clipboard, selectiondata, info, data):
```

where *clipboard* is the gtk.Clipboard, *selectiondata* is a gtk.SelectionData object to set with the data, *info* is the application assigned integer associated with a target, and *data* is the *user_data* argument.

The signature of *clear_func* is:

```
  def clear_func(clipboard, data):
```

where *clipboard* is the gtk.Clipboard and *data* is the *user_data* argument.

gtk.Clipboard.get_owner

```
    def get_owner()
```

Returns :	the owner of the clipboard, if any; otherwise `None`.

Note

This method is available in PyGTK 2.2 and above.

The `get_owner()` method returns the owner set by the `set_with_owner()` method if neither the `set_with_data()` method nor the `clear()` method have been subsequently called. This method returns `None` otherwise.

gtk.Clipboard.clear

```
    def clear()
```

Note

This method is available in PyGTK 2.2 and above.

The `clear()` method clears the contents of the clipboard. Generally this should only be called between the time you call the `set_with_data()`, and when the *clear_func* you supplied is called. Otherwise, the clipboard may be owned by someone else.

gtk.Clipboard.set_text

```
    def set_text(text, len=-1)
```

text :	a string.
len :	the length of *text*, in bytes, or –1, to calculate the length.

Note

This method is available in PyGTK 2.2 and above.

The `set_text()` method sets the contents of the clipboard to the string specified by *text*. If *len* is given it determines the length of *text* to be copied. If *len* is not specified it defaults to –1 and the method calculates the text length.

gtk.Clipboard.request_contents

```
def request_contents(target, callback, user_data=None)
```

target :	a gtk.gdk.Atom or string representing the form that the clipboard owner should convert the selection to.
callback :	a function to call when the results are received (or the retrieval fails).
user_data :	user data to pass to *callback*

Note

This method is available in PyGTK 2.4 and above.

The `request_contents()` method requests the contents of clipboard in the form specified by *target*. When the results of the request are later received the function specified by *callback* will be invoked and passed the data specified by *user_data*. The signature of *callback* is:

```
def callback(clipboard, selection_data, data):
```

where *clipboard* is the gtk.Clipboard that invoked callback and *selection_data* is the gtk.SelectionData containing the target data and *data* is *user_data*.

gtk.Clipboard.request_text

```
def request_text(callback, user_data=None)
```

callback :	a function to call when the text is received, or the retrieval fails. (It will always be called one way or the other.)
user_data :	user data to pass to *callback*.

Note

This method is available in PyGTK 2.4 and above.

The `request_text()` method requests the contents of the clipboard as text. When the text is later received, it will be converted to UTF−8 if necessary, and *callback* will be called with the data specified by *user_data*. The signature of *callback* is:

```
def callback(clipboard, text, data):
```

where *clipboard* is the gtk.Clipboard that *text* is retrieved from and *data* is *user_data*. *text* will contain the resulting text if the request succeeded, or the empty string if it failed. This could happen for various reasons, in particular if the clipboard was empty or if the contents of the clipboard could not be converted into text form.

gtk.Clipboard.request_targets

```
def request_targets(callback, user_data=None)
```

callback :	a function to call when the targets are received, or the retrieval fails. (It will always be called one way or the other.)
user_data :	user data to pass to *callback*.

Note

This method is available in PyGTK 2.4 and above.

The `request_targets()` method requests the contents of the clipboard as list of supported targets. When the list is later received, *`callback`* will be called with the data specified by *`user_data`*. The signature of *`callback`* is:

```
def callback(clipboard, targets, data):
```

where *`clipboard`* is the `gtk.Clipboard` that *`targets`* is retrieved from. *`targets`* is a tuple containing the `gtk.gdk.Atom` objects corresponding to the targets of clipboard. *`targets`* will contain the resulting targets if the request succeeded, or an empty tuple if it failed.

gtk.Clipboard.wait_for_contents

```
def wait_for_contents(target)
```

`target` :	an atom or string representing the form into which the clipboard owner should convert the selection.
Returns :	a newly−allocated `gtk.SelectionData` object or `None` if retrieving the given target failed.

Note

This method is available in PyGTK 2.2 and above.

The `wait_for_contents()` method requests the contents of the clipboard using the target specified by *`target`*. This method waits for the data to be received using the main loop, so events, timeouts, etc, may be dispatched during the wait.

gtk.Clipboard.wait_for_text

```
def wait_for_text()
```

Returns :	a string, or `None` if retrieving the selection data failed. (This could happen for various reasons, in particular if the clipboard was empty or if the contents of the clipboard could not be converted into text form.)

Note

This method is available in PyGTK 2.2 and above.

The `wait_for_text()` method requests the contents of the clipboard as text and converts the result to UTF−8 if necessary. This method waits for the data to be received using the main loop, so events, timeouts, etc, may be dispatched during the wait.

gtk.Clipboard.wait_is_text_available

```
def wait_is_text_available()
```

Returns :	`TRUE` is there is text available.

Note

This method is available in PyGTK 2.2 and above.

The `wait_is_text_available()` method tests to see if there is text available to be copied from the clipboard. This is done by requesting the "TARGETS" atom and checking if it contains any of the names: "STRING", "TEXT", "COMPOUND_TEXT", "UTF8_STRING". This method waits for the data to be received using the main loop, so events, timeouts, etc, may be dispatched during the wait.

This method is a little faster than calling the `wait_for_text()` since it doesn't need to retrieve the actual text.

gtk.Clipboard.wait_for_targets

```
def wait_for_targets()
```

Returns :	returns a tuple containing any targets that are present on the clipboard or `None`.

Note

This method is available in PyGTK 2.4 and above.

The `wait_for_targets`() method returns a tuple containing the targets (as `gtk.gdk.Atom` objects) that are present on the clipboard, or `None` if there aren't any targets available. This function waits for the data to be received using the main loop, so events, timeouts, etc, may be dispatched during the wait.

gtk.Clipboard.wait_is_target_available

```
def wait_is_target_available(target)
```

`target` :	an atom or string representing the target of interest.
Returns :	`TRUE` if the target is available.

Note

This method is available in PyGTK 2.6 and above.

The `wait_is_target_available`() method tests to see if the target specified by *`target`* is available to be copied from the clipboard. This method can be used to determine if a Paste menu item should be insensitive or not.

If you want to see if there's text available on the clipboard, use the `wait_is_text_available()` method instead.

gtk.Clipboard.set_can_store

```
def set_can_store()
```

`targets` :	a list of 3–tuples containing information about the available forms that should be stored or `None` to indicate that all forms should be stored.

Note

This method is available in PyGTK 2.6 and above.

The `set_can_store`() method sets a hint that the `gtk.Clipboard` can store the list of targets specified by *`targets`* can be stored somewhere when the application exits or when the `store()` method is called. This value is reset when the clipboard owner changes. Where the clipboard data is stored is platform dependent, see the `gtk.gdk.Display.store_clipboard()` method for more information. If *`targets`* is `None` all target forms currently available on the clipboard should be stored.

The 3–tuples listed in *`targets`* contain the following items:

- a string representing a target supported by the clipboard

- a flags value used for drag and drop – a combination of: `gtk.TARGET_SAME_APP` and `gtk.TARGET_SAME_WIDGET`
- an application assigned integer that is passed as a signal parameter to help identify the target type

gtk.Clipboard.store

```
  def store()
```

Note

This method is available in PyGTK 2.6 and above.

The `store()` method stores the current clipboard data (as specified by the `set_can_store()` method) somewhere so that it will stay around after the application has quit.

Functions

gtk.clipboard_get

```
  def gtk.clipboard_get(selection="CLIPBOARD")
```

`selection` :	a string specifying a `gtk.Clipboard`. If not specified it defaults to "CLIPBOARD".
Returns :	the appropriate clipboard object or if no clipboard already exists, a new one will be created. Once a clipboard object has been created, it is persistent for all time and cannot be freed.

Note

This function is available in PyGTK 2.4 and above.

The `gtk.clipboard_get()` function returns the `gtk.Clipboard` specified by selection for the default `gtk.gdk.Display`. See the gtk.Clipboard constructor for more information.

Prev Up Next
gtk.CheckMenuItem Home gtk.ColorButton

gtk.ColorButton
Prev The gtk Class Reference Next

gtk.ColorButton

gtk.ColorButton — a button to launch a color selection dialog (new in PyGTK 2.4)

Synopsis

```
class gtk.ColorButton(gtk.Button):
    gtk.ColorButton(color)
    def set_color(color)
    def get_color()
    def set_alpha(alpha)
    def get_alpha()
    def set_use_alpha(use_alpha)
    def get_use_alpha()
    def set_title(title)
    def get_title()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Button
            +-- gtk.ColorButton
```

Properties

"alpha"	Read–Write	The selected opacity value (0 fully transparent, 65535 fully opaque). Available in GTK+ 2.4 and above.
"color"	Read–Write	The selected color. Available in GTK+ 2.4 and above.
"title"	Read–Write	The title of the color selection dialog. Available in GTK+ 2.4 and above.
"use–alpha"	Read–Write	If `TRUE`, the color swatch on the button is rendered against a checkerboard background to show its opacity and the opacity slider is displayed in the color selection dialog. Available in GTK+ 2.4 and above.

Signal Prototypes

"color–set"	`def callback(`*`colorbutton`*`, `*`user_param1`*`, ...)`

Description

The `gtk.ColorButton` is a button which displays the currently selected color and, when clicked, opens a `gtk.ColorSelectionDialog` to change the color. It's a suitable widget for selecting a color in a preference dialog. The `gtk.ColorButton` is available in PyGTK 2.4 and above.

Constructor

```
    gtk.ColorButton(color=gtk.gdk.Color(0,0,0))
```

`color` :	an optional `gtk.gdk.Color` to set the current color with
Returns :	a new color button.

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new color button with the current color set to the color specified by the optional `gtk.gdk.Color` *`color`*. A color button is a small button containing a swatch representing the current selected color. When the button is clicked, a `gtk.ColorSelectionDialog` will open, allowing the user to select a color. The swatch will be updated to reflect the new color the user selects.

Methods

gtk.ColorButton.set_color

```
    def set_color(color)
```

color :	A gtk.gdk.Color to set the current color with.

Note

This method is available in PyGTK 2.4 and above.

The `set_color`() method sets the current color (and the "color" property) to the color specified by the `gtk.gdk.Color` *`color`*.

gtk.ColorButton.get_color

```
    def get_color()
```

Returns :	a gtk.gdk.Color specifying the current color.

Note

This method is available in PyGTK 2.4 and above.

The `get_color`() method returns the value of the "color" property which is a `gtk.gdk.Color` specifying the current color in the `gtk.ColorButton` widget.

gtk.ColorButton.set_alpha

```
    def set_alpha(alpha)
```

alpha :	The opacity in the range 0 to 65535.

Note

This method is available in PyGTK 2.4 and above.

The `set_alpha`() method sets the current opacity (and the "alpha" property) to the value specified by *`alpha`*.

gtk.ColorButton.get_alpha

```
    def get_alpha()
```

Returns :	the opacity in the range 0 to 65535.

Note

This method is available in PyGTK 2.4 and above.

The `get_alpha`() method returns the value of the "alpha" property that contains the opacity setting.

gtk.ColorButton.set_use_alpha

```
    def set_use_alpha(use_alpha)
```

use_alpha :	if TRUE, the color button should use the alpha channel.

Note

This method is available in PyGTK 2.4 and above.

The set_use_alpha() method sets the "use−alpha" property to the value of *use_alpha*. If *use_alpha* is TRUE, the color swatch on the button is rendered against a checkerboard background to show its opacity and the opacity slider is displayed in the color selection dialog.

gtk.ColorButton.get_use_alpha

```
    def get_use_alpha()
```

Returns :	TRUE if the color sample should use the alpha channel

Note

This method is available in PyGTK 2.4 and above.

The get_use_alpha() method returns the value of the "use−alpha" property. If TRUE the color selection dialog should use the alpha channel.

gtk.ColorButton.set_title

```
    def set_title(title)
```

title :	a string containing the new gtk.ColorSelectionDialog title.

Note

This method is available in PyGTK 2.4 and above.

The set_title() method sets the title for the color selection dialog to the string contained in *title*. The "title" property is also set.

gtk.ColorButton.get_title

```
    def get_title()
```

Returns :	the title of the gtk.ColorSelectionDialog

Note

This method is available in PyGTK 2.4 and above.

The get_title() method returns the value of the "title" property that contains the title of the color selection dialog.

Signals

The "color−set" gtk.ColorButton Signal

```
    def callback(colorbutton, user_param1, ...)
```

colorbutton :	the colorbutton that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

Note

This signal is available in PyGTK 2.4 and above.

The "color−set" signal is emitted when the user selects a color. When handling this signal, use the get_color() and the get_alpha() methods to find out what color was just selected.

gtk.ColorSelection

gtk.ColorSelection — a widget used to select a color.

Synopsis

```
class gtk.ColorSelection(gtk.VBox):
    gtk.ColorSelection()
    def get_has_opacity_control()
    def set_has_opacity_control(has_opacity)
    def get_has_palette()
    def set_has_palette(has_palette)
    def set_current_color(color)
    def set_current_alpha(alpha)
    def get_current_color()
    def get_current_alpha()
    def set_previous_color(color)
    def set_previous_alpha(alpha)
    def get_previous_color()
    def get_previous_alpha()
    def is_adjusting()
```

```
Functions

    def gtk.color_selection_palette_from_string(str)
    def gtk.color_selection_palette_to_string(colors)
```

Ancestry

```
+-- gobject.GObject
  +--gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
```

```
        +-- gtk.Box
          +-- gtk.VBox
            +-- gtk.ColorSelection
```

Properties

"has–palette"	Read/Write	If `TRUE`, a palette should be used
"has–opacity–control"	Read/Write	If `TRUE`, the color selector should allow setting opacity
"current–color"	Read/Write	The current color as a `gtk.gdk.Color`
"current–alpha"	Read/Write	The current opacity value (0 fully transparent, 65535 fully opaque)

Signal Prototypes

"color–changed"	`def callback(`*`colorselection, user_param1, ...`*`)`

Description

The `gtk.ColorSelection` is a widget that is used to select a color. It consists of a color wheel and entry boxes for color parameters such as hue, saturation, value, red, green, blue, and color name and optionally an opacity control and/or a color palette. It is found on the standard color selection dialog box `gtk.ColorSelectionDialog`. The color wheel can be manipulated to set a new color or new entries can be made in the entry boxes. The new color is displayed next to the previous color. An eyedropper button is available to allow the selection of a color from a pixel location on the display.

Constructor

```
    gtk.ColorSelection()
```

Returns :	a new `gtk.ColorSelection`

Creates a new `gtk.ColorSelection` widget.

Methods

gtk.ColorSelection.get_has_opacity_control

```
    def get_has_opacity_control()
```

Returns :	`TRUE` if the *`colorsel`* has an opacity control; `FALSE` if it doesn't.

The `get_opacity`() method determines whether the colorselection is displaying an opacity control.

gtk.ColorSelection.set_has_opacity_control

```
    def set_has_opacity_control(has_opacity)
```

`has_opacity` :	If `TRUE` the colorselection will display the opacity control.

The `set_has_opacity_control`() method sets the "has–opacity–control" property to the value of *`has_opacity`*. If *`has_opacity`* is `TRUE` the colorselection will display the opacity control slider and entry box; otherwise the opacity control is not displayed.

gtk.ColorSelection.get_has_palette

```
def get_has_palette()
```

Returns :	`TRUE` if the selector has a palette; `FALSE` if it hasn't.

The `get_has_palette()` method returns the value of the "has−palette" property that determines whether the color selector displays a color palette.

gtk.ColorSelection.set_has_palette

```
def set_has_palette(has_palette)
```

`has_palette` :	If `TRUE` the color palette is displayed.

The `set_has_palette()` method sets the "has−palette" property to the value of *`has_palette`*. If *`has_palette`* is TRUE the palette will be displayed; otherwise the palette will be hidden.

gtk.ColorSelection.set_current_color

```
def set_current_color(color)
```

`color` :	A `gtk.gdk.Color` to set the current color with.

The `set_current_color()` method sets the current color to the value of *`color`*. The first time this is called, it will also set the previous color to the value of *`color`* too.

gtk.ColorSelection.set_current_alpha

```
def set_current_alpha(alpha)
```

`alpha` :	an integer between 0 and 65535.

The `set_current_alpha()` method sets the "current−alpha" property (the opacity) to the value of alpha. The alpha (opacity) is displayed in the range of 0 to 255 in the colorselection. The first time this is called, it will also set the previous opacity to the value of *`alpha`* too.

gtk.ColorSelection.get_current_color

```
def get_current_color()
```

Returns :	a `gtk.gdk.Color` representing the current color.

The `get_current_color()` method retrieves the current color in the colorselection.

gtk.ColorSelection.get_current_alpha

```
def get_current_alpha()
```

Returns :	the current alpha value in the range 0 to 65535.

The `get_current_alpha()` method returns the value of the "current_alpha" property that controls the opacity value.

gtk.ColorSelection.set_previous_color

```
def set_previous_color(color)
```

`color` :	a `gtk.gdk.Color` to set the previous color with.

The `set_previous_color()` method sets the 'previous' color to the value of *`color`*. Applications usually do not call this method. The first time `set_current_color()` the 'previous' color will be set.

gtk.ColorSelection.set_previous_alpha

```
    def set_previous_alpha(alpha)
```

`alpha` :	an integer between 0 and 65535.

The set_previous_alpha() method sets the 'previous' alpha to the value of *`alpha`*. Applications usually do not call this method. The first time `set_current_alpha()` the 'previous' alpha will be set.

gtk.ColorSelection.get_previous_color

```
    def get_previous_color()
```

Returns :	a `gtk.gdk.Color` with the previous color value.

The `get_previous_color()` method retrieves the previous color value.

gtk.ColorSelection.get_previous_alpha

```
    def get_previous_alpha()
```

Returns :	an integer between 0 and 65535.

The get_previous_alpha() method returns the previous alpha value.

gtk.ColorSelection.is_adjusting

```
    def is_adjusting()
```

Returns :	`TRUE` if the user is currently dragging a color around, and `FALSE` if the selection has stopped.

The `is_adjusting()` method retrieves the current state of the colorselection. If TRUE the user is in the process of changing the current color.

Functions

gtk.color_selection_palette_from_string

```
    def gtk.color_selection_palette_from_string(str)
```

`str` :	the string containing the list of colors
Returns :	a list of `gtk.gdk.Color` objects or `None` if the conversion fails

The `gtk.color_selection_palette_from_string()` function returns a list of `gtk.gdk.Color` objects corresponding to the color specifications in the string specified by *`str`*. str is a colon−separated list of color names readable by `gtk.gtk.color_parse()`. If *`str`* cannot be converted to a list of color this function returns `None`.

gtk.color_selection_palette_to_string

```
    def gtk.color_selection_palette_to_string(colors)
```

colors :	a list or tuple of gtk.gdk.Color objects
Returns :	a string containing a colon−separated list of *colors*

The gtk.color_selection_palette_to_string() function returns a string containing a colon−separated list of the representation of the gtk.gdk.Color objects in *colors*.

This function is useful to save a special palette of colors for a gtk.ColorSelection as a string that can later be used by calling the gobject.set_property() method to set the "gtk−color−palette" property on the default gtk.Settings returned from the gtk.settings_get_default()) function.

Signals

The "color−changed" gtk.ColorSelection Signal

```
    def callback(colorselection, user_param1, ...)
```

colorselection :	the colorselection that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "color−changed" signal is emitted when the current color in the colorselection changes.

Prev	Up	Next
gtk.ColorButton	Home	gtk.ColorSelectionDialog

	gtk.ColorSelectionDialog	
Prev	**The gtk Class Reference**	Next

gtk.ColorSelectionDialog

gtk.ColorSelectionDialog — a standard dialog for selecting a color.

Synopsis

```
class gtk.ColorSelectionDialog(gtk.Dialog):
    gtk.ColorSelectionDialog(title)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Window
            +-- gtk.Dialog
              +-- gtk.ColorSelectionDialog
```

Attributes

"colorsel"	Read	The gtk.ColorSelection widget contained in the dialog.
"ok_button"	Read	The "OK" button contained in the dialog.
"cancel_button"	Read	The "Cancel" button contained in the dialog.
"help_button"	Read	The "Help" button contained in the dialog.

Description

The gtk.ColorSelectionDialog provides a standard dialog that allows a user to select a color. The gtk.ColorSelectionDialog uses an embedded gtk.ColorSelection to provide color selection capability.

Use the attributes (ok_button, cancel_button and help_button) of the colorselectiondialog to connect handlers to the "OK", "Cancel" and "Help" button "clicked" signals. The colorsel attribute provides access to the colorselection widget. Connect a handler to its "color–changed" signal to be notified when the color is changed. The current color can be retrieved using the gtk.ColorSelection.get_current_color() method.

Constructor

```
    gtk.ColorSelectionDialog(title)
```

title :	a string to be used as the dialog title.
Returns :	a new colorselectiondialog

Creates a new `gtk.ColorSelectionDialog` using the string contained in *`title`* as the text for the dialog title.

Prev	Up	Next
gtk.ColorSelection	Home	gtk.Combo

gtk.Combo

Prev **The gtk Class Reference** Next

gtk.Combo

gtk.Combo a text entry field with a dropdown list.

Synopsis

```
class gtk.Combo(gtk.HBox):
    gtk.Combo()
    def set_value_in_list(val, ok_if_empty)
    def set_use_arrows(val)
    def set_use_arrows_always(val)
    def set_case_sensitive(val)
    def set_item_string(item, item_value)
    def set_popdown_strings(strings)
    def disable_activate()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Box
          +-- gtk.HBox
            +-- gtk.Combo
```

Properties

"enable–arrow–keys"	Read/Write	If `TRUE`, the arrow keys move through the list of items
"enable–arrows–always"	Read/Write	If `TRUE`, the arrow keys work, even if the entry contents are not in the list
"case–sensitive"	Read/Write	If `TRUE`, list item matching is case sensitive
"allow–empty"	Read/Write	If `TRUE`, an empty value may be entered in this field
"value–in–list"	Read/Write	If `TRUE`, entered values must already be present in the list

Attributes

"entry"	Read	the text entry widget
"list"	Read	the list shown in the drop–down window

Description

Note

The `gtk.Combo` is deprecated in GTK+ 2.4 and PyGTK 2.4 in favor of the `gtk.ComboBox`.

The `gtk.Combo` widget consists of a single–line text entry field and a drop–down list. The drop–down list is displayed when the user clicks on a small arrow button to the right of the entry field. The drop–down list is a `gtk.List` widget and can be accessed using the list attribute of the `gtk.Combo`. List elements can contain arbitrary widgets, but if an element is not a plain label, then you must use the `gtk.List.set_item_string`() method. This sets the string which will be placed in the text entry field when the item is selected.

By default, the user can step through the items in the list using the arrow (cursor) keys, though this behavior can be turned off with the `set_use_arrows()` method. Normally the arrow keys are only active when the contents of the text entry field matches one of the items in the list. If the contents of the entry field do not match any of the list items, then pressing the arrow keys does nothing. However, by calling `set_use_arrows_always()` you can specify that the arrow keys are always active. If the contents of the entry field does not match any of the items in the list, then pressing the up or down arrow key will set the entry field to the last or first item in the list, respectively.

Note the list attribute references a gtk.List which is a deprecated widget so the combo widget may be changed or deprecated in the future.

Constructor

```
gtk.Combo()
```

Returns :	a combo object*

Creates an new `gtk.Combo` object.

Methods

gtk.Combo.set_value_in_list

```
def set_value_in_list(val, ok_if_empty)
```

`val` :	If TRUE the value entered must match one of the values in the list.
`ok_if_empty` :	If TRUE an empty value is considered valid.

The `set_value_in_list()` method specifies whether the value entered in the text entry field must match one of the values in the list. This method sets the "value–in–list" property to the value of *`val`* and the "allow–empty" property to the value of *`ok_if_empty`*.

If *`val`* is TRUE the user will not be able to perform any other action (the widget grabs the focus) until a valid value has been entered. If *`ok_if_empty`* is TRUE an empty field is considered an acceptable value.

gtk.Combo.set_use_arrows

```
def set_use_arrows(val)
```

`val` :	If TRUE can be used to navigate through the list items

The `set_use_arrows()` method sets the "enable–use–arrows" property to the value of *`val`*. If *`val`* is TRUE the arrows keys can be used to navigate through the list items.

gtk.Combo.set_use_arrows_always

```
def set_use_arrows_always(val)
```

`val` :	If TRUE the arrow keys will still work even if the text entry field does not match any of the list items.

The `set_use_arrows_always()` method sets the "enable–arrows–always" property to the values of *`val`*. If *`val`* is TRUE the arrow keys will work even if the text entry field does not match any of the list items.

gtk.Combo.set_case_sensitive

```
def set_case_sensitive(val)
```

`val` :	If TRUE the text in the list items is case sensitive.

The `set_case_sensitive()` method sets the "case–sensitive" property to the value of *`val`*. If *`val`* is TRUE the text in the combo list items and the text entry field are case sensitive. The default value of "set–case–sensitive" is FALSE.

gtk.Combo.set_item_string

```
    def set_item_string(item, item_value)
```

item :	a list item
item_value :	a string to place in the entry field when *item* is selected

The `set_item_string()` method sets the string (from *item_value*) to place in the combo text entry field when the *item* is selected. This method is only needed if the list item is other than a simple label (e.g. a pixmap).

gtk.Combo.set_popdown_strings

```
    def set_popdown_strings(strings)
```

strings :	a list of strings to populate the list

The `set_popdown_strings()` method is a convenience method that sets the strings used in the popdown list from. *strings* (a Python list or tuple object).

gtk.Combo.disable_activate

```
    def disable_activate()
```

The `disable_activate()` method prevents the combo from showing the popup list when the entry emits the "activate" signal, i.e. when the **Return** key is pressed. This may be useful if, for example, you want the **Return** key to close a dialog instead.

Prev	Up	Next
gtk.ColorSelectionDialog	Home	gtk.ComboBox

	gtk.ComboBox	
Prev	**The gtk Class Reference**	Next

gtk.ComboBox

gtk.ComboBox — a widget used to choose from a list of items (new in PyGTK 2.4)

Synopsis

```
class gtk.ComboBox(gtk.Bin, gtk.CellLayout):
    gtk.ComboBox(model=None)
    def get_wrap_width()
    def set_wrap_width(width)
    def get_row_span_column()
    def set_row_span_column(row_span)
    def get_column_span_column()
    def set_column_span_column(column_span)
    def get_active()
    def set_active(index)
    def get_active_iter()
    def set_active_iter(iter)
    def set_model(model=None)
    def get_model()
    def append_text(text)
    def insert_text(position, text)
    def prepend_text(text)
    def remove_text(position)
    def get_active_text()
    def popup()
    def popdown()
    def get_popup_accessible()
    def set_row_separator_func(func=None, data=None)
    def get_add_tearoffs()
    def set_add_tearoffs(add_tearoffs)
    def get_focus_on_click()
    def set_focus_on_click(focus_on_click)
Functions

    def gtk.combo_box_new_text()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.ComboBox (implements gtk.CellLayout)
```

Properties

"active"	Read–Write	The index of the item that is currently active.
"add–tearoffs"	Read–Write	If `TRUE` generated menus have tearoff menu items. Note that this only affects menu style combo boxes. Default value: `FALSE`. Available in GTK+ 2.6 and above.
"column–span–column"	Read–Write	The TreeModel column containing the column span values.
"has–frame"	Read–Write	If `TRUE` the combo box grabs focus when it is clicked with the mouse. Default value: `TRUE`. Available in GTK+ 2.6 and above.
"model"	Read–Write	The TreeModel for the combo box.
"row–span–column"	Read–Write	The TreeModel column containing the row span values.
"wrap–width"	Read–Write	The number of columns to use to lay out the popup items.

Style Properties

"appears–as–list"	Read–Write	If `TRUE`, the combo box dropdowns should look like lists rather than menus.

Signal Prototypes

"changed"	`def callback(`*`combobox, user_param1, ...`*`)`

Description

Note

This widget is available in GTK+ 2.4 and PyGTK 2.4 and above.

The `gtk.ComboBox` is a replacement for the `gtk.OptionMenu`. The `gtk.ComboBox` implements the `gtk.CellLayout` interface that provides a number of useful methods for managing the contents. A `gtk.ComboBox` is created with the gtk.ComboBox() constructor that is associated with the optional `gtk.TreeModel`. If no `gtk.TreeModel` is specified it can be added later with the `set_model()` method.

Alternatively, the `gtk.combo_box_new_text()` function creates a simple `gtk.ComboBox` and associated `gtk.ListStore` model. A `gtk.CellRendererText` is also created and packed in the new combo box. In this simple combo box each list item is a text string that can be selected. The convenience methods `append_text()`, `prepend_text()`, `insert_text()` and `remove_text()` can be used to manage the contents of the `gtk.ComboBox`. Using the `gtk.combo_box_new_text()` function is equivalent to:

```
liststore = gtk.ListStore(gobject.TYPE_STRING)
combobox = gtk.ComboBox(liststore)
cell = gtk.CellRendererText()
combobox.pack_start(cell, gtk.TRUE)
combobox.add_attribute(cell, 'text', 0)
```

Constructor

```
gtk.ComboBox(model=None)
```

`model` :	A valid `gtk.TreeModel`.
Returns :	A new `gtk.ComboBox`.

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new `gtk.ComboBox` associated with the optional `gtk.TreeModel` specified by *`model`*. If *`model`* is not specified the combo box will not have an associated tree model.

Methods

gtk.ComboBox.get_wrap_width

```
def get_wrap_width()
```

Returns :	The wrap width.

Note

This method is available in PyGTK 2.6 and above.

The `get_wrap_width`() method returns the value of the "wrap–width" property of the combo box as set by the `set_wrap_width()` method. The wrap width is basically the preferred number of columns to use to lay out the popup i.e. lays out the popup items in a table with *`width`* columns.

gtk.ComboBox.set_wrap_width

```
def set_wrap_width(width)
```

`width` :	The preferred number of columns of width.

Note

This method is available in PyGTK 2.4 and above.

The `set_wrap_width`() method sets the wrap width (and the "wrap−width" property) of the combo box to the value specified by *`width`*. The wrap width is basically the preferred number of columns to use to lay out the popup i.e. lays out the popup items in a table with *`width`* columns.

gtk.ComboBox.get_row_span_column

```
    def get_row_span_column()
```

Returns :	The row span column.

Note

This method is available in PyGTK 2.6 and above.

The `get_row_span_column`() method returns the value of the "row−span−column" property. The "row−span−column" property indicates the column in the associated `gtk.TreeModel` row that contains an integer that indicates how many rows the item should span.

gtk.ComboBox.set_row_span_column

```
    def set_row_span_column(row_span)
```

`row_span` :	A column in the model passed during construction.

Note

This method is available in PyGTK 2.4 and above.

The `set_row_span_column`() method sets the "row−span−column" property to the value specified by *`row_span`*. The "row−span−column" property indicates the column in the associated `gtk.TreeModel` row that contains an integer that indicates how many rows the item should span.

gtk.ComboBox.get_column_span_column

```
    def get_column_span_column()
```

Returns :	The column span column.

Note

This method is available in PyGTK 2.6 and above.

The `get_column_span_column`() method returns the value of the "column−span−column" property. The "column−span−column" property indicates the column in the associated `gtk.TreeModel` row that contains an integer that indicates how many columns the item should span.

gtk.ComboBox.set_column_span_column

```
    def set_column_span_column(column_span)
```

`column_span` :	A column in the model passed during construction.

Note

This method is available in PyGTK 2.4 and above.

The `set_column_span_column()` method sets the "column−span−column" property to the value specified by *`column_span`*. The "column−span−column" property indicates the column in the associated `gtk.TreeModel` row that contains an integer that indicates how many columns the item should span.

gtk.ComboBox.get_active

```
def get_active()
```

Returns :	An integer which is the model index of the currently active item, or −1 if there's no active item.

Note

This method is available in PyGTK 2.4 and above.

The `get_active()` method returns the value of the "active" property which is the index in the model of the currently active item, or −1 if there's no active item.

gtk.ComboBox.set_active

```
def set_active(index)
```

`index` :	An index in the model passed during construction, or −1 to have no active item.

Note

This method is available in PyGTK 2.4 and above.

The `set_active()` method sets the active item of the combo_box to the item with the model index specified by *`index`*. If *`index`* is −1 the combo box will have no active item. The "active" property is also set to the value of *`index`*.

gtk.ComboBox.get_active_iter

```
def get_active_iter()
```

Returns :	A `gtk.TreeIter` that points at the active item or `None` if there is no active item.

Note

This method is available in PyGTK 2.4 and above.

The `get_active_iter()` method returns a `gtk.TreeIter` that points to the current active item or `None` if there is no active item.

gtk.ComboBox.set_active_iter

```
def set_active_iter(iter)
```

`iter` :	A valid `gtk.TreeIter` pointing at an item in the associated `gtk.TreeModel`.

Note

This method is available in PyGTK 2.4 and above.

The `set_active_iter()` method sets the current active item to be the one referenced by *`iter`* in the associated `gtk.TreeModel`. *`iter`* must correspond to a path of depth one. The "active" property is also set by this method.

gtk.ComboBox.set_model

```
    def set_model(model=None)
```

`model` :	A `gtk.TreeModel`.

Note

This method is available in PyGTK 2.4 and above.

The `set_model()` method sets the model used by the combo box to the value specified by *`model`*. The "model" property will also be set to the value of *`model`*. A previously set model will be unset. If *`model`* is `None` or not specified, the old model will be unset.

Note

In PyGTK 2.4.0 the model could not be `None` and did not default to `None`.

gtk.ComboBox.get_model

```
    def get_model()
```

Returns :	A `gtk.TreeModel` or `None`.

Note

This method is available in PyGTK 2.4 and above.

The `get_model()` method returns the value of the "model" property which contains the `gtk.TreeModel` that is acting as data source for the combo_box or `None` if no `gtk.TreeModel` is associated with the combo box.

gtk.ComboBox.append_text

```
    def append_text(text)
```

`text` :	A string.

Note

This method is available in PyGTK 2.4 and above.

The `append_text()` method appends the string specified by *`text`* to the list of strings stored in the combo box `gtk.ListStore`. Note that you can only use this method with combo boxes constructed with the `gtk.combo_box_new_text()` function.

gtk.ComboBox.insert_text

```
    def insert_text(position, text)
```

`position` :	A model index where the *`text`* should be inserted.
`text` :	A string.

Note

This method is available in PyGTK 2.4 and above.

The `insert_text`() method inserts the string specified by *`text`* in the combo box `gtk.ListStore` at the index specified by *`position`*. Note that you can only use this method with combo boxes constructed with the `gtk.combo_box_new_text()` function.

gtk.ComboBox.prepend_text

```
    def prepend_text(text)
```

`text` :	A string.

Note

This method is available in PyGTK 2.4 and above.

The `prepend_text`() method prepends the string specified by *`text`* to the list of strings stored in the `gtk.ListStore` associated with the combo_box. Note that you can only use this method with combo boxes constructed with the `gtk.combo_box_new_text()` function.

gtk.ComboBox.remove_text

```
    def remove_text(position)
```

`position` :	Index of the item to remove.

Note

This method is available in PyGTK 2.4 and above.

The `remove_text`() method removes the string at the index specified by *`position`* in the associated `gtk.ListStore`. Note that you can only use this function with combo boxes constructed with the `gtk.combo_box_new_text()` function.

gtk.ComboBox.get_active_text

```
    def get_active_text()
```

Returns :	The currently active text.

Note

This method is available in PyGTK 2.6 and above.

The `get_active_text`() method returns the currently active string or `None` if no entry is selected. Note that you can only use this function with combo boxes constructed with the `gtk.combo_box_new_text()` function.

gtk.ComboBox.popup

```
def popup()
```

Note

This method is available in PyGTK 2.4 and above.

The `popup`() method pops up the menu or dropdown list of the combo box. This method is mostly intended for use by accessibility technologies; applications should have little use for it.

gtk.ComboBox.popdown

```
def popdown()
```

Note

This method is available in PyGTK 2.4 and above.

The `popdown`() method hides the menu or dropdown list of the combo box. This method is mostly intended for use by accessibility technologies; applications should have little use for it.

gtk.ComboBox.get_popup_accessible

```
def get_popup_accessible()
```

Returns :	the accessible object corresponding to the popup.

Note

This method is available in PyGTK 2.6 and above.

The `get_popup_accessible`() method gets the accessible object corresponding to the popup. This method is mostly intended for use by accessibility technologies; applications should have little use for it.

gtk.ComboBox.set_row_separator_func

```
def set_row_separator_func(func=None, data=None)
```

`func` :	a function or `None`
`data` :	user data to pass to *`func`*

Note

This method is available in PyGTK 2.6 and above.

The `set_row_separator_func`() method sets the row separator function to *`func`*, which is used to determine if a row should be drawn as a separator. If func is `None`, no separators are drawn. This is the default value.

The signature of *`func`* is:

```
def func(model, iter, user_data):
```

where *`model`* is the `gtk.TreeModel` used by the combo box, *`iter`* is a `gtk.TreeIter` pointing at a row in *`model`* and *`user_data`* is *`data`*. *`func`* returns `TRUE` if the row is a separator. A common way to implement *func* is to have a boolean column in *model*, that indicates if the row is a separator.

gtk.ComboBox.get_add_tearoffs

```
def get_add_tearoffs()
```

Returns :	`TRUE` if menus should have a tearoff menuitem.

Note

This method is available in PyGTK 2.6 and above.

The `get_add_tearoffs()` method returns the value of the "add−tearoffs" property.

gtk.ComboBox.set_add_tearoffs

```
def set_add_tearoffs(add_tearoffs)
```

`add_tearoffs` :	if `TRUE` add tearoff menu items

Note

This method is available in PyGTK 2.6 and above.

The `set_add_tearoffs()` method sets the "add−tearoffs" property to the value of *add_tearoffs*. If *add_tearoffs* is `TRUE`, the popup menu should have a tearoff menu item.

gtk.ComboBox.get_focus_on_click

```
def get_focus_on_click()
```

Returns :	`TRUE` if the combo box grabs focus when it is clicked with the mouse.

Note

This method is available in PyGTK 2.6 and above.

The `get_focus_on_click()` method returns the value of the "focus−on−click" property.

gtk.ComboBox.set_focus_on_click

```
def set_focus_on_click(focus_on_click)
```

`focus_on_click` :	if `TRUE` the combo box grabs focus when clicked with the mouse.

Note

This method is available in PyGTK 2.6 and above.

The `set_focus_on_click()` method sets the value of the "focus−on−click" property to the value of *focus_on_click*. If *focus_on_click* is `TRUE` the combo box grabs focus when clicked with the mouse.

Functions

gtk.combo_box_new_text

```
    def gtk.combo_box_new_text()
```

Returns :	A new `gtk.ComboBox` for text items.

Note

This function is available in PyGTK 2.4 and above.

The `gtk.combo_box_new_text()` function is a convenience function that constructs a new text combo box, which is a `gtk.ComboBox` just displaying strings. If you use this function to create a text combo box, you should only manipulate its data source with the following convenience methods: `append_text()`, `insert_text()`, `prepend_text()` and `remove_text()`.

Signals

The "changed" gtk.ComboBox Signal

```
    def callback(combobox, user_param1, ...)
```

`combobox` :	the combo box that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

Note

This signal is available in PyGTK 2.4 and above.

The "changed" signal is emitted when a new item in the combo box is selected.

gtk.ComboBoxEntry

gtk.ComboBoxEntry   a text entry field with a dropdown list (new in PyGTK 2.4)

Synopsis

```
class gtk.ComboBoxEntry(gtk.ComboBox, gtk.CellLayout):
    gtk.ComboBoxEntry(model=None, column=-1)
    def set_text_column(text_column)
    def get_text_column()
Functions

    def gtk.combo_box_entry_new_text()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.ComboBox
            +-- gtk.ComboBoxEntry (implements gtk.CellLayout)
```

Properties

"text−column"	Read−Write	The column in the `gtk.TreeModel` to get the strings from.

Description

Note

This widget is available in GTK+ 2.4 and PyGTK 2.4 and above.

The `gtk.ComboBoxEntry` is a replacement for the `gtk.Combo`. The `gtk.ComboBoxEntry` is subclassed from `gtk.ComboBox` and implements the `gtk.CellLayout` interface; refer to their descriptions for a number of useful methods and properties for managing the contents of a combo box entry. A `gtk.ComboBoxEntry` also contains a child `gtk.Entry` accessed by using the child attribute of the combo box entry:

```
  entry = comboboxentry.child
```

A `gtk.ComboBoxEntry` is created with the gtk.ComboBoxEntry() constructor. The constructor can also be called with the optional parameters *`model`* (a `gtk.TreeModel` – default `None`) and *`column`* (a number of a column in *`model`* – default 0). If no `gtk.TreeModel` is specified it can be added later with the `set_model()` method. The text column can be changed using the `set_text_column()` method. A new combo box entry is created and packed with a `gtk.CellRendererText` but no attribute mappings are set on the cell renderer.

Alternatively, the `gtk.combo_box_entry_new_text()` function creates a `gtk.ComboBoxEntry` with an associated `gtk.ListStore` model and the text column attribute mapping set to 0. In this combo box entry each list item is a text string that can be selected. The convenience methods `gtk.ComboBox.append_text()`, `gtk.ComboBox.prepend_text()`, `gtk.ComboBox.insert_text()` and `gtk.ComboBox.remove_text()` can be used to manage the contents of the `gtk.ComboBoxEntry`. Using the `gtk.combo_box_entry_new_text()` function is equivalent to:

```
  liststore = gtk.ListStore(gobject.TYPE_STRING)
  comboboxentry = gtk.ComboBoxEntry(liststore, 0)
```

Constructor

```
    gtk.ComboBoxEntry(model=None, column=-1)
```

`model` :	The `gtk.TreeModel` to associate with the combo box entry, or `None`
`column` :	The number of the column to use for setting the strings of the combo box entry.
Returns :	A new `gtk.ComboBoxEntry`.

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new `gtk.ComboBoxEntry` that has a `gtk.Entry` as child and associated with the `gtk.TreeModel` specified by *`model`* and using the column of *`model`* specified by *`column`* to retrieve strings. If *`model`* was not specified it defaults to `None` but you can change the model using the `gtk.ComboBox.set_model()` method. If *`column`* was not specified it defaults to −1 meaning the text column is unset but you can change it using the `set_text_column()` method. Once the text column is set either in the constructor or using the `set_text_column()` method it cannot be changed. A ValueError exception is thrown if *`column`* is outside the range of column numbers for *`model`*.

Methods

gtk.ComboBoxEntry.set_text_column

```
    def set_text_column(text_column)
```

`text_column` :	A column in the associated `gtk.TreeModel` to use as the data source for the strings.

Note

This method is available in PyGTK 2.4 and above.

The `set_text_column()` method sets the "text−column" property to the value of *`tree_column`* only if the text column has not been set (that is, "text−column" is −1). The value of *`tree_column`* is the number of the tree model column used as the data source for the strings of the combo box entry.

gtk.ComboBoxEntry.get_text_column

```
    def get_text_column()
```

Returns :	The number of the column in the associated `gtk.TreeModel` used as the data source for the combo box entry.

Note

This method is available in PyGTK 2.4 and above.

The `get_text_column()` method returns the number of the `gtk.TreeModel` column that is used as the data source for the strings of the combo box entry.

Functions

gtk.combo_box_entry_new_text

```
    def gtk.combo_box_entry_new_text()
```

Returns :	A new `gtk.ComboBoxEntry` widget.

Note

This function is available in PyGTK 2.4 and above.

The `gtk.combo_box_entry_new_text()` function is a convenience function which constructs a new `gtk.ComboBoxEntry`, just displaying strings. If you use this function to create a combo box entry, you should only manipulate its `gtk.TreeModel` data source with the following `gtk.ComboBox` convenience methods: `gtk.ComboBox.append_text()`, `gtk.ComboBox.insert_text()`, `gtk.ComboBox.prepend_text()` and `gtk.ComboBox.remove_text()`.

gtk.Container

gtk.Container a base class for widgets that contain other widgets

Synopsis

```
class gtk.Container(gtk.Widget):
    def set_border_width(border_width)
    def get_border_width()
    def add(widget)
    def remove(widget)
    def set_resize_mode(resize_mode)
    def get_resize_mode()
    def check_resize()
    def forall(callback, callback_data)
    def foreach(callback, callback_data)
    def get_children()
    def propagate_expose(child, event)
    def set_focus_chain(focusable_widgets)
    def get_focus_chain()
    def unset_focus_chain()
    def set_reallocate_redraws(needs_redraws)
    def set_focus_child(child)
    def set_focus_vadjustment(adjustment)
    def get_focus_vadjustment()
    def set_focus_hadjustment(adjustment)
    def get_focus_hadjustment()
    def resize_children()
    def child_type()
    def add_with_properties(widget, first_prop_name, first_prop_value, ...)
    def child_set(child, first_prop_name, first_prop_value, ...)
    def child_get(child, first_prop_name, ...)
    def child_set_property(child, property_name, value)
    def child_get_property(child, property_name)
```

```
Functions

    def gtk.container_class_install_child_property(klass, property_id, pspec)
    def gtk.container_class_list_child_properties(klass)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
```

Properties

"border–width"	Read/Write	The width of the empty border outside the containers children.
"resize–mode"	Read/Write	Specify how resize events are handled. One of: `gtk.RESIZE_PARENT`, `gtk.RESIZE_QUEUE` or `gtk.RESIZE_IMMEDIATE`
"child"	Read	The child widget in the container

Attributes

"border_width"	Read	The width of the empty border outside the containers children.
"resize_mode"	Read	Specify how resize events are handled. One of: `gtk.RESIZE_PARENT`, `gtk.RESIZE_QUEUE` or `gtk.RESIZE_IMMEDIATE`
"focus_child"	Read	The child widget that has the focus
"need_resize"	Read	If TRUE the container needs resizing
"reallocate_redraws"	Read	if TRUE redraw the container when a child gets reallocated
"has_focus_chain"	Read	If TRUE the container had its focus chain explicitly set

Signal Prototypes

"add"	`def callback(`*container*`, `*widget*`, `*user_param1*`, ...)`
"check–resize"	`def callback(`*container*`, `*user_param1*`, ...)`
"remove"	`def callback(`*container*`, `*widget*`, `*user_param1*`, ...)`
"set–focus–child"	`def callback(`*container*`, `*widget*`, `*user_param1*`, ...)`

Description

The `gtk.Container` class provides common attributes and methods for a large number of widget subclasses that manage the layout of other widgets within the area of a window.

A `PyGTK` user interface is constructed by nesting widgets inside widgets. Container widgets are the inner nodes in the resulting tree of widgets: they contain other widgets. So, for example, you might have a `gtk.Window` containing a `gtk.Frame` containing a `gtk.Label`. If you wanted an image instead of a textual label inside the frame, you might replace the `gtk.Label` widget with a `gtk.Image` widget.

There are two major kinds of container widgets. Both are subclasses of the abstract `gtk.Container` base class.

The first type of container widget has a single child widget and derives from `gtk.Bin`. These containers are decorators, that add some kind of functionality to the child. For example, a `gtk.Button` makes its child into a clickable button; a `gtk.Frame` draws a frame around its child and a `gtk.Window` places its child widget inside a top−level window.

The second type of container can have more than one child; its purpose is to manage layout. This means that these containers assign sizes and positions to their children. For example, a `gtk.HBox` arranges its children in a horizontal row, and a `gtk.Table` arranges the widgets it contains in a two−dimensional grid.

To fulfill its task, a layout container must negotiate the size requirements with its parent and its children. This negotiation is carried out in two phases, size requisition and size allocation.

Size Requisition

The size requisition of a widget is it's desired width and height. This is represented by a `gtk.Requisition`.

How a widget determines its desired size depends on the widget. A `gtk.Label`, for example, requests enough space to display all its text. Container widgets generally base their size request on the requisitions of their children.

The size requisition phase of the widget layout process operates top−down. It starts at a top−level widget, typically a GtkWindow. The top−level widget asks its child for its size requisition by calling gtk_widget_size_request(). To determine its requisition, the child asks its own children for their requisitions and so on. Finally, the top−level widget will get a requisition back from its child.

Size Allocation

When the top−level widget has determined how much space its child would like to have, the second phase of the size negotiation, size allocation, begins. Depending on its configuration (see the `gtk.Window.set_resizable()` method), the top−level widget may be able to expand in order to satisfy the size request or it may have to ignore the size request and keep its fixed size. It then tells its child widget how much space it gets by calling the `size_allocate()` method. The child widget divides the space among its children and tells each child how much space it got, and so on. Under normal circumstances, a `gtk.Window` will always give its child the amount of space the child requested.

A child's size allocation is represented by a `gtk.gdk.Rectangle` that contains not only a width and height, but also a position (i.e. X and Y coordinates), so that containers can tell their children not only how much space is available, but also where they are positioned inside the space available to the container.

Widgets are required to honor the size allocation they receive; a size request is only a request, and widgets must be able to cope with any size.

Child Properties

`gtk.Container` introduces child properties − these are object properties that are not specific to either the container or the contained widget, but rather to their relation. Typical examples of child properties are the "position" or "pack−type" of a widget which is contained in a `gtk.Box`.

Use the `gtk.container_class_install_child_property()` function to install child properties for a container class and the `gtk.container_class_list_child_properties()` function to get information about existing child properties.

To set the value of a child property, use the `child_set_property()`, or `child_set()` methods. To obtain the value of a child property, use the `child_get_property()`, or `child_get()` methods. To emit notification about child property changes, use the `gtk.Widget.child_notify()` method.

Methods

gtk.Container.set_border_width

```
def set_border_width(border_width)
```

`border_width` :	The amount of blank space to leave *outside* the container. Valid values are in the range 0–65535 pixels.

The `set_border_width()` method sets the "border–width" property of the container. The border width of a container is the amount of space to leave around the outside of the container. The only exception to this is `gtk.Window`; because toplevel windows can't leave space outside, they leave the space inside. The border is added on all sides of the container.

gtk.Container.get_border_width

```
def get_border_width()
```

Returns :	the current border width

The `get_border_width()` method retrieves the value of the "border–width" property of the container. See `set_border_width()`.

gtk.Container.add

```
def add(widget)
```

`widget` :	a widget to be placed inside the container

The `add()` method adds *`widget`* to the container. This method is typically used for simple containers such as `gtk.Window`, `gtk.Frame`, or `gtk.Button` that hold a single child widget. For layout containers that handle multiple children such as `gtk.Box` or `gtk.Table`, this function will pick default packing parameters that may not be correct. Containers that handle multiple children usually have additional methods such as `gtk.Box.pack_start()` and `gtk.Table.attach()` as an alternative to `add()`. Adding a widget to a container usually results in the resizing and redrawing of the container contents.

gtk.Container.remove

```
def remove(widget)
```

`widget` :	a current child of *`container`*

The `remove()` method removes *`widget`* from the container. *`widget`* must be inside the container. Note that the container will own a reference to *`widget`*, and that this may be the last reference held; so removing a widget from its container can cause that widget to be destroyed. If you want to use *`widget`* again, you should add a reference to it.

gtk.Container.set_resize_mode

```
def set_resize_mode(resize_mode)
```

`resize_mode` :	the new resize mode.

The `set-resize_mode()` method sets the "resize=mode" property of the container. The resize mode of a container determines whether a resize request will be passed to the container's parent (`gtk.RESIZE_PARENT`), queued for later execution (`gtk.RESIZE_QUEUE`) or executed immediately (`gtk.RESIZE_IMMEDIATE`).

gtk.Container.get_resize_mode

```
def get_resize_mode()
```

Returns :	the current resize mode

The get_resize_mode() method returns the value of the "resize–mode" property for of the container. See `set_resize_mode()`.

gtk.Container.check_resize

```
def check_resize()
```

The `check_resize()` method emits the "check–resize" signal on the container.

gtk.Container.forall

```
def foreach(callback, callback_data=None)
```

`callback` :	a callback
`callback_data` :	the callback user data

The `forall()` method arranges to invoke *`callback`* on each child of the container including children that are considered "internal" (implementation details of the container). "Internal" children generally weren't added by the user of the container, but were added by the container implementation itself. Most applications should use the `foreach()` method, rather than the `forall()` method.

gtk.Container.foreach

```
def foreach(callback, callback_data=None)
```

`callback` :	a callback
`callback_data` :	the callback user data

The `foreach()` method arranges to invoke *`callback`* on each non–internal child of the container.

gtk.Container.get_children

```
def get_children()
```

Returns :	a list of the container's non–internal children.

The `get_children()` method returns the the container's non–internal children.

gtk.Container.propagate_expose

```
def propagate_expose(child, event)
```

`child` :	a child of the container
`event` :	a expose event sent to the container

The `propagate_expose`() method sends synthetic expose events to all children that don't have their own `gtk.gdk.Window`s when the container receives an expose event.

The `propagate_expose()` takes care of deciding whether an expose event needs to be sent to the child, intersecting the event's area with the child area, and sending the event.

In most cases, a container can simply either simply inherit the expose implementation from `gtk.Container`, or, do some drawing and then chain to the expose implementation from `gtk.Container`.

gtk.Container.set_focus_chain

```
def set_focus_chain(focusable_widgets)
```

`focusable_widgets` :	a list or tuple containing a chain of focusable widgets.

The `set_focus_chain`() method sets a focus chain, overriding the one computed automatically by GTK. In principle each widget in the chain should be a descendant of the container, but this is not enforced by this method, since it's allowed to set the focus chain before you pack the widgets, or have a widget in the chain that isn't always packed. The necessary checks are done when the focus chain is actually traversed.

gtk.Container.get_focus_chain

```
def get_focus_chain()
```

Returns :	a list containing the widgets in the focus chain if the focus chain of the container has been set explicitly or None if no focus chain has been explicitly set.

The `get_focus_chain`() method retrieves the focus chain of the container, if one has been set explicitly. If no focus chain has been explicitly set, GTK computes the focus chain based on the positions of the children. In that case, the method returns None.

gtk.Container.unset_focus_chain

```
def unset_focus_chain()
```

The `unset_focus_chain`() method removes a focus chain explicitly set with `set_focus_chain()`.

gtk.Container.set_reallocate_redraws

```
def set_reallocate_redraws(needs_redraws)
```

`needs_redraws` :	the new value for the container's reallocate_redraws attribute.

The `set_reallocate_redraws`() method sets the reallocate_redraws attribute of the container to the value of *`needs_redraws`*. Containers requesting reallocation redraws get automatically redrawn if any of their children change allocation.

gtk.Container.set_focus_child

```
def set_focus_child(child)
```

`child` :	the child widget that will get the focus.

The `set_focus_child()` method emits the "set−focus−child" signal that arranges for the child widget referenced by *`child`* to get the focus and recalculates the container adjustments.

gtk.Container.set_focus_vadjustment

```
def set_focus_vadjustment(adjustment)
```

`adjustment` :	The new vertical focus adjustment

The `set_focus_vadjustment()` method sets the vertical focus adjustment to the value of *`adjustment`*.

gtk.Container.get_focus_vadjustment

```
def get_focus_vadjustment()
```

Returns :	the vertical focus adjustment, or `None` if none has been set.

The get_focus_vadjustment() method retrieves the vertical focus adjustment for the container. See the `set_focus_vadjustment()` method.

gtk.Container.set_focus_hadjustment

```
def set_focus_hadjustment(adjustment)
```

`adjustment` :	The new horizontal focus adjustment

The `set_focus_hadjustment()` method sets the horizontal focus adjustment to the value of *`adjustment`*.

gtk.Container.get_focus_hadjustment

```
def get_focus_hadjustment()
```

Returns :	the horizontal focus adjustment, or `None` if none has been set.

The `get_focus_hadjustment()` method retrieves the horizontal focus adjustment for the container. See `set_focus_hadjustment()`.

gtk.Container.resize_children

```
def resize_children()
```

The `resize_children()` method causes the container to recalculate its size and its children's sizes.

gtk.Container.child_type

```
def child_type()
```

Returns :	a type.

The `child_type()` method returns the type of the children that can be added to the container. Note that this may return a void type to indicate that no more children can be added, e.g. for a gtk.Paned which already has two children or a gtk.Window that already has a child.

gtk.Container.add_with_properties

```
def add_with_properties(widget, first_prop_name, first_prop_value, ...)
```

widget :	a widget to be added
first_prop_name :	the first property name
first_prop_value :	a value for the first property
... :	additional property name and value pairs

The `add_with_properties`() method adds the child widget specified by *widget* to the container while allowing the setting of zero or more container child property values at the same time. Containers supporting add with settable child properties are: gtk.Box, gtk.Fixed, gtk.Notebook and gtk.Table.

For example the following adds a button to a gtk.Fixed layout widget and sets the child properties "x" and "y" specifying the child position in the layout:

```
fixed.add_with_properties(button, "x", 10, "y", 20")
```

gtk.Container.child_set

```
def child_set(child, first_prop_name, ...)
```

child :	the child widget
first_prop_name :	the first property name
first_prop_value :	the value of the first property
... :	additional property name and value pairs

The `child_set`() method sets the properties for *child* using the given property name and value pairs.

gtk.Container.child_get

```
def child_get(child, first_prop_name, ...)
```

child :	the child widget to get the child properties for
first_prop_name :	the first property name
... :	additional property names
Returns :	a tuple containing the property values requested

The `child_get`() method retrieves the requested container child properties for *child*.

gtk.Container.child_set_property

```
def child_set_property(child, property_name, value)
```

child :	the child widget
property_name :	the child property name
value :	a value to associate with the property

The `child_set_property`() method sets the property name specified by *property_name* with the value specified in *value*.

gtk.Container.child_get_property

```
def child_get_property(child, property_name)
```

`child` :	the child widget
`property_name` :	the child property name
Returns :	the value of the child property for the widget

The `child_get_property()` method retrieves the value of the child property specified by *`property_name`* for the widget *`child`*.

Functions

gtk.container_class_install_child_property

```
def gtk.container_class_install_child_property(klass, property_id, pspec)
```

`klass` :	a `gtk.Container` class or instance.
`property_id` :	an integer property ID
`pspec` :	a 4–tuple containing a parameter specification

Note

This function is available in PyGTK 2.4 and above.

The `gtk.container_class_install_child_property()` function installs a child property for the container class specified by *`klass`* using the integer ID specified by *`property_id`*. *`pspec`* is a 4–tuple containing the parameter specification:

- a string specifying the name of the property
- an object specifying the property type
- a string specifying the nickname for the property or `None`
- a string specifying the short deciription for the property or `None`

gtk.container_class_list_child_properties

```
def gtk.container_class_list_child_properties(klass)
```

`klass` :	a `gtk.Container` class or instance.
Returns :	a tuple containing the list of child properties

Note

This function is available in PyGTK 2.4 and above.

The `gtk.container_class_list_child_properties()` function returns a tuple containing the child properties of the container class specified by *`klass`*.

Signals

The "add" gtk.Container Signal

```
    def callback(container, widget, user_param1, ...)
```

container :	the container that received the signal
widget :	the child widget
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "add" signal is emitted when *widget* is added to the *container*.

The "check–resize" gtk.Container Signal

```
    def callback(container, user_param1, ...)
```

container :	the container that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "check–resize" signal is emitted when the `check_resize()` method is called forcing the recalculation of the container and its children. See the set_resize_mode() method for details.

The "remove" gtk.Container Signal

```
    def callback(container, widget, user_param1, ...)
```

container :	the container that received the signal
widget :	the child widget
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "remove" signal is emitted when *widget* is removed from *container*.

The "set–focus–child" gtk.Container Signal

```
    def callback(container, widget, user_param1, ...)
```

container :	the container that received the signal
widget :	the child widget
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "set–focus–child" signal is emitted when the set_focus_child() method is called. *widget* is set as the child in *container* with the focus.

Prev	Up	Next
gtk.ComboBoxEntry	Home	gtk.Curve

gtk.Curve

Prev	**The gtk Class Reference**	Next

gtk.Curve

gtk.Curve — allows direct editing of a curve.

Synopsis

```
class gtk.Curve(gtk.DrawingArea):
    gtk.Curve()
    def reset()
    def set_gamma(gamma)
    def set_range(min_x, max_x, min_y, max_y)
    def get_vector(size=-1)
    def set_vector(vector)
    def set_curve_type(type)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.DrawingArea
        +-- gtk.Curve
```

Properties

"curve–type"	Read/Write	The curve type. One of linear (`gtk.CURVE_TYPE_LINEAR`), spline interpolated (`gtk.CURVE_TYPE_SPLINE`), or free–form (`gtk.CURVE_TYPE_FREE`).
"min–x"	Read/Write	The minimum possible value for X
"max–x"	Read/Write	The maximum possible value for X
"min–y"	Read/Write	The minimum possible value for Y
"max–y"	Read/Write	The maximum possible value for Y

Signal Prototypes

"curve–type–changed"	`def callback(curve, widget, user_param1, ...)`

Description

Note

This widget is considered too specialized or little–used for PyGTK, and will in the future be moved to some other package. If your application needs this widget, feel free to use it, as the widget does work and is useful in some applications; it's just not of general interest. However, it will eventually move out of the PyGTK distribution.

The `gtk.Curve` widget allows the user to edit a curve covering a range of values. It is typically used to fine–tune color balances in graphics applications like the Gimp. The `gtk.Curve` widget has 3 modes of operation – spline, linear and free. In spline mode the user places points on the curve which are automatically connected together into a smooth curve. In linear mode the user places points on the curve which are connected by straight lines. In free mode the user can draw the points of the curve freely, and they are not connected at all.

Constructor

```
gtk.Curve()
```

Returns :	a new gtk.Curve object

Creates a new `gtk.Curve` object

Methods

gtk.Curve.reset

```
def reset()
```

The `reset()` method resets the curve to a straight line from the minimum x and y values to the maximum x and y values (i.e. from the bottom–left to the top–right corners). The curve type is not changed.

gtk.Curve.set_gamma

```
def set_gamma(gamma)
```

`gamma` :	the gamma value

The `set_gamma()` method recomputes the entire curve using the value in *`gamma`*. A gamma value of 1 results in a straight line. Values greater than 1 result in a curve above the straight line. Values less than 1 result in a curve below the straight line. The curve type is changed to `gtk.CURVE_TYPE_FREE`.

gtk.Curve.set_range

```
def set_range(min_x, max_x, min_y, max_y)
```

`min_x` :	the new minimum x value
`max_x` :	the maximum x value.
`min_y` :	the new minimum y value
`max_y` :	the maximum y value.

The `set_range()` method sets the "min–x", "min–y", "max–x" and "max–y" properties from *`min_x`*, *`min_y`*, *`max_x`*, and *`max_y`*. The curve is also reset with a call to `reset()`.

gtk.Curve.get_vector

```
def set_vector(size=-1)
```

`size` :	the number of points to return or −1 to return all the points in the curve.
Returns :	a tuple containing the points

The `get_vector()` method returns a tuple of points representing the curve. The number of points to return is specified by *`size`*; if *`size`* is −1 all the points in the curve are returned.

gtk.Curve.set_vector

```
    def get_vector(vector)
```

`vector` :	a list or tuple containing the points of the curve

The `set_vector()` method sets the curve using the points in *`vector`*. The curve type is set to `gtk.CURVE_TYPE_FREE`.

gtk.Curve.set_curve_type

```
    def set_curve_type(type)
```

`type` :	the new curve type: `gtk.CURVE_TYPE_LINEAR`, `gtk.CURVE_TYPE_SPLINE` or `gtk.CURVE_TYPE_FREE`

The `set_curve_type()` method sets the "curve−type" property with the value of *`type`*. The curve type must be one of `gtk.CURVE_TYPE_LINEAR`, `gtk.CURVE_TYPE_SPLINE` or `gtk.CURVE_TYPE_FREE`. The curve will remain unchanged except when changing from a free curve to a linear or spline curve, in which case the curve will be changed as little as possible.

Signals

The "curve−type−changed" gtk.Curve Signal

```
    def callback(curve, user_param1, ...)
```

`curve` :	the curve that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "curve−type−changed" signal is emitted when the curve type has been changed. The curve type can be changed explicitly with a call to `set_curve_type()`. It is also changed as a side−effect of calling `reset()` or `set_gamma()`

gtk.Dialog

gtk.Dialog popup windows for user information and action

Synopsis

```
class gtk.Dialog(gtk.Window):
    gtk.Dialog(title=None, parent=None, flags=0, buttons=None)
    def add_action_widget(child, response_id)
    def add_button(button_text, response_id)
    def add_buttons(buttons)
    def set_response_sensitive(response_id, setting)
    def set_default_response(response_id)
    def set_has_separator(setting)
    def get_has_separator()
    def response(response_id)
    def run()
    def set_alternative_button_order(new_order)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Window
            +-- gtk.Dialog
```

Properties

"has−separator"	Read/Write	If `TRUE`, the dialog has a separator bar above its buttons

Style Properties

"button−spacing"	Read	The spacing between buttons in pixels.
"action−area−border"	Read	The width of the vborder around the button area in pixels.
"content−area−border"	Read	The width of the border around the main dialog area in pixels.

Attributes

"vbox"	Read	A `gtk.VBox` that is the main container of the dialog − all the other widgets are packed in it.
"action_area"	Read	A `gtk.HBox` that contains the buttons of the dialog.

Signal Prototypes

"close"	`def callback(dialog, user_param1, ...)`
"response"	`def callback(dialog, response_id, user_param1, ...)`

Description

Dialog boxes are a convenient way to prompt the user for a small amount of input, e.g. to display a message, ask a question, or anything else that does not require extensive effort on the user's part. Dialogs are organized as a window split vertically. The top section is a `gtk.VBox`, and is where widgets such as a `gtk.Label` or a `gtk.Entry` should be packed. The bottom area is known as the action_area which is generally used for packing buttons into the dialog which may perform functions such as cancel, ok, or apply. The two areas are separated by a `gtk.HSeparator`.

The `gtk.Dialog` boxes are created with a call to `gtk.Dialog()`() that sets the dialog title, some convenient flags, and adds simple buttons. In a newly created dialog, the two primary areas of the window can be accessed as the vbox and action_area attributes, as can be seen from the example, below. A modal dialog (that is, one which freezes the rest of the application from user input), can be created by passing the `gtk.DIALOG_MODAL` flag to the `gtk.Dialog()` constructor or by calling `set_modal()` on the dialog.

If you add buttons to `gtk.Dialog` using `gtk.Dialog()`, `add_button()`, or `add_action_widget()`, clicking the button will emit a signal called "response" with a response ID that you specified. PyGTK will never assign a meaning to positive response IDs; these are entirely user–defined. But for convenience, you can use the pre–defined response IDs (these all have values less than zero):

- `gtk.RESPONSE_NONE`
- `gtk.RESPONSE_REJECT`
- `gtk.RESPONSE_ACCEPT`
- `gtk.RESPONSE_DELETE_EVENT`
- `gtk.RESPONSE_OK`
- `gtk.RESPONSE_CANCEL`
- `gtk.RESPONSE_CLOSE`
- `gtk.RESPONSE_YES`
- `gtk.RESPONSE_NO`
- `gtk.RESPONSE_APPLY`
- `gtk.RESPONSE_HELP`

If a dialog receives a delete event, the "response" signal will be emitted with a response ID of `gtk.RESPONSE_NONE`.

If you want to block waiting for a dialog to return before returning control flow to your code, you can call `run()`. This function enters a recursive main loop and waits for the user to respond to the dialog, returning the response ID corresponding to the button the user clicked.

Constructor

```
gtk.Dialog(title=None, parent=None, flags=0, buttons=None)
```

`title` :	The title of the dialog, or `None`
`parent` :	The transient parent of the dialog, or `None`
`flags` :	flags that control the operation of the dialog
`buttons` :	a tuple containing button text/response id pairs or `None`
Returns :	a new `gtk.Dialog`

Creates a new `gtk.Dialog` with the title text specified by *`title`* (or `None` for the default title; see `gtk.Window.set_title()`) and transient parent window specified by *`parent`* (or `None` for none; see `gtk.Window.set_transient_for()`). The *`flags`* argument can be used to make the dialog modal (`gtk.DIALOG_MODAL`) and/or to have it destroyed along with its transient parent (`gtk.DIALOG_DESTROY_WITH_PARENT`) and/or remove the separator (`gtk.DIALOG_NO_SEPARATOR`). After *`flags`*, a tuple of button text/response ID pairs should be listed, or `None` (the default value) is no buttons are needed. The button text can be either a stock ID such as `gtk.STOCK_OK`, or some arbitrary text. A response ID can be any positive number, or one of the pre–defined values:

- `gtk.RESPONSE_NONE`
- `gtk.RESPONSE_REJECT`
- `gtk.RESPONSE_ACCEPT`
- `gtk.RESPONSE_DELETE_EVENT`
- `gtk.RESPONSE_OK`
- `gtk.RESPONSE_CANCEL`
- `gtk.RESPONSE_CLOSE`
- `gtk.RESPONSE_YES`

- `gtk.RESPONSE_NO`
- `gtk.RESPONSE_APPLY`
- `gtk.RESPONSE_HELP`

If the user clicks one of these dialog buttons, the `gtk.Dialog` will emit the "response" signal with the corresponding response ID. If a `gtk.Dialog` receives the "delete_event" signal, it will emit "response" with a response ID of `gtk.RESPONSE_DELETE_EVENT`. However, destroying a dialog does not emit the "response" signal; so be careful relying on "response" when using the `gtk.DIALOG_DESTROY_WITH_PARENT` flag. Buttons are added from left to right, so the first button in the list will be the leftmost button in the dialog.

Here's a simple example:

```
  dialog = gtk.Dialog("My dialog",
                      main_app_window,
                      gtk.DIALOG_MODAL | gtk_.DIALOG_DESTROY_WITH_PARENT,
                      (gtk.STOCK_OK, gtk.RESPONSE_ACCEPT,
                       gtk.STOCK_CANCEL, gtk.RESPONSE_REJECT))
```

Methods

gtk.Dialog.add_action_widget

```
    def add_action_widget(child, response_id)
```

`child` :	an activatable widget
`response_id` :	a response ID

The `add_action_widget()` method adds an activatable widget to the action area of a `gtk.Dialog`, connecting a signal handler that will emit the "response" signal on the dialog when the widget is activated. The widget is appended to the end of the dialog's action area. If you want to add a non–activatable widget, simply pack it into the `action_area`.

gtk.Dialog.add_button

```
    def add_button(button_text, response_id)
```

`button_text` :	the text of the button, or a stock ID
`response_id` :	the response ID for the button
Returns :	the button widget that was added

The `add_button()` method adds a button with the text specified by *`button_text`* (or a stock button, if *`button_text`* is a stock ID) and sets things up so that clicking the button will emit the "response" signal with the specified *`response_id`*. The button is appended to the end of the dialog's action area. The button widget is returned, but usually you don't need it.

gtk.Dialog.add_buttons

```
    def add_buttons(buttons)
```

`buttons` :	a tuple containing 2–tuples each containing button text (or stock ID) and a response id

The `add_buttons()` method adds several buttons to the `gtk.Dialog` using the data specified in *`buttons`*. This method is the same as calling the `gtk.Dialog.add_button()` repeatedly. buttons is a tuple containing 2–tuples specifying the data for one button – button text and a response ID integer.

gtk.Dialog.set_response_sensitive

```
    def set_response_sensitive(response_id, setting)
```

`response_id` :	a response ID
`setting` :	the new value for sensitive

The `set_response_sensitive`() method calls the `gtk.Window.set_sensitive()` method with the specified *`response_id`* for each widget in the dialog's action area. This method is a convenience function to sensitize/desensitize all dialog buttons at once.

gtk.Dialog.set_default_response

```
    def set_default_response(response_id)
```

`response_id` :	a response ID

The `set_default_response`() method sets the last widget in the dialog's action area with the specified *`response_id`* as the default widget for the dialog. Pressing **Enter** normally activates the default widget.

gtk.Dialog.set_has_separator

```
    def set_has_separator(setting)
```

`setting` :	If `TRUE` use a separator

The `set_has_separator`() method sets the "has−separator" property to the value of *`setting`*. If *`setting`* is `TRUE` (the default value) the dialog has a separator above the buttons.

gtk.Dialog.get_has_separator

```
    def get_has_separator()
```

Returns :	the value of the "has−separator" property

The `get_has_separator`() method returns the value of the "has−separator" property.

gtk.Dialog.response

```
    def response(response_id)
```

`response_id` :	response ID

The `response`() method emits the "response" signal with the value specified in *`response_id`*. This method is used to indicate that the user has responded to the dialog in some way; typically either you or `gtk.Dialog.run()` will be monitoring the "response" signal and take appropriate action.

gtk.Dialog.run

```
    def run()
```

Returns :	a response ID

The `run`() method blocks in a recursive main loop until the dialog either emits the "response" signal, or is destroyed. If the dialog is destroyed, the `run`() method returns `gtk.RESPONSE_NONE`; otherwise, it returns the response ID from the "response" signal emission. Before entering the recursive main loop, the `run`() method calls the `gtk.Widget.show()` on the dialog for you. Note that you still need to show any children of the dialog yourself.

During the `run()` method, the default behavior of "delete_event" is disabled; if the dialog receives a "delete_event", it will not be destroyed as windows usually are, and the `run()` method will return `gtk.RESPONSE_DELETE_EVENT`. Also, during the `run()` method the dialog will be modal. You can force the `run()` method to return at any time by calling `response()` to emit the "response" signal. Destroying the dialog during the `run()` method is a very bad idea, because your post−run code won't know whether the dialog was destroyed or not.

After the `run()` method returns, you are responsible for hiding or destroying the dialog as needed.

gtk.Dialog.set_alternative_button_order

```
def set_alternative_button_order(new_order)
```

`new_order` :	a sequence containing response id integer values

Note

This method is available in PyGTK 2.6 and above.

The `set_alternative_button_order()` method sets an alternative button order for the dialog based on the sequence of response ids specified by *`new_order`*. If the "gtk−alternative−button−order" property of the `gtk.Settings` object is set to `TRUE`, the dialog buttons are reordered according to the order of the response ids passed to this method.

By default, GTK+ dialogs use the button order advocated by the Gnome Human Interface Guidelines with the affirmative button at the far right, and the cancel button left of it. But the builtin GTK+ dialogs and `gtk.MessageDialog`s do provide an alternative button order, which is more suitable on some platforms, e.g. Windows.

Use this method after adding all the buttons to your dialog, as the following example shows:

```
settings = gtk.settings_get_default()
settings.set_property('gtk-alternative-button-order', True)

dialog = gtk.Dialog()
cancel_button = dialog.add_button(gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL)
ok_button = dialog.add_button(gtk.STOCK_OK, gtk.RESPONSE_OK)
ok_button.grab_default()
help_button = dialog.add_button(gtk.STOCK_HELP, gtk.RESPONSE_HELP)

dialog.set_alternative_button_order([gtk.RESPONSE_OK, gtk.RESPONSE_CANCEL,
                                     gtk.RESPONSE_HELP])
```

Signals

The "close" gtk.Dialog Signal

```
def callback(dialog, user_param1, ...)
```

`dialog` :	the dialog that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "close" signal is emitted when the dialog is closed.

The "response" gtk.Dialog Signal

```
  def callback(dialog, response_id, user_param1, ...)
```

dialog :	the dialog that received the signal
response_id :	the response id received by the dialog
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "response" signal is emitted when an action_area widget is activated (button "clicked"), the dialog receives a delete_event or the application calls the `response()` method. When a delete_event triggers the "response" signal the *response_id* will be `gtk.RESPONSE_NONE`.

gtk.DrawingArea

gtk.DrawingArea — a widget for custom user interface elements.

Synopsis

```
class gtk.DrawingArea(gtk.Widget):
    gtk.DrawingArea()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.DrawingArea
```

Description

The `gtk.DrawingArea` widget is used for creating custom user interface elements. It's essentially a blank widget containing a `gtk.gdk.Window` that you can draw on. The contained `gtk.gdk.Window` is accessed using the `gtk.Widget` "window" attribute as:

```
  gdkwindow = drawingarea.window
```

Since a `gtk.gdk.Window` is a subclass of `gtk.gdk.Drawable`, all of the `gtk.gdk.Drawable` methods are available for drawing on the `gtk.DrawingArea`'s gdkwindow.

After creating a drawing area, the application may want to connect to:

- Mouse and button press signals to respond to input from the user. Use the `gtk.Widget.add_events()` method to enable events you wish to receive. To receive keyboard events, you will need to set the `gtk.CAN_FOCUS` flag on the drawing area
- The "realize" signal to take any necessary actions when the widget is instantiated on a particular display.
- The "configure_event" signal to take any necessary actions when the widget changes size.

- The "expose_event" signal to handle redrawing the contents of the widget when a drawing area first comes on screen, or when it's covered by another window and then uncovered (exposed). You can also force an expose event by adding to the "damage region" of the drawing area's window using the `gtk.Widget.queue_draw_area()` method.

See the Drawing Area chapter in the tutorial for more information on using a `gtk.DrawingArea`.

Sometimes a `gtk.Image` is a useful alternative to a drawing area. You can put a `gtk.gdk.Pixmap` in the `gtk.Image` and draw to the `gtk.gdk.Pixmap`, calling the `gtk.Widget.queue_draw()` method on the `gtk.Image` when you want to refresh to the screen.

Constructor

```
  gtk.DrawingArea()
```

Returns :	a new drawingarea widget

Creates a new `gtk.DrawingArea` widget.

Prev	Up	Next
gtk.Dialog	Home	gtk.Editable

gtk.Editable

gtk.Editable — an interface for text−editing widgets.

Synopsis

```
class gtk.Editable(gobject.GInterface):
    def select_region(start, end)
    def get_selection_bounds()
    def insert_text(text, position=0)
    def delete_text(start_pos, end_pos)
    def get_chars(start_pos, end_pos)
    def cut_clipboard()
    def copy_clipboard()
    def paste_clipboard()
    def delete_selection()
    def set_position(position)
    def get_position()
    def set_editable(is_editable)
    def get_editable()
```

Signal Prototypes

"changed"	`def callback(editable, user_param1, ...)`
"delete−text"	`def callback(editable, start, end, user_param1, ...)`
"insert−text"	`def callback(editable, new_text, new_text_length, position, user_param1, ...)`

Description

gtk.Editable is an interface for text editing widgets, such as gtk.Entry. The editable class contains methods for generically manipulating an editable widget, a large number of action signals used for key bindings, and several signals that an application can connect to to modify the behavior of a widget.

Methods

gtk.Editable.select_region

```
def select_region(start, end)
```

start :	the new start position of the selection
end :	the new end position of the selection

The `select_region`() method selects a region of text from *start* up to, but not including *end*. If *end* is negative, then the selection will run from *start* to the end of the text.

gtk.Editable.get_selection_bounds

```
def get_selection_bounds()
```

Returns :	a tuple containing the start and end positions of the selection or an empty tuple if there is no selection

The `get_selection_bounds`() method returns a tuple that contains the start and end positions of the selection if any or an empty tuple if there is no selection.

gtk.Editable.insert_text

```
def insert_text(text, position=0)
```

text :	the text to be inserted
position :	the position where the text should be inserted

The `insert_text`() method inserts the string specified by *text* at the location specified by *position*.

gtk.Editable.delete_text

```
def delete_text(start_pos, end_pos)
```

start_pos :	the start position of the text to delete
end_pos :	the end position of the text to delete

The `delete_text`() method deletes a sequence of characters starting from *start_pos* up to, but not including *end_pos*. If *end_pos* is negative, then the characters deleted will be those characters from *start_pos* to the end of the text.

gtk.Editable.get_chars

```
def get_chars(start_pos, end_pos)
```

start_pos :	the start position

`end_pos` :	the end position
Returns :	a string containing the characters from start to end

The `get_chars`() method retrieves a string of characters starting from *`start_pos`* up to, but not including *`end_pos`*. If *`end_pos`* is negative, then all the characters from *`start_pos`* to the end of the text are retrieved.

gtk.Editable.cut_clipboard

```
    def cut_clipboard()
```

The `cut_clipboard`() method copies the characters in the current selection to the clipboard and then deletes them from the widget.

gtk.Editable.copy_clipboard

```
    def copy_clipboard()
```

The `copy_clipboard`() method copies the characters in the current selection to the clipboard

gtk.Editable.paste_clipboard

```
    def paste_clipboard()
```

The `paste_clipboard`() method copies the contents of the clipboard to the widget at the cursor position.

gtk.Editable.delete_selection

```
    def delete_selection()
```

The `delete_selection`() method deletes the characters in the selection and releases the selection ownership

gtk.Editable.set_position

```
    def set_position(position)
```

`position` :	the new cursor position

The `set_position`() method sets the cursor position to be just before the character at the location specified by *`position`*. If *`position`* is less than 0 or greater than the number of characters in the widget the cursor is positioned after the last character in the widget. Note *`position`* is in characters not bytes.

gtk.Editable.get_position

```
    def get_position()
```

Returns :	the cursor position

The `get_position`() method retrieves the cursor position as a character index starting from 0. If the cursor is after the last character the position will equal the number of characters in the widget. Note *`position`* is in characters not bytes.

gtk.Editable.set_editable

```
    def set_editable(is_editable)
```

`is_editable` :	if TRUE the text can be edited

The `set_editable()` method sets the widget "editable" attribute of the widget to the value specified by *`is_editable`*. If *`is_editable`* is `TRUE` the text can be edited; if `FALSE`, the text cannot be edited.

gtk.Editable.get_editable

```
    def get_editable()
```

Returns :	`TRUE` if the text is editable.

The `get_editable()` method retrieves the value of the widget "editable" attribute that specifies whether the text is editable. See `set_editable()`.

Signals

The "changed" gtk.Editable Signal

```
    def callback(editable, user_param1, ...)
```

`editable` :	the editable that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "changed" signal is emitted when the contents of the widget have changed.

The "delete−text" gtk.Editable Signal

```
    def callback(editable, start, end, user_param1, ...)
```

`editable` :	the editable that received the signal
`start` :	the start position
`end` :	the end position
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "delete−text" signal is emitted when text is deleted from the widget by the user. The default handler for this signal will normally be responsible for deleting the text, so by connecting to this signal and then stopping the signal with the `gobject.stop_emission()` method, it is possible to prevent it from being deleted. The *`start`* and *`end`* parameters are interpreted as for `delete_text()`

The "insert−text" gtk.Editable Signal

```
    def callback(editable, new_text, new_text_length, position, user_param1, ...)
```

`editable` :	the editable that received the signal
`new_text` :	the string that is being inserted
`new_text_length` :	the length of the new text
`position` :	a pointer to the location at which the new text will be inserted
`user_param1` :	the first user parameter (if any) specified with the `connect()` method

`...` :	additional user parameters (if any)

The "insert−text" signal is emitted when text is inserted into the widget by the user. The default handler for this signal will normally be responsible for inserting the text, so by connecting to this signal and then stopping the signal with the `gobject.stop_emission()` method, it is possible to prevent it from being inserted entirely. The *`position`* parameter is a `gobject.GPointer` object containing a pointer to the insertion position − there is no way to access the position value from `PyGTK`.

gtk.Entry

gtk.Entry — a single line text entry field.

Synopsis

```
class gtk.Entry(gtk.Widget, gtk.Editable, gtk.CellEditable):
    gtk.Entry(max=0)
    def set_visibility(visible)
    def get_visibility()
    def set_invisible_char(ch)
    def get_invisible_char()
    def set_has_frame(setting)
    def get_has_frame()
    def set_max_length(max)
    def get_max_length()
    def set_activates_default(setting)
    def get_activates_default()
    def set_width_chars(n_chars)
    def get_width_chars()
    def set_text(text)
    def get_text()
    def get_layout()
    def get_layout_offsets()
    def set_alignment(xalign)
    def get_alignment()
    def set_completion(width_chars)
    def get_completion()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Entry (implements gtk.Editable, gtk.CellEditable)
```

Properties

"cursor−position"	Read	The current position of the insertion cursor as a character index.
"selection−bound"	Read	The position of the opposite end of the selection from the cursor as a character index.
"editable"	Read/Write	If `TRUE` the entry contents can be edited
"max−length"	Read/Write	The maximum number of characters for this entry. Zero if no maximum.
"visibility"	Read/Write	If `FALSE` displays the "invisible char" instead of the actual text (e.g. password mode)
"has−frame"	Read/Write	If `FALSE` removes outside bevel from entry.
"invisible−char"	Read/Write	The character to use when masking entry contents (when "visibility" is `FALSE`)
"activates−default"	Read/Write	If `TRUE` activate the default widget (such as the default button in a dialog) when Enter is pressed.
"width−chars"	Read/Write	The number of characters to leave space for in the entry.
"scroll−offset"	Read	The number of pixels of the entry scrolled off the screen to the left.
"text"	Read/Write	The contents of the entry.
"xalign"	Read/Write	The horizontal alignment ranging from 0.0 to 1.0 representing the fraction of freespace to the left (right for RTL layouts) of the text. Available in GTK+ 2.4 and above.

Signal Prototypes

"activate"	`def callback(entry, user_param1, ...)`
"copy−clipboard"	`def callback(entry, user_param1, ...)`
"cut−clipboard"	`def callback(entry, user_param1, ...)`
"delete−from−cursor"	`def callback(entry, delete_type, count, user_param1, ...)`
"insert−at−cursor"	`def callback(entry, string, user_param1, ...)`
"move−cursor"	`def callback(entry, step_size, count, extend_selection, user_param1, ...)`
"paste−clipboard"	`def callback(entry, user_param1, ...)`
"populate−popup"	`def callback(entry, menu, user_param1, ...)`
"toggle−overwrite"	`def callback(entry, user_param1, ...)`

Description

The `gtk.Entry` widget is a single line text entry widget. A fairly large set of key bindings are supported by default. If the entered text is longer than the allocation of the widget, the widget will scroll so that the cursor position is visible.

Constructor

```
gtk.Entry(max=0)
```

max :	the maximum length of the entry, or 0 for no maximum. (other than the maximum length of entries.) The value passed in will be limited to the range 0−65536.
Returns :	a new `gtk.Entry`.

Creates a new `gtk.Entry` widget with the maximum length specified by *max*.

Methods

gtk.Entry.set_visibility

```
def set_visibility(visible)
```

`visible` :	If FALSE the contents are obscured using the "invisible−char"

The `set_visibility()` method sets the "visibility" property to the value of *`visible`*. If visible is `TRUE` the contents of the entry are displayed as plain text. If visible is `FALSE`, the contents are obscured by replacing the characters with the invisible character (specified by the "invisible−char" property). The characters will also be obscured when the text in the entry widget is copied elsewhere. The default invisible char is the asterisk '*', but it can be changed with `set_invisible_char()`.

gtk.Entry.get_visibility

```
def get_visibility()
```

Returns :	`TRUE` if the text is currently visible

The `get_visibility()` method retrieves the value of the "visibility" property that determines whether the text in *`entry`* is visible. See `set_visibility()`.

gtk.Entry.set_invisible_char

```
def set_invisible_char(ch)
```

`ch` :	a Unicode character

The `set_invisible_char()` method sets the "invisible−char" property with the value of *`ch`*. The "invisible−char" is the character to use in place of the actual text when `set_visibility()` has been called to set text visibility to `FALSE`. i.e. this is the character used in "password mode" to show the user how many characters have been typed. The default invisible character is an asterisk ('*'). If you set the invisible character to 0, then the user will get no feedback at all; there will be no text on the screen as they type.

gtk.Entry.get_invisible_char

```
def get_invisible_char()
```

Returns :	the current invisible char, or 0, if the entry does not show invisible text at all.

The get_invisible_char() method retrieves the value of the "invisible−char" property. The "invisible−char" is the character displayed in place of the real characters for entries with visibility set to `FALSE`. See `set_invisible_char()`.

gtk.Entry.set_has_frame

```
def set_has_frame(setting)
```

`setting` :	If TRUE the entry is displayed with a beveled frame around it.

The `set_has_frame()` method sets the "has−frame" property to the value of *`setting`*. If *`setting`* is `TRUE` the entry is displayed with a beveled frame around it.

gtk.Entry.get_has_frame

```
def get_has_frame()
```

Returns :	whether the entry has a beveled frame

The get_has_frame() method gets the value of the "has−frame" property. If "has−frame" is TRUE the entry will be displayed with a beveled frame around it.

gtk.Entry.set_max_length

```
def set_max_length(max)
```

`max` :	the maximum length of the entry, or 0 for no maximum. (other than the maximum length of entries.) The value passed in will be clamped to the range 0−65536.

The `set_max_length`() method sets the "max−length" property to the value of *`max`*. The "max−length" property sets the maximum allowed length of the contents of the widget. If the current contents are longer than the given length, then they will be truncated to fit. If max is 0 then there is no maximum length (other than 65536).

gtk.Entry.get_max_length

```
def get_max_length()
```

Returns :	the maximum allowed number of characters in `gtk.Entry`, or 0 if there is no maximum.

The get_max_length() method retrieves the value of the "max−length" property that specifies the maximum allowed length of the text in the entry. See `set_max_length()`.

gtk.Entry.set_activates_default

```
def set_activates_default(setting)
```

`setting` :	If `TRUE` activate the window's default widget on an **Enter** key press

The `set_activates_default`() method sets the "activates−default" property to the value of *`setting`*. If *`setting`* is `TRUE`, pressing the **Enter** key in the entry will activate the default widget for the window containing the entry. This usually means that the dialog box containing the entry will be closed, since the default widget is usually one of the dialog buttons.

(For experts: if *`setting`* is `TRUE`, the entry calls `gtk.Window.activate_default()` on the window containing the entry, in the default handler for the "activate" signal.)

gtk.Entry.get_activates_default

```
def get_activates_default()
```

Returns :	`TRUE` if the entry will activate the default widget

The `get_activates_default`() method retrieves the value of the "activates−default" property which is set by `set_activates_default()`. If "activates−default" is TRUE pressing the **Enter** key in the entry will activate the default widget for the window containing the entry.

gtk.Entry.set_width_chars

```
    def set_width_chars(n_chars)
```

n_chars :	width in chars

The `set_width_chars()` method sets the "width−chars" property to the value of *`n_char`*. Setting the "width−chars" property changes the size request of the entry to be about the right size for *`n_chars`* characters. Note that it only changes the size *request*, the size can still be affected by how you pack the widget into containers. If *`n_chars`* is −1, the size reverts to the default entry size.

gtk.Entry.get_width_chars

```
    def get_width_chars()
```

Returns :	number of chars to request space for, or negative if unset

The `get_width_chars()` method gets the value of the "width−chars" property which is set by the `set_width_chars()` method.

gtk.Entry.set_text

```
    def set_text(text)
```

text :	a string to use as the new contents of the entry

The `set_text()` method sets the "text" property to the value of *`text`*. The string in *`text`* replaces the current contents of the entry.

gtk.Entry.get_text

```
    def get_text()
```

Returns :	the contents of the entry as a string

The `get_text()` method returns the value of the "text" property which is a string containing the contents of the entry.

gtk.Entry.get_layout

```
    def get_layout()
```

Returns :	the `pango.Layout` for this entry

The get_layout() method gets the `pango.Layout` used to display the entry. The layout is useful to e.g. convert text positions to pixel positions, in combination with `get_layout_offsets()`.

gtk.Entry.get_layout_offsets

```
    def get_layout_offsets()
```

Returns :	a tuple containing the X and Y offsets of the pango layout

The `get_layout_offsets()` method obtains the position of the `pango.Layout` used to render text in the entry, in widget coordinates and returns it as a tuple. This method is used to line up the text in an entry with some other text, e.g. when using the entry to implement editable cells in a sheet widget. It is also useful to convert mouse events into coordinates inside the `pango.Layout`, e.g. to take some action if some part of the entry text is clicked.

Note that as the user scrolls around in the entry the offsets will change; you'll need to connect to the "notify::scroll_offset" signal to track this.

gtk.Entry.set_alignment

```
def set_alignment(xalign)
```

`xalign` :	The horizontal alignment ranging from 0.0 to 1.0 representing the freespace to the left (right for RTL layouts) of the text.

Note

This method is available in PyGTK 2.4 and above.

The `set_alignment()` method sets the "xalign" property to the value of *`xalign`*. The alignment controls the horizontal positioning of the contents when the displayed text is shorter than the width of the entry. The value of *`xalign`* is the fraction of freespace to the left (right in RTL layouts) of the text.

gtk.Entry.get_alignment

```
def get_alignment()
```

Returns :	The horizontal alignment ranging from 0.0 to 1.0 representing the freespace to the left (right for RTL layouts) of the text.

Note

This method is available in PyGTK 2.4 and above.

The `get_alignment()` method returns the value of the "xalign" property which is the fraction of freespace (if any) to the left (right in RTL layouts) of the text.

gtk.Entry.set_completion

```
def set_completion(completion)
```

`completion` :	a `gtk.EntryCompletion`

Note

This method is available in PyGTK 2.4 and above.

The `set_completion()` method sets the `gtk.EntryCompletion` specified by *`completion`* to be the auxiliary completion object to use with the entry. All further configuration of the completion mechanism is done using *`completion`* and the `gtk.EntryCompletion` methods.

gtk.Entry.get_completion

```
def get_completion()
```

Returns :	the auxiliary completion object

Note

This method is available in PyGTK 2.4 and above.

The `get_completion()` method returns the `gtk.EntryCompletion` object currently in use by the entry.

Signals

The "activate" gtk.Entry Signal

```
def callback(entry, user_param1, ...)
```

entry :	the entry that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "activate" signal is emitted when the entry is activated either by user action (pressing the **Enter** key) or programmatically with the `gtk.Widget.activate()` method

The "copy−clipboard" gtk.Entry Signal

```
def callback(entry, user_param1, ...)
```

entry :	the entry that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "copy−clipboard" signal is emitted when the selection text in the entry is copied to the clipboard.

The "cut−clipboard" gtk.Entry Signal

```
def callback(entry, user_param1, ...)
```

entry :	the entry that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "cut−clipboard" signal is emitted when the selection in the entry is cut (removed from the entry) and placed in the clipboard.

The "delete−from−cursor" gtk.Entry Signal

```
def callback(entry, delete_type, count, user_param1, ...)
```

entry :	the entry that received the signal
delete_type :	the type of deletion
count :	the number of deletions of the type to perform
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "delete−from−cursor" signal is emitted when the a deletion from the cursor i.e. removal o text at the cursor position, either before it (negative *`count`*) or after it (positive *`count`*). The value of *`delete_type`* can be one of:

- gtk.DELETE_CHARS
- gtk.DELETE_WORD_ENDS
- gtk.DELETE_WORDS
- gtk.DELETE_DISPLAY_LINES
- gtk.DELETE_DISPLAY_LINE_ENDS
- gtk.DELETE_PARAGRAPH_ENDS
- gtk.DELETE_PARAGRAPHS
- gtk.DELETE_WHITESPACE

The "insert−at−cursor" gtk.Entry Signal

```
def callback(entry, string, user_param1, ...)
```

entry :	the entry that received the signal
string :	the text being inserted in the entry
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "insert−at−cursor" signal is emitted when text is being inserted in the entry.

The "move−cursor" gtk.Entry Signal

```
def callback(entry, step, count, extend_selection, user_param1, ...)
```

entry :	the entry that received the signal
step :	the size of the step to move the cursor
count :	the number of steps to move the cursor
extend_selection :	if TRUE extend the selection as well as moving the cursor
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "move−cursor" signal is emitted when the cursor is being moved by *count* steps of size *step*. If *extend_selection* is `TRUE` the selection is extended as the cursor is being moved. The value of *step* can be one of:

- gtk.MOVEMENT_LOGICAL_POSITIONS
- gtk.MOVEMENT_VISUAL_POSITIONS
- gtk.MOVEMENT_WORDS
- gtk.MOVEMENT_DISPLAY_LINES
- gtk.MOVEMENT_DISPLAY_LINE_ENDS
- gtk.MOVEMENT_PARAGRAPH_ENDS
- gtk.MOVEMENT_PARAGRAPHS
- gtk.MOVEMENT_PAGES
- gtk.MOVEMENT_BUFFER_ENDS

The "paste−clipboard" gtk.Entry Signal

```
def callback(entry, user_param1, ...)
```

entry :	the entry that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method

... :	additional user parameters (if any)

The "paste−clipboard" signal is emitted when the contents of the clipboard are pasted into the entry.

The "populate−popup" gtk.Entry Signal

```
    def callback(entry, menu, user_param1, ...)
```

entry :	the entry that received the signal
menu :	the menu that needs populating
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "populate−popup" signal is emitted when the *menu* needs populating.

The "toggle−overwrite" gtk.Entry Signal

```
    def callback(entry, user_param1, ...)
```

entry :	the entry that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "toggle−overwrite" signal is emitted when the internal entry attribute "overwrite_mode" is toggled by pressing the **Insert** key.

Prev	Up	Next
gtk.Editable	Home	gtk.EntryCompletion

gtk.EntryCompletion

Prev **The gtk Class Reference** Next

gtk.EntryCompletion

gtk.EntryCompletion completion functionality for `gtk.Entry` (new in PyGTK 2.4)

Synopsis

```
class gtk.EntryCompletion(gobject.GObject, gtk.CellLayout):
    gtk.EntryCompletion()
    def get_entry()
    def set_model(model=None)
    def get_model()
    def set_match_func(func, func_data)
    def set_minimum_key_length(length)
    def get_minimum_key_length()
    def complete()
    def insert_action_text(index, text)
    def insert_action_markup(index, markup)
    def delete_action(index)
    def insert_prefix()
    def set_text_column(column)
    def set_inline_completion(inline_completion)
    def get_inline_completion()
    def set_popup_completion(popup_completion)
    def get_popup_completion()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.EntryCompletion (implements gtk.CellLayout)
```

Properties

"inline–completion"	Read–Write	If `TRUE` the common prefix should be inserted automatically. Default value: `FALSE`. Available in GTK+ 2.6 and above.
"minimum–key–length"	Read–Write	Minimum length of the search key in order to look up matches.
"model"	Read–Write	The `gtk.TreeModel` to find matches in.
"popup–completion"	Read–Write	If `TRUE` the completions should be shown in a popup window. Default value: `TRUE`. Available in GTK+ 2.6 and above.
"text–column"	Read–Write	The column of the model containing the strings. Allowed values: >= –1. Default value: –1. Available in GTK+ 2.6 and above.

Signal Prototypes

"action–activated"	`def callback(completion, index, user_param1, ...)`
"insert–prefix"	`def callback(completion, prefix, user_param1, ...)`
"match–selected"	`def callback(completion, model, iter, user_param1, ...)`

Description

Note

This widget is available in PyGTK 2.4 and above.

`gtk.EntryCompletion` is an auxiliary object to be used in conjunction with `gtk.Entry` to provide completion functionality. It implements the `gtk.CellLayout` interface, to allow the user to add extra cells to the popup display of completions.

To add completion functionality to an entry, use the `gtk.Entry.set_completion()` method. In addition to regular completion matches, that will be inserted into the entry when they are selected, `gtk.EntryCompletion` also allows "actions" to be displayed in the popup window below any completions. Their appearance is similar to menuitems, to differentiate them clearly from completion strings. When an action is selected, the "action–activated" signal is emitted.

A `gtk.TreeModel` (e.g. a `gtk.ListStore`) containing the completion strings is associated with the `gtk.EntryCompletion` using the `set_model()` method. The tree model column containing the completion strings can be set using the convenience method `set_text_column()` that also creates a `gtk.CellRendererText` and packs it into the entry completion.

Otherwise, you can create `gtk.CellRenderer` objects and pack them into the `gtk.EntryCompletion` using the `gtk.CellLayout` methods `gtk.CellLayout.pack_start()` or `gtk.CellLayout.pack_start()`. However, you will also have to define a match function and set it with the `set_match_func()` method.

If you wanted to create a completion list with the strings to insert and some additional info e.g. an icon or description you could do something like:

```
entry = gtk.Entry()
completion = gtk.EntryCompletion()
entry.set_completion(completion)
liststore = gtk.ListStore(gobject.TYPE_STRING, gtk.gdk.Pixbuf)
completion.set_model(liststore)
pixbufcell = gtk.CellRendererPixbuf()
completion.pack_start(pixbufcell)
completion.add_attribute(pixbufcell, 'pixbuf', 1)
# create a gtk.CellRendererText and pack it in the completion. Also set the
# 'text' attribute
completion.set_text_column(0)
# load up the liststore with string - pixbuf data - assuming pixbuf created
liststore.append(['string text', pixbuf])
```

This will create an entry that will display a pixbuf and the text string during completion.

Actions are easily managed using the `insert_action_text()`, `insert_action_markup()` and `delete_action()` methods.

Constructor

```
gtk.EntryCompletion()
```

Returns :	A newly created `gtk.EntryCompletion` object.

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new `gtk.EntryCompletion` object.

Methods

gtk.EntryCompletion.get_entry

```
def get_entry()
```

Returns :	The `gtk.Entry` that the completion is attached to.

Note

This method is available in PyGTK 2.4 and above.

The `get_entry()` method retrieves the `gtk.Entry` that the entry completion is attached to.

gtk.EntryCompletion.set_model

```
def set_model(model=None)
```

`model` :	The `gtk.TreeModel` to use with the entry completion.

Note

This method is available in PyGTK 2.4 and above.

The `set_model`() method sets the `gtk.TreeModel` specified by *`model`* to be used with the entry completion. A previously set model will be removed before the new model is set. If *`model`* is `None` or not specified, the old model will be unset.

Note

In PyGTK 2.4.0 the model could not be `None` and did not default to `None`.

gtk.EntryCompletion.get_model

```
def get_model()
```

Returns :	The current `gtk.TreeModel`, or `None` if not set.

Note

This method is available in PyGTK 2.4 and above.

The `get_model`() method returns the `gtk.TreeModel` that the entry completion is using as data source. Returns `None` if the model is unset.

gtk.EntryCompletion.set_match_func

```
def set_match_func(func, func_data)
```

`func` :	A function to be used.
`func_data` :	The user data for *`func`*.

Note

This method is available in PyGTK 2.4 and above.

The `set_match_func`() method sets the match function specified by *`func`*. The match function is used by the entry completion to determine if a row of the associated tree model should be in the completion list.

The signature of the match function is:

```
def match_func(completion, key_string, iter, func_data):
```

where *`completion`* is the `gtk.EntryCompletion` that the match function is invoked on, *`key_string`* is the current contents of the `gtk.Entry` to be matched, *`iter`* is a `gtk.TreeIter` pointing at a row in the `gtk.TreeModel` associated with completion and *`func_data`* is the data specified when the `set_match_func()` method was called. The match function should return `TRUE` if the completion string should be displayed; otherwise, `FALSE`.

A simple example match function is:

```
# Assumes that the func_data is set to the number of the text column in the
# model.
def match_func(completion, key, iter, column):
  model = completion.get_model()
  text = model.get_value(iter, column)
```

```
    if text.startswith(key):
      return True
    return False
```

You must use the set_match_func() method to display completions if you don't use the set_text_column() method.

gtk.EntryCompletion.set_minimum_key_length

```
def set_minimum_key_length(length)
```

length :	The minimum length of the key string in order to start completing.

Note

This method is available in PyGTK 2.4 and above.

The `set_minimum_key_length()` method sets the minimum length of the search key to the value specified by *length*. This means that the key string (contents of the gtk.Entry) must be at least *length* characters before a completion list will be displayed. This is useful for long lists, where completing using a small key will take too much time and will likely return too large a dataset.

gtk.EntryCompletion.get_minimum_key_length

```
def get_minimum_key_length()
```

Returns :	The currently used minimum key length.

Note

This method is available in PyGTK 2.4 and above.

The `get_minimum_key_length()` method returns the minimum key length set for the entry completion. See the set_minimum_key_length() method for more information.

gtk.EntryCompletion.complete

```
def complete()
```

Note

This method is available in PyGTK 2.4 and above.

The `complete()` method requests a completion operation, i.e. a refiltering of the current list with completions, using the current key. The completion list view will be updated accordingly.

gtk.EntryCompletion.insert_action_text

```
def insert_action_text(index, text)
```

index :	The index in the action list where the item should be inserted.
text :	The text of the item to insert.

Note

This method is available in PyGTK 2.4 and above.

The `insert_action_text()` method inserts an action in the action item list of the entry completion at the position specified by *`index`* with the text specified by *`text`*. If you want the action item to have markup, use the `gtk.EntryCompletion.insert_action_markup()` method.

gtk.EntryCompletion.insert_action_markup

```
  def insert_action_markup(index, markup)
```

`index` :	The index in the action list where the item should be inserted.
`markup` :	The Pango markup of the item to insert.

Note

This method is available in PyGTK 2.4 and above.

The `insert_action_markup()` method inserts an action item in the action item list of the entry completion at the position specified by *`index`* with the Pango markup specified by *`markup`*.

gtk.EntryCompletion.delete_action

```
  def delete_action(index)
```

`index` :	The index of the item to delete.

Note

This method is available in PyGTK 2.4 and above.

The `delete_action()` method deletes the action item at the position in the action item list specified by *`index`*.

gtk.EntryCompletion.insert_prefix

```
  def insert_prefix()
```

Note

This method is available in PyGTK 2.6 and above.

The `insert_prefix()` method requests a prefix insertion.

gtk.EntryCompletion.set_text_column

```
  def set_text_column(column)
```

`column` :	The column in the model to get strings from.

Note

This method is available in PyGTK 2.4 and above.

The `set_text_column()` method is a convenience method for setting up the most common completion scenario: a completion list with just strings. This method creates and adds a `gtk.CellRendererText` using the column specified by *`column`* as the source for completion strings. If you don't use this method you will have to install a `gtk.CellRendererText` in the entry completion and set a match function using the `set_match_func()` method to display the completion strings. In GTK+ 2.6 the "text–column" property is set to the value of *`column`*.

gtk.EntryCompletion.set_inline_completion

```
def set_inline_completion(inline_completion)
```

`inline_completion` :	if `TRUE` do inline completion

Note

This method is available in PyGTK 2.6 and above.

The `set_inline_completion()` method sets the "inline–completion" property to the value of *`inline_completion`*. If *`inline_completion`* is `TRUE`, the common prefix of the possible completions should be automatically inserted in the entry.

gtk.EntryCompletion.get_inline_completion

```
def get_inline_completion()
```

Returns :	`TRUE` if automatic inline completion is enabled.

Note

This method is available in PyGTK 2.6 and above.

The `get_inline_completion()` method returns the value of the "inline–completion" property. If the value of the "inline–completion" property is `TRUE` the common prefix of possible completions is automatically inserted in the entry.

gtk.EntryCompletion.set_popup_completion

```
def set_popup_completion(popup_completion)
```

`popup_completion` :	If `TRUE` do popup completion.

Note

This method is available in PyGTK 2.6 and above.

The `set_popup_completion()` method sets the "popup–completion" property to the value of *`popup_completion`*. If *`popup_completion`* is `TRUE` the completions should be presented in a popup window.

gtk.EntryCompletion.get_popup_completion

```
    def get_popup_completion()
```

Returns :	`TRUE` if completions should be displayed in a popup.

Note

This method is available in PyGTK 2.6 and above.

The `get_popup_completion`() method returns the value of the "popup–completion" property. If the value of "popup–completion" property is `TRUE` the completions should be presented in a popup window.

Signals

The "action–activated" gtk.EntryCompletion Signal

```
    def callback(completion, index, user_param1, ...)
```

`completion` :	the entry completion that received the signal
`index` :	the index of the action item that was activated.
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.4 and above.

The "action–activated" signal is emitted when an action item is selected from the popup action list.

The "insert–prefix" gtk.EntryCompletion Signal

```
    def callback(completion, prefix, user_param1, ...)
```

`completion` :	the entry completion that received the signal
`prefix` :	the common prefix of all possible completions
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.6 and above.

The "insert–prefix" signal is emitted when the inline auto–completion is triggered. The default behavior is to make the entry display the whole prefix and select the newly inserted part.

Applications may connect to this signal in order to insert only a smaller part of the prefix into the entry – e.g. the entry used in the `gtk.FileChooser` inserts only the part of the prefix up to the next '/'.

The "match–selected" gtk.EntryCompletion Signal

```
    def callback(completion, model, iter, user_param1, ...)
```

completion :	the entry completion that received the signal
model :	the `gtk.TreeModel` that *iter* points into.
iter :	a `gtk.TreeIter` pointing at the selection completion string row in *model*.
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.4 and above.

The "match−selected" signal is emitted when a completion string was selected from the completion list. *iter* points at the row in *model* that contains the completion string.

Prev Up Next
gtk.Entry Home gtk.EventBox

gtk.EventBox
Prev The gtk Class Reference Next

gtk.EventBox

gtk.EventBox a widget used to catch events for widgets which do not have their own window.

Synopsis

```
class gtk.EventBox(gtk.Bin):
    gtk.EventBox()
    def get_visible_window()
    def set_visible_window(visible_window)
    def get_above_child()
    def set_above_child(above_child)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.EventBox
```

Properties

"above−child"	Read−Write	If `TRUE`, the event−trapping window of the eventbox is above the window of the child widget as opposed to below it. Available in GTK+ 2.4 and above.
"visible−window"	Read−Write	If `TRUE`, the event box is visible, as opposed to invisible and only used to trap events. Available in GTK+ 2.4 and above.

Description

The `gtk.EventBox` widget is an invisible container widget that provides a window for widgets that do not have their own window. In GTK, widgets must have a window to be able to receive event signals. Those widgets that are "windowless" can use a `gtk.EventBox` to receive event signals.

Constructor

```
gtk.EventBox()
```

Returns :	an eventbox widget

Creates a new `gtk.EventBox` widget.

Methods

gtk.EventBox.get_visible_window

```
def get_visible_window()
```

Returns :	`TRUE` if the event box window is visible.

Note

This method is available in PyGTK 2.4 and above.

The `gtk_visible_window`() method returns `TRUE` if the event box has a visible window. See the `set_visible_window()` method for details.

gtk.EventBox.set_visible_window

```
def set_visible_window(visible_window)
```

`visible_window` :	if `TRUE` the event box window is visible.

Note

This method is available in PyGTK 2.4 and above.

The `set_visible_window`() method sets whether the event box uses a visible or invisible child window according to the value specified by *`visible_window`*. If *`visible_window`* is `TRUE` the event box uses a visible child window; otherwise, an invisible child window. The default is to use visible windows.

In an invisible window event box, the window that that the event box creates is a `gtk.gdk.INPUT_ONLY` window, that is invisible and only serves to receive events. A visible window event box creates a visible (`gtk.gdk.INPUT_OUTPUT`) window that acts as the parent window for all the widgets contained in the event box.

You should generally make your event box invisible if you just want to trap events. Creating a visible window may cause artifacts that are visible to the user, especially if the user is using a theme with gradients or pixmaps. The main reason to create a non input–only event box is if you want to set the background to a different color or draw on it.

Note

There is one unexpected issue for an invisible event box that has its window below the child. (See the `set_above_child()` method.) Since the input–only window is not an ancestor window of any windows that descendant widgets of the event box create, events on these windows aren't propagated up by the windowing system, but only by GTK+. The practical effect of this is if an event isn't in the event mask for the descendant window (see the `gtk.Widget.add_events()` method), it won't be received by the event box.

This problem doesn't occur for visible event boxes, because the event box window is actually the ancestor of the descendant windows, not just at the same place on the screen.

gtk.EventBox.get_above_child

```
    def get_above_child()
```

Returns :	`TRUE` if the event box window is above the window of its child.

Note

This method is available in PyGTK 2.4 and above.

The `get_above_child`() method returns the value of the "above−child" property that indicates whether the event box window is above or below the windows of its child. See the `set_above_child()` method for details.

gtk.EventBox.set_above_child

```
    def set_above_child(above_child)
```

`above_child` :	if `TRUE` the event box window is above the windows of its child

Note

This method is available in PyGTK 2.4 and above.

The `set_above_child`() method sets the "above−child" property to the value of *`above_child`*. If *`above_child`* is `TRUE`, the event box window is positioned above the windows of its child; otherwise, below it. If the window is above, all events inside the event box will go to the event box. If the window is below, events in windows of child widgets will first got to that widget, and then to its parents. The default is to keep the window below the child.

Prev	Up	Next
gtk.EntryCompletion	Home	gtk.Expander

gtk.Expander

Prev **The gtk Class Reference** Next

gtk.Expander

gtk.Expander — a container that can hide its child (new in PyGTK 2.4)

Synopsis

```
class gtk.Expander(gtk.Bin):
    gtk.Expander(label=None)
    def set_expanded(expanded)
    def get_expanded()
    def set_spacing(spacing)
    def get_spacing()
    def set_label(label)
    def get_label()
    def set_use_underline(use_underline)
    def get_use_underline()
    def set_use_markup(use_markup)
    def get_use_markup()
    def set_label_widget(label_widget)
    def get_label_widget()
```

```
Functions

    def gtk.expander_new_with_mnemonic(label=None)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Expander
```

Properties

"expanded"	Read–Write	If `TRUE`, the expander has been opened to reveal the child widget.
"label"	Read–Write	The text of the expander's label.
"label–widget"	Read–Write	A widget to display in place of the usual expander label.
"spacing"	Read–Write	The space to put between the label and the child.
"use–markup"	Read–Write	If `TRUE`, the text of the label includes Pango markup. See the `pango.parse_markup()` function.
"use–underline"	Read–Write	If `TRUE`, n underline in the text indicates the next character should be used for the mnemonic accelerator key.

Style Properties

"expander–size"	Read	The size of the expander arrow.
"expander–spacing"	Read	The spacing around expander arrow.

Signal Prototypes

"activate"	`def callback(`*`expander, user_param1, ...`*`)`

Description

Note

This widget is available in PyGTK 2.4 and above.

A `gtk.Expander` allows the user to hide or show its child by clicking on an expander triangle similar to the triangles used in a `gtk.TreeView`.

Normally you use an expander as you would use any other descendant of `gtk.Bin`; you create the child widget and use `gtk.Container.add()` to add it to the expander. When the expander is toggled, it will take care of showing and hiding the child automatically.

Special Usage

There there are situations in which you may prefer to show and hide the expanded widget yourself, such as when you want to actually create the widget at expansion time. In this case, create a `gtk.Expander` but do not add a child to it. The expander widget has the "expanded" property that can be used to monitor its expansion state. You should watch this property with a signal connection as follows:

```
expander = gtk.expander_new_with_mnemonic("_More Options")
expander.connect("notify::expanded", expander_callback)

...

def expander_callback(expander, param_spec, user_data):
  if expander.get_expanded():
    # Show or create widgets
  else:
    # Hide or destroy widgets
```

The "activate" signal can also be used to track the expansion though it occurs before the "expanded" property is changed so the logic of the `expander_callback()` function would have to be reversed.

Constructor

```
gtk.Expander(label=None)
```

`label` :	the text of the label or `None`
Returns :	a new `gtk.Expander` widget.

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new expander using *`label`* as the text of the label. If *`label`* is `None` or not specified, no label will be created.

Methods

gtk.Expander.set_expanded

```
def set_expanded(expanded)
```

`expanded` :	if `TRUE`, the child widget is revealed

Note

This method is available in PyGTK 2.4 and above.

The `set_expanded()` method sets the "expanded" property to the value of *`expanded`*. If *`expanded`* is `TRUE`, the child widget is revealed; if `FALSE`, the child widget is hidden.

gtk.Expander.get_expanded

```
    def get_expanded()
```

Returns :	TRUE if the child is revealed.

Note

This method is available in PyGTK 2.4 and above.

The `get_expanded()` method returns the value of the "expanded" property. If "expanded" is `TRUE` the child widget is revealed.

gtk.Expander.set_spacing

```
    def set_spacing(spacing)
```

`spacing` :	the distance between the expander and child in pixels.

Note

This method is available in PyGTK 2.4 and above.

The `set_spacing()` method sets the "spacing" property to the value of *`spacing`* that sets is the number of pixels to place between expander and the child.

gtk.Expander.get_spacing

```
    def get_spacing()
```

Returns :	the spacing between the expander and child.

Note

This method is available in PyGTK 2.4 and above.

The `get_spacing()` method returns the value of the "spacing" property set by the `set_spacing()` method.

gtk.Expander.set_label

```
    def set_label(label)
```

`label` :	a string to use as the label or `None`

Note

This method is available in PyGTK 2.4 and above.

The `set_label()` method sets the "label" property to the value of *`label`* and sets the text of the label of the expander. Any previously set label will be cleared. If *`label`* is `None` the expander will have no label.

gtk.Expander.get_label

```
    def get_label()
```

Returns :	the text of the label widget.

Note

This method is available in PyGTK 2.4 and above.

The `get_label()` method returns the value of the "label" property that contains the text of the expander label, as set by the `set_label()` method. If the label text has not been set the return value will be `None`.

gtk.Expander.set_use_underline

```
    def set_use_underline(use_underline)
```

`use_underline` :	`TRUE` if underlines in the text indicate mnemonics

Note

This method is available in PyGTK 2.4 and above.

The `set_use_underline()` method sets the "use_underline" property to the value of *`use_underline`*. If *`use_underline`* is `TRUE`, an underline in the text of the expander label indicates the next character should be used for the mnemonic accelerator key.

gtk.Expander.get_use_underline

```
    def get_use_underline()
```

Returns :	`TRUE` if an embedded underline in the expander label indicates the mnemonic accelerator keys.

Note

This method is available in PyGTK 2.4 and above.

The `get_use_underline()` method returns the value of the "use–underline" property. If "use–underline" is `TRUE` an embedded underline in the expander label indicates a mnemonic. See the `set_use_underline()` method.

gtk.Expander.set_use_markup

```
    def set_use_markup(use_markup)
```

`use_markup` :	if `TRUE`, the label's text should be parsed for markup

Note

This method is available in PyGTK 2.4 and above.

The `set_use_markup()` method sets the "use–markup" property to the value of *`use_markup`*. If *`use_markup`* is `TRUE` the text of the label contains markup in the Pango text markup language. See the `gtk.Label.set_markup()` method for more information.

gtk.Expander.get_use_markup

```
    def get_use_markup()
```

Returns :	`TRUE` if the label's text will be parsed for markup

Note

This method is available in PyGTK 2.4 and above.

The `get_use_markup()` method returns the value of the "use−markup" property. If "use−markup" is `TRUE`, the label's text is interpreted as marked up with the Pango text markup language. See the `set_use_markup()` method.

gtk.Expander.set_label_widget

```
    def set_label_widget(label_widget)
```

`label_widget` :	the new label widget or `None`

Note

This method is available in PyGTK 2.4 and above.

The `set_label_widget()` method sets the expander to use the widget specified by *`label_widget`* as the label instead of a `gtk.Label`. This widget appears embedded alongside the expander arrow. If *`label_widget`* is `None`, the expander will have no label.

gtk.Expander.get_label_widget

```
    def get_label_widget()
```

Returns :	the label widget, or `None` if there is none.

Note

This method is available in PyGTK 2.4 and above.

The `get_label_widget()` method retrieves the expander's label widget. See the `set_label_widget()` method.

Functions

gtk.expander_new_with_mnemonic

```
    def gtk.expander_new_with_mnemonic(label=None)
```

`label` :	the text of the label with an underscore in front of the mnemonic character or `None`
Returns :	a new `gtk.Expander` widget.

Note

This function is available in PyGTK 2.4 and above.

The `gtk.expander_new_with_mnemonic()` function creates a new `gtk.Expander` using *`label`* as the text of the label. If characters in *`label`* are preceded by an underscore, they are underlined. If you need a literal underscore character in a label, use '__' (two underscores). The first underlined character represents a keyboard accelerator called a mnemonic. Pressing **Alt** with that key activates the button. If *`label`* is `None` the expander will have no label.

Signals

The "activate" gtk.Expander Signal

```
def callback(expander, user_param1, ...)
```

expander :	the expander that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

Note

This signal is available in PyGTK 2.4 and above.

The "activate" signal is emitted when the expander is activated by the user clicking on the expander toggle.

Prev	Up	Next
gtk.EventBox	Home	gtk.FileChooser

gtk.FileChooser

Prev The gtk Class Reference Next

gtk.FileChooser

gtk.FileChooser an interface for choosing files (new in PyGTK 2.4)

Synopsis

```
class gtk.FileChooser(gobject.GInterface):
    def set_action(action)
    def get_action()
    def set_local_only(local_only)
    def get_local_only()
    def set_select_multiple(select_multiple)
    def get_select_multiple()
    def set_current_name(name)
    def get_filename()
    def set_filename(filename)
    def select_filename(filename)
    def unselect_filename(filename)
    def select_all()
    def unselect_all()
    def get_filenames()
    def set_current_folder(filename)
    def get_current_folder()
    def get_uri()
    def set_uri(uri)
    def select_uri(uri)
    def unselect_uri(uri)
    def get_uris()
    def set_current_folder_uri(uri)
    def get_current_folder_uri()
    def set_preview_widget(preview_widget)
    def get_preview_widget()
    def set_preview_widget_active(active)
    def get_preview_widget_active()
    def set_use_preview_label(use_label)
    def get_use_preview_label()
    def get_preview_filename()
    def get_preview_uri()
    def set_extra_widget(extra_widget)
    def get_extra_widget()
    def add_filter(filter)
    def remove_filter(filter)
    def list_filters()
    def set_filter(filter)
    def get_filter()
    def add_shortcut_folder(folder)
    def remove_shortcut_folder(folder)
    def list_shortcut_folders()
    def add_shortcut_folder_uri(uri)
    def remove_shortcut_folder_uri(uri)
    def list_shortcut_folder_uris()
    def set_show_hidden(show_hidden)
    def get_show_hidden()
```

Properties

"action"	Read–Write	The type of operation that the file selector is performing – one of: `gtk.FILE_CHOOSER_ACTION_OPEN`, `gtk.FILE_CHOOSER_ACTION_SAVE`, `gtk.FILE_CHOOSER_ACTION_SELECT_FOLDER` or `gtk.FILE_CHOOSER_ACTION_CREATE_FOLDER`. Default value: `gtk.FILE_CHOOSER_ACTION_OPEN`
"extra–widget"	Read–Write	An application supplied widget for extra options.
"file–system–backend"	Write–Construct	The name of the file system backend to use. Default value: `None`
"filter"	Read–Write	The current `gtk.FileFilter` for selecting which files are displayed.
"local–only"	Read–Write	If `TRUE`, the selected file(s) should be limited to local file: URLs. Default value: `TRUE`
"preview–widget"	Read–Write	An application supplied widget for custom previews.
"preview–widget–active"	Read–Write	If `TRUE`, the application supplied widget for custom previews should be shown. Default value: `TRUE`
"select–multiple"	Read–Write	If `TRUE`, allow multiple files to be selected except if `gtk.FILE_CHOOSER_ACTION_SAVE` is set as the "action" property. Default value: `FALSE`
"show–hidden"	Read–Write	If `TRUE`, hidden files and folders should be displayed. Default value: `FALSE`
"use–preview–label"	Read–Write	If `TRUE`, display a stock label with the name of the previewed file. Default value: `TRUE`

Signal Prototypes

"current–folder–changed"	`def callback(`*`filechooser, user_param1, ...`*`)`
"file–activated"	`def callback(`*`filechooser, user_param1, ...`*`)`
"selection–changed"	`def callback(`*`filechooser, user_param1, ...`*`)`
"update–preview"	`def callback(`*`filechooser, user_param1, ...`*`)`

Description

Note

This interface is available in PyGTK 2.4 and above.

gtk.FileChooser is an interface that can be implemented by file selection widgets. In PyGTK, the main objects that implement this interface are gtk.FileChooserWidget and gtk.FileChooserDialog. You do not need to write an object that implements the gtk.FileChooser interface unless you are trying to adapt an existing file selector to expose a standard programming interface.

gtk.FileChooser has several elements to its display:

- a list of shortcut folders on the left that is divided into two lists by a horizontal line:
    - a default list of folders usually including "Home" and "Filesystem" with folders added using the add_shortcut_folder() or add_shortcut_folder_uri() methods.
    - a list of user specified shortcut folders managed using the "Add" and "Remove" buttons at the bottom of the file chooser.
- the "Add" and "Remove" buttons that allow a user to add or remove a folder to or from the user's shortcut folder list.
- the current folder path as a series of buttons above the file selection window. The buttons can be clicked to jump to the associated ancestor folder.
- the file selection window that displays the contents of the current folder in alphabetical order with last modified time.

Adding A Preview Widget

You can add a custom preview widget to a file chooser and get notification when the preview needs to be updated. To install a preview widget, use the set_preview_widget() method. Then, connect to the "update−preview" signal to be notified when you need to update the contents of the preview.

Your callback should use the get_preview_filename() method to see what needs previewing. Once you have generated the preview for the corresponding file, you must call the set_preview_widget_active() method with a boolean flag that indicates whether your callback could successfully generate a preview. An example use of a custom preview is:

```
    ...
    preview = gtk.Image()

    my_file_chooser.set_preview_widget(preview)
    my_file_chooser.connect("update-preview", update_preview_cb, preview)
    ...

  def update_preview_cb(file_chooser, preview):
    filename = file_chooser.get_preview_filename()
    try:
      pixbuf = gtk.gdk.pixbuf_new_from_file_at_size(filename, 128, 128)
      preview.set_from_pixbuf(pixbuf)
      have_preview = True
    except:
      have_preview = False
    file_chooser.set_preview_widget_active(have_preview)
    return
  ...
```

Adding Extra Widgets

You can add extra widgets to a file chooser to provide options that are not present in the default design. For example, you can add a `gtk.ToggleButton` to give the user the option to open a file in read−only mode. You can use the `set_extra_widget()` method to insert additional widgets in a file chooser. For example:

```
toggle = gtk.CheckButton("Open file read-only")
toggle.show ()
my_file_chooser.set_extra_widget(toggle)
```

If you want to set more than one extra widget in the file chooser, you can use a container such as a `gtk.VBox` or a `gtk.Table` to hold your widgets; then set the container as the whole extra widget.

Key Bindings

The `gtk.FileChooserDialog` uses the private `GtkFileChooserDefaultClass` that has several key bindings and their associated signals. This section describes the available key binding signals.

The default keys that activate the key−binding signals in GtkFileChooserDefaultClass are as follows:

Signal name	Key
location−popup	Control−L
up−folder	Alt−Up
down−folder	Alt−Down
home−folder	Alt−Home

To change these defaults to something else, you could include the following fragment in your `.gtkrc-2.0` file:

```
binding "my-own-gtkfilechooser-bindings" {
	bind "<Alt><Shift>l" {
		"location-popup" ()
	}
	bind "<Alt><Shift>Up" {
		"up-folder" ()
	}
	bind "<Alt><Shift>Down" {
		"down-folder" ()
	}
	bind "<Alt><Shift>Home" {
		"home-folder-folder" ()
	}
}

class "GtkFileChooserDefault" binding "my-own-gtkfilechooser-bindings"
```

The "GtkFileChooserDefault::location−popup" signal is used to make the file chooser show a "Location" dialog which the user can use to manually type the name of the file he wishes to select. By default this is bound to **Control−L**.

```
def location_popup_cb(filechooser, user_data):
```

where *`filechooser`* is the `gtk.FileChooser` that received the signal, *`user_data`* is user data set when the signal handler was connected.

The "GtkFileChooserDefault::up−folder" signal is used to make the file chooser go to the parent of the current folder in the file hierarchy. By default this is bound to **Alt−Up**.

```
def up_folder_cb(filechooser, user_data):
```

where *filechooser* is the object that received the signal and *user_data* is the user data set when the signal handler was connected.

The "GtkFileChooserDefault::down−folder" signal is used to make the file chooser go to a child of the current folder in the file hierarchy. The subfolder that will be used is displayed in the path bar widget of the file chooser. For example, if the path bar is showing "/foo/bar/baz", then this will cause the file chooser to switch to the "baz" subfolder. By default this is bound to **Alt−Down**.

```
def down_folder_cb(filechooser, user_data):
```

where *filechooser* is the object that received the signal and *user_data* is the user data set when the signal handler was connected.

The "GtkFileChooserDefault::home−folder" signal is used to make the file chooser show the user's home folder in the file list. By default this is bound to **Alt−Home**.

```
def home_folder_cb(filechooser, user_data):
```

where *filechooser* is the object that received the signal and *user_data* is the user data set when the signal handler was connected.

Methods

gtk.FileChooser.set_action

```
def set_action(action)
```

action :	the file selection action – one of: `gtk.FILE_CHOOSER_ACTION_OPEN`, `gtk.FILE_CHOOSER_ACTION_SAVE`, `gtk.FILE_CHOOSER_ACTION_SELECT_FOLDER` or `gtk.FILE_CHOOSER_ACTION_CREATE_FOLDER`.

Note

This method is available in PyGTK 2.4 and above.

The `set_action()` method sets the "action" property to the value of *action*. The type of operation that that the chooser is performing is set by *action* causing the user interface to be changed to suit the selected action. The value of *action* must be one of:

`gtk.FILE_CHOOSER_ACTION_OPEN`	Indicates open mode. The file chooser will only let the user pick an existing file.
`gtk.FILE_CHOOSER_ACTION_SAVE`	Indicates save mode. The file chooser will let the user pick an existing file, or type in a new filename.
`gtk.FILE_CHOOSER_ACTION_SELECT_FOLDER`	Indicates an open mode for selecting folders. The file chooser will let the user pick an existing folder.
`gtk.FILE_CHOOSER_ACTION_CREATE_FOLDER`	Indicates a mode for creating a new folder. The file chooser will let the user name an existing or new folder

For example, an option to create a new folder might be shown if the action is `gtk.FILE_CHOOSER_ACTION_SAVE` but not if the action is `gtk.FILE_CHOOSER_ACTION_OPEN`.

gtk.FileChooser.get_action

```
    def get_action()
```

Returns :	the action that is set for the file selector

Note

This method is available in PyGTK 2.4 and above.

The `get_action()` method returns the value of the "action" property that indicates the type of operation that the file chooser is set to perform. See the `set_action()` method for more information.

gtk.FileChooser.set_local_only

```
    def set_local_only(local_only)
```

`local_only` :	if `TRUE`, only local files can be selected

Note

This method is available in PyGTK 2.4 and above.

The `set_local_only()` method sets the "local–only" property to the value of *`local_only`*. If *`local_only`* is `TRUE` (the default), only local files can be selected in the file selector and the selected files are guaranteed to be accessible through the operating system's native file system. Therefore, the application only needs to use the filename methods in `gtk.FileChooser`. For example, the application can use the `get_filename()` method instead of the URI method `get_uri()`.

gtk.FileChooser.get_local_only

```
    def get_local_only()
```

Returns :	`TRUE` if only local files can be selected.

Note

This method is available in PyGTK 2.4 and above.

The `get_local_only()` method returns the value of the "local–only" property that indicates whether only local files can be selected in the file selector. See the `set_local_only()` method for more information.

gtk.FileChooser.set_select_multiple

```
    def set_select_multiple(select_multiple)
```

`select_multiple` :	if `TRUE` multiple files can be selected.

Note

This method is available in PyGTK 2.4 and above.

The `set_select_multiple()` method sets the "select_multiple" property to the value of *`select_multiple`*. If *`select_multiple`* is `TRUE`, multiple files can be selected in the file selector.

Note

The "select–multiple" property cannot be set `TRUE` when the file chooser action is `gtk.FILE_CHOOSER_ACTION_SAVE` or `gtk.FILE_CHOOSER_ACTION_CREATE_FOLDER`.

gtk.FileChooser.get_select_multiple

```
def get_select_multiple()
```

Returns :	`TRUE` if multiple files can be selected.

Note

This method is available in PyGTK 2.4 and above.

The `get_select_multiple`() method returns the value of the "select_multiple" property that indicates whether multiple files can be selected in the file selector. See the `set_select_multiple()` method for more information.

gtk.FileChooser.set_current_name

```
def set_current_name(name)
```

`name` :	the filename to use, as a UTF–8 string

Note

This method is available in PyGTK 2.4 and above.

The `set_current_name`() method sets the current name in the file selector to the value of *`name`*, as if entered by the user. Note that the name passed in here is a UTF–8 string rather than a filename. This method is meant for such uses as a suggested name in a "Save As..." dialog.

If you want to preselect a particular existing file, you should use the `set_filename()` method instead.

gtk.FileChooser.get_filename

```
def get_filename()
```

Returns :	The currently selected filename, or `None` if no file is selected, or the selected file can't be represented with a local filename.

Note

This method is available in PyGTK 2.4 and above.

The `get_filename`() method returns the filename of the currently selected file in the file selector. If multiple files are selected, one of the filenames will be returned at random. If the file chooser is in folder mode, this function returns the selected folder.

gtk.FileChooser.set_filename

```
def set_filename(filename)
```

`filename` :	the filename to set as current

Returns :	`TRUE` if both the folder could be changed and the file was selected successfully, `FALSE` otherwise.

Note

This method is available in PyGTK 2.4 and above.

The `set_filename`() method sets *`filename`* as the current filename for the the file chooser. If the file name isn't in the current folder of the chooser, then the current folder of the chooser will be changed to the folder containing *`filename`*. This is equivalent to a sequence of `unselect_all()` followed by `select_filename()`.

Note that the file must exist, or nothing will be done except for the directory change. To pre−enter a filename for the user, as in a "Save as ..." dialog, use the `set_current_name()` method.

gtk.FileChooser.select_filename

```
def select_filename(filename)
```

`filename` :	the filename to select
Returns :	`TRUE` if both the folder could be changed and the file was selected successfully, `FALSE` otherwise.

Note

This method is available in PyGTK 2.4 and above.

The `select_filename`() method selects the filename specified by *`filename`*. If *`filename`* isn't in the current folder of the chooser, then the current folder of the chooser will be changed to the folder containing *`filename`*.

gtk.FileChooser.unselect_filename

```
def unselect_filename(filename)
```

`filename` :	the filename to unselect

Note

This method is available in PyGTK 2.4 and above.

The `unselect_filename`() method unselects the currently selected filename specified by *`filename`*. If *`filename`* is not in the current directory, does not exist, or is otherwise not currently selected, this method does nothing.

gtk.FileChooser.select_all

```
def select_all()
```

Note

This method is available in PyGTK 2.4 and above.

The `select_all`() method selects all the files in the current folder of a file chooser.

gtk.FileChooser.unselect_all

```
    def unselect_all()
```

Note

This method is available in PyGTK 2.4 and above.

The `unselect_all`() method unselects all the files in the current folder of a file chooser.

gtk.FileChooser.get_filenames

```
    def get_filenames()
```

Returns :	a list containing the filenames of all selected files and subfolders in the current folder.

Note

This method is available in PyGTK 2.4 and above.

The `get_filenames`() method returns a list containing all the selected files and subfolders in the current folder of the chooser. The returned names are full absolute paths. If files in the current folder cannot be represented as local filenames they will be ignored. (See the `get_uris()` method for more information)

gtk.FileChooser.set_current_folder

```
    def set_current_folder(filename)
```

`filename` :	the full path of the new current folder
Returns :	`TRUE` if the folder could be changed successfully, `FALSE` otherwise.

Note

This method is available in PyGTK 2.4 and above.

The `set_current_folder`() method sets the current folder for the chooser to the local filename specified by *`filename`*. The user will be shown the full contents of the current folder, plus user interface elements for navigating to other folders.

gtk.FileChooser.get_current_folder

```
    def get_current_folder()
```

Returns :	the full path of the current folder, or `None` if the current path cannot be represented as a local filename.

Note

This method is available in PyGTK 2.4 and above.

The `get_current_folder`() method returns the current folder of the chooser as a local filename. See the `set_current_folder()` method for more information.

gtk.FileChooser.get_uri

```
def get_uri()
```

Returns :	The currently selected URI, or `None` if no file is selected.

Note

This method is available in PyGTK 2.4 and above.

The `get_uri`() method returns the URI for the currently selected file in the file selector. If multiple files are selected, one of the filenames will be returned at random. If the file chooser is in folder mode, this function returns the selected folder.

gtk.FileChooser.set_uri

```
def set_uri(uri)
```

`uri` :	the URI to set as the current file
Returns :	`TRUE` if both the folder could be changed and the URI was successfully selected.

Note

This method is available in PyGTK 2.4 and above.

The `set_uri`() method sets the file referred to by *`uri`* as the current file for the file chooser; If the file name isn't in the current folder of the chooser, then the current folder of the chooser will be changed to the folder containing *`uri`*. This is equivalent to the sequence of `unselect_all()` followed by `select_uri()`. Note that the file must exist, or nothing will be done except for the directory change. To pre–enter a filename for the user, as in a "Save As ..." dialog, use the `set_current_name()` method.

gtk.FileChooser.select_uri

```
def select_uri(uri)
```

`uri` :	the URI of the file to select
Returns :	`TRUE` if both the folder could be changed and the URI was successfully selected.

Note

This method is available in PyGTK 2.4 and above.

The `select_uri`() method selects the file referred to by *`uri`*. If the URI doesn't refer to a file in the current folder of the chooser, then the current folder of the chooser will be changed to the folder containing the file referenced by *`uri`*.

gtk.FileChooser.unselect_uri

```
def unselect_uri(uri)
```

`uri` :	the URI of the file to unselect

Note

This method is available in PyGTK 2.4 and above.

The `unselect_uri()` method unselects the file referred to by *`uri`*. If the file is not in the current directory, does not exist, or is otherwise not currently selected, this method does nothing.

gtk.FileChooser.get_uris

```
def get_uris()
```

Returns :	a list containing the URIs of all selected files and subfolders in the current folder.

Note

This method is available in PyGTK 2.4 and above.

The `get_uris()` method returns a list containing all the selected files and subfolders in the current folder of the chooser. The returned names are full absolute URIs.

gtk.FileChooser.set_current_folder_uri

```
def set_current_folder_uri(uri)
```

`uri` :	the URI for the new current folder
Returns :	`TRUE` if the folder could be changed successfully, `FALSE` otherwise.

Note

This method is available in PyGTK 2.4 and above.

The `set_current_folder_uri()` method sets the current folder for the chooser to the folder referenced by *`uri`*. The user will be shown the full contents of the current folder, plus user interface elements for navigating to other folders.

gtk.FileChooser.get_current_folder_uri

```
def get_current_folder_uri()
```

Returns :	the URI for the current folder.

Note

This method is available in PyGTK 2.4 and above.

The `get_current_folder_uri()` method returns the URI reference of the current folder of the chooser. See the `set_current_folder_uri()` method for more information.

gtk.FileChooser.set_preview_widget

```
def set_preview_widget(preview_widget)
```

`preview_widget` :	a widget for displaying a preview.

Note

This method is available in PyGTK 2.4 and above.

The `set_preview_widget()` method sets the "preview−widget" property to the value of *`preview_widget`*. The *`preview_widget`* is used to preview the currently selected file. To implement a custom preview:

- set the preview widget
- connect a callback to the "selection−changed" signal of the file chooser
- in the callback, call the `get_preview_filename()` method or the `get_preview_uri()` method to retrieve the selected file name or URI
- if you can, display a preview of the selected file and set the preview active using the `set_preview_widget_active()` method
- otherwise, set the preview inactive

When there is no application−supplied preview widget, or the application−supplied preview widget is not active, the file chooser may display an internally generated preview of the current file or it may display no preview at all.

gtk.FileChooser.get_preview_widget

```
def get_preview_widget()
```

Returns :	the current preview widget, or `None`

Note

This method is available in PyGTK 2.4 and above.

The `get_preview_widget()` method returns the value of the "preview_widget" property i.e. the current preview widget. See the `set_preview_widget()` method for more information.

gtk.FileChooser.set_preview_widget_active

```
def set_preview_widget_active(active)
```

`active` :	if `TRUE`, display the user−specified preview widget

Note

This method is available in PyGTK 2.4 and above.

The `set_preview_widget_active()` method sets the "preview_widget_active" property to the value of *`active`*. If *`active`* is `TRUE`, the preview widget set by the `set_preview_widget()` method should be shown for the current filename. When *`active`* is `FALSE`, the file chooser may display an internally generated preview of the current file or it may display no preview at all. See the `set_preview_widget()` for more details.

gtk.FileChooser.get_preview_widget_active

```
def get_preview_widget_active()
```

Returns :	`TRUE` if the preview widget is active for the current filename.

Note

This method is available in PyGTK 2.4 and above.

The `get_preview_widget_active()` method returns the value of the "preview−widget−active" property that indicates whether the preview widget set by the `set_preview_widget()` method should be shown for the current filename. See the set_preview_widget_active() method for more details.

gtk.FileChooser.set_use_preview_label

```
    def set_use_preview_label(use_label)
```

use_label :	if TRUE, display a stock label with the name of the previewed file

Note

This method is available in PyGTK 2.4 and above.

The set_use_preview_label() method sets the "use−preview−label" property to the value of *use_label*. If *use_label* is TRUE (the default), the file chooser should display a stock label with the name of the file that is being previewed. Applications that want to draw the whole preview area themselves should set this to FALSE and display the name themselves in their preview widget. See the set_preview_widget() method for more information.

gtk.FileChooser.get_use_preview_label

```
    def get_use_preview_label()
```

Returns :	TRUE if the file chooser is set to display a label with the name of the previewed file; FALSE otherwise.

Note

This method is available in PyGTK 2.4 and above.

The get_use_preview_label() method returns the value of the "use−preview−label" property that indicates whether a stock label should be drawn with the name of the previewed file. See the set_use_preview_label() for more information.

gtk.FileChooser.get_preview_filename

```
    def get_preview_filename()
```

Returns :	the filename to preview, or None.

Note

This method is available in PyGTK 2.4 and above.

The get_preview_filename() method returns the filename that should be previewed in a custom preview widget or None if no file is selected, or if the selected file cannot be represented as a local filename. See the set_preview_widget() method for more details.

gtk.FileChooser.get_preview_uri

```
    def get_preview_uri()
```

Returns :	the URI for the file to preview, or None.

Note

This method is available in PyGTK 2.4 and above.

The `get_preview_uri()` method returns the URI of the file that should be previewed in a custom preview widget or `None` if no file is selected. See the `set_preview_widget()` method fr more details.

gtk.FileChooser.set_extra_widget

```
def set_extra_widget(extra_widget)
```

`extra_widget` :	the widget to display extra options

Note

This method is available in PyGTK 2.4 and above.

The `set_extra_widget()` method sets the "extra−widget" property to the value of *`extra_widget`*. *`extra_widget`* is an application−supplied widget used to display extra options to the user.

gtk.FileChooser.get_extra_widget

```
def get_extra_widget()
```

Returns :	the current extra widget, or `None`.

Note

This method is available in PyGTK 2.4 and above.

The `get_extra_widget()` method returns the value of the "extra−widget" property that contains either a widget used to display extra options to the user or `None` if no extra widget is in use. See the `set_extra_widget()` for more information.

gtk.FileChooser.add_filter

```
def add_filter(filter)
```

`filter` :	a `gtk.FileFilter`

Note

This method is available in PyGTK 2.4 and above.

The `add_filter()` method adds the `gtk.FileFilter` specified by *`filter`* to the list of filters that the user can select from. When a filter is selected, only files that are passed by that filter are displayed.

gtk.FileChooser.remove_filter

```
def remove_filter(filter)
```

`filter` :	a `gtk.FileFilter`

Note

This method is available in PyGTK 2.4 and above.

The `remove_filter()` method removes the `gtk.FileFilter` specified by *`filter`* from the list of filters that the user can select from.

gtk.FileChooser.list_filters

```
    def list_filters()
```

Returns :	a list containing the current set of user selectable filters.

Note

This method is available in PyGTK 2.4 and above.

The `list_filters()` method returns the current set of user–selectable filters. See the `add_filter()` and `remove_filter()` method for more details.

gtk.FileChooser.set_filter

```
    def set_filter(filter)
```

`filter` :	a `gtk.FileFilter`

Note

This method is available in PyGTK 2.4 and above.

The `set_filter()` method sets the "filter" property to the value of *`filter`* and also sets the current filter to *`filter`*. Only the files that pass *`filter`* will be displayed. If the user–selectable list of filters is non–empty, then *`filter`* should be one of the filters in that list. Setting the current filter when the list of filters is empty is useful if you want to restrict the displayed set of files without letting the user change it.

gtk.FileChooser.get_filter

```
    def get_filter()
```

Returns :	the current filter, or `None`.

Note

This method is available in PyGTK 2.4 and above.

The `get_filter()` method returns the value of the "filter" property which is the current filter. See the `set_filter()` method for more information.

gtk.FileChooser.add_shortcut_folder

```
    def add_shortcut_folder(folder)
```

`folder` :	the filename of the folder to add
Returns :	`TRUE` if the folder could be added successfully.

Note

This method is available in PyGTK 2.4 and above.

The `add_shortcut_folder()` adds the folder specified by *`folder`* the list of shortcut folders in a file chooser. Shortcut folders are displayed at the upper left in the `gtk.FileChooser`. By double–clicking on a shortcut the user can open that folder directly. Note that shortcut folders do not get saved, as they are provided by the application. For example, you can use this to add a "/usr/share/mydrawprogram/Clipart" folder to the volume list.

The GError exception is raised if an error occurred while adding the folder.

gtk.FileChooser.remove_shortcut_folder

```
def remove_shortcut_folder(folder)
```

`folder` :	the filename of the folder to remove
Returns :	`TRUE` if *`folder`* was removed from the list of shortcut folders.

Note

This method is available in PyGTK 2.4 and above.

The `remove_shortcut_folder()` method removes the folder specified by *`folder`* from a file chooser's list of shortcut folders. `remove_shortcut_folder()` returns TRUE if successful. See the `add_shortcut_folder()` method for more information.

The GError exception is raised if an error occurred while removing the folder.

gtk.FileChooser.list_shortcut_folders

```
def list_shortcut_folders()
```

Returns :	A list of shortcut folder filenames, or `None` if there are no shortcut folders.

Note

This method is available in PyGTK 2.4 and above.

The `list_shortcut_folders()` method returns the list of shortcut folders in the file chooser, as set by the `add_shortcut_folder()` method or `None` if there are no shortcut folders. It is not possible to get a list of the user–specified shortcut folders.

gtk.FileChooser.add_shortcut_folder_uri

```
def add_shortcut_folder_uri(uri)
```

`uri` :	the URI of the folder to add
Returns :	`TRUE` if the folder was added

Note

This method is available in PyGTK 2.4 and above.

The `add_shortcut_folder_uri()` method adds a folder with the URI specified by *`uri`* to the list of shortcut folders in a file chooser. Note that shortcut folders do not get saved, as they are provided by the application. For example, you can use this to add a "file:///usr/share/mydrawprogram/Clipart" folder to the volume list. See the `add_shortcut_folder()` method for more details.

The GError exception is raised if an error occurred while adding the folder.

gtk.FileChooser.remove_shortcut_folder_uri

```
    def remove_shortcut_folder_uri(uri)
```

`uri` :	URI of the folder to remove
Returns :	`TRUE` if the folder was removed.

Note

This method is available in PyGTK 2.4 and above.

The `remove_shortcut_folder_uri()` method removes the folder with the URI specified by *`uri`* from the file chooser's list of shortcut folders.

The GError exception is raised if an error occurred while removing the folder.

gtk.FileChooser.list_shortcut_folder_uris

```
    def list_shortcut_folder_uris()
```

Returns :	a list of shortcut folder URIs, or `None`

Note

This method is available in PyGTK 2.4 and above.

The `list_shortcut_folder_uris()` method returns a list of the shortcut folders in the file chooser, as set by the `add_shortcut_folder_uri()` method. It is not possible to get a list of the user−specified folder URIs.

gtk.FileChooser.set_show_hidden

```
    def set_show_hidden(show_hidden)
```

`show_hidden` :	if `TRUE` hidden files and folders should be displayed.

Note

This method is available in PyGTK 2.6 and above.

The `set_show_hidden()` method sets the "show−hidden" property to the value of *`show_hidden`*. If *`show_hidden`* is `TRUE`, hidden files and folders should be displayed in the file selector.

gtk.FileChooser.get_show_hidden

```
    def get_show_hidden()
```

Returns :	`TRUE` if hidden files and folders are displayed.

Note

This method is available in PyGTK 2.6 and above.

The `get_show_hidden()` method returns the value of the "show−hidden" property that indicates whether hidden files and folders should be displayed in the file selector. See the `set_show_hidden()` method for more information.

Signals

The "current−folder−changed" gtk.FileChooser Signal

```
def callback(filechooser, user_param1, ...)
```

filechooser :	the filechooser widget that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

Note

This signal is available in PyGTK 2.4 and above.

The "current−folder−changed" signal is emitted when the current folder displayed in *filechooser* is changed. Normally you do not need to connect to this signal, unless you need to keep track of which folder a file chooser is showing.

The "file−activated" gtk.FileChooser Signal

```
def callback(filechooser, user_param1, ...)
```

filechooser :	the filechooser widget that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

Note

This signal is available in PyGTK 2.4 and above.

The "file−activated" signal is emitted when the user double−clicks on a file (not a folder) or presses **Enter**. Normally you do not need to connect to this signal. It is used internally by `gtk.FileChooserDialog` to know when to activate the default button in the dialog.

The "selection−changed" gtk.FileChooser Signal

```
def callback(filechooser, user_param1, ...)
```

filechooser :	the filechooser widget that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

Note

This signal is available in PyGTK 2.4 and above.

The "selection−changed" signal is emitted when the file selection in *filechooser* is changed either by clicking on a filename or by changing the current folder. Normally you do not need to connect to this signal, as it is easier to wait for the file chooser to finish running, and then to get the list of selected files using the functions mentioned below.

The "update−preview" gtk.FileChooser Signal

```
    def callback(filechooser, user_param1, ...)
```

filechooser :	the filechooser widget that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

Note

This signal is available in PyGTK 2.4 and above.

The "update−preview" signal is emitted when the preview when a file chooser should be regenerated. For example, this can happen when the currently selected file changes. You should use this signal if you want your file chooser to have a preview widget.

Once you have installed a preview widget with the `set_preview_widget()` method, you should update it when this signal is emitted. You can use the methods `get_preview_filename()` or `get_preview_uri()` to get the name of the file to preview. Your widget may not be able to preview all kinds of files so your callback must call the `set_preview_widget_active()` to inform the file chooser if the preview was generated successfully or not.

Please see the example code in the section called Adding a Preview Widget.

Also see the `set_preview_widget()`, `set_preview_widget_active()`, `set_use_preview_label()`, `get_preview_filename()` and `get_preview_uri()` methods for more information

gtk.FileChooserButton

gtk.FileChooserButton — a button to launch a `gtk.FileChooserDialog` (new in PyGTK 2.6)

Synopsis

```
class gtk.FileChooserButton(gtk.HBox):
    gtk.FileChooserButton(title, backend=None)
    gtk.FileChooserButton(dialog)
    def get_title()
    def set_title(title)
    def get_width_chars()
    def set_width_chars(n_chars)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Box
           +-- gtk.HBox
             +-- gtk.FileChooserButton
```

Properties

"dialog"	Write−Construct Only	The `gtk.FileChooserDialog` associated with the button.
"title"	Read−Write	The string to use as the title on the `gtk.FileChooserDialog` associated with the button.
"width−chars"	Read−Write	The width of the entry and label inside the button, in characters. Allowed values: >= −1. Default value: −1

Description

Note

This widget is available in PyGTK 2.6 and above.

The `gtk.FileChooserButton` is a widget that lets the user select a file. It implements the `gtk.FileChooser` interface. Visually, it is a file name with a button to bring up a `gtk.FileChooserDialog`. The user can then use that dialog to change the file associated with that button. This widget does not support setting the "select−multiple" property to `TRUE`. For example to create a `gtk.FileChooserButton` and set the current folder to '/etc' use:

```
  filechooserbutton = gtk.FileChooserButton('Select a File')
  filechooserbutton.set_current_folder('/etc')
```

The `gtk.FileChooserButton` supports the `gtk.FILE_CHOOSER_ACTION_OPEN` and `gtk.FILE_CHOOSER_ACTION_SELECT_FOLDER` actions of the GTK FileChooser Action Constants.

Note

The `gtk.FileChooserButton` will ellipsize the label, and thus will thus request little horizontal space. To give the button more space, you should call the `gtk.Widget.size_request()` method, the `set_width_chars()` method, or pack the button in such a way that other interface elements give space to the widget.

Constructor

gtk.FileChooserButton

```
gtk.FileChooserButton(title, backend=None)
```

title :	the title of the browse dialog
backend :	the name of a file system backend or `None`
Returns :	a new `gtk.FileChooserButton`

Note

This constructor is available in PyGTK 2.6 and above.

Creates a new button widget that opens a `gtk.FileChooserDialog` when clicked. The title of the `gtk.FileChooserDialog` is specified by *`title`*. If *`backend`* is specified it is the name of a file system backend.

gtk.FileChooserButton

```
gtk.FileChooserButton(dialog)
```

dialog :	a `gtk.FileChooserDialog`
Returns :	a new `gtk.FileChooserButton`

Note

This constructor is available in PyGTK 2.6 and above.

Creates a new button widget that opens a `gtk.FileChooserDialog` specified by *`dialog`* when clicked.

Methods

gtk.FileChooserButton.get_title

```
def get_title()
```

Returns :	the title of the `gtk.FileChooserDialog`

Note

This method is available in PyGTK 2.6 and above.

The `get_title`() method returns the value of the "title" property which contains the title of the associated `gtk.FileChooserDialog`.

gtk.FileChooserButton.set_title

```
def set_title(title)
```

title :	a string to use as the title of the associated `gtk.FileChooserDialog`.

Note

This method is available in PyGTK 2.6 and above.

The `set_title()` method sets the "title" property to the value of *`title`*. The "title" property contains the title string of the associated gtk.FileChooserDialog.

gtk.FileChooserButton.get_width_chars

```
    def get_width_chars()
```

Returns :, :,	the width in characters of the button

Note

This method is available in PyGTK 2.6 and above.

The `get_width_chars()` method returns the value of the "width–characters" property which contains the number of characters the button width should be set to.

gtk.FileChooserButton.set_width_chars

```
    def set_width_chars(n_chars)
```

`n_chars` :	the width in characters for the button

Note

This method is available in PyGTK 2.6 and above.

The `set_width_chars()` method sets the "width–chars" property to the value of *`n_chars`*. The "width–chars" property contains the width in characters that the button should be set to.

gtk.FileChooserDialog

gtk.FileChooserDialog   a file chooser dialog, suitable for "File/Open" or "File/Save" commands(new in PyGTK 2.4)

Synopsis

```
class gtk.FileChooserDialog(gtk.Dialog, gtk.FileChooser):
    gtk.FileChooserDialog(title=None, parent=None, action=gtk.FILE_CHOOSER_ACTION_OPEN, buttons
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Window
            +-- gtk.Dialog
```

```
              +-- gtk.FileChooserDialog (implements gtk.FileChooser)
```

Description

Note

This widget is available in PyGTK 2.4 and above.

`gtk.FileChooserDialog` is a dialog box suitable for use with "File/Open" or "File/Save as" commands. This widget works by putting a `gtk.FileChooserWidget` inside a `gtk.Dialog`. It implements the `gtk.FileChooser` interface, so you can use all of the `gtk.FileChooser` methods, signals and properties on the file chooser dialog as well as those for `gtk.Dialog`.

Note that `gtk.FileChooserDialog` does not have any methods of its own. Instead, you should use the methods and signals that work on a `gtk.FileChooser`.

Constructor

```
    gtk.FileChooserDialog(title=None, parent=None, action=gtk.FILE_CHOOSER_ACTION_OPEN, buttons=
```

`title` :	The title of the dialog
`parent` :	The transient parent of the dialog, or `None`
`action` :	The open or save mode for the dialog – one of: `gtk.FILE_CHOOSER_ACTION_OPEN`, `gtk.FILE_CHOOSER_ACTION_SAVE`, `gtk.FILE_CHOOSER_ACTION_SELECT_FOLDER` or `gtk.FILE_CHOOSER_ACTION_CREATE_FOLDER`.
`buttons` :	a tuple containing button label–response id pairs or `None`
`backend` :	The name of the specific filesystem backend to use.
Returns :	a new `gtk.FileChooserDialog`

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new `gtk.FileChooserDialog`. The constructor can be invoked with the optional parameters: *`title`*, *`parent`*, *`action`*, *`buttons`* and *`backend`*. This constructor is analogous to the gtk.Dialog() constructor.

gtk.FileChooserWidget

gtk.FileChooserWidget — a file chooser widget that can be embedded in other widgets(new in PyGTK 2.4)

Synopsis

```
class gtk.FileChooserWidget(gtk.VBox, gtk.FileChooser):
    gtk.FileChooserWidget(action=gtk.FILE_CHOOSER_ACTION_OPEN, backend=None)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Box
          +-- gtk.VBox
            +-- gtk.FileChooserWidget (implements gtk.FileChooser)
```

Description

Note

This widget is available in PyGTK 2.4 and above.

The `gtk.FileChooserWidget` is a widget suitable for selecting files. It is the main building block of a `gtk.FileChooserDialog`. Most applications will only need to use the latter; you can use `gtk.FileChooserWidget` as part of a larger window if you have special needs.

Note that `gtk.FileChooserWidget` does not have any methods, signals or properties of its own. Instead, you should use the methods, signals and properties of the `gtk.FileChooser`.

Constructor

```
    gtk.FileChooserWidget(action=gtk.FILE_CHOOSER_ACTION_OPEN, backend=None)
```

`action` :	The open or save mode for the widget or `None`.
`backend` :	The name of the specific filesystem backend to use or `None`.
Returns :	a new `gtk.FileChooserWidget`

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new `gtk.FileChooserWidget`. This is a file chooser widget that can be embedded in custom windows, and it is the same widget that is used by `gtk.FileChooserDialog`.

Prev	Up	Next
gtk.FileChooserDialog	Home	gtk.FileFilter

gtk.FileFilter

Prev	**The gtk Class Reference**	Next

gtk.FileFilter

gtk.FileFilter — a filter for selecting a file subset (new in PyGTK 2.4)

Synopsis

```
class gtk.FileFilter(gtk.Object):
    gtk.FileFilter()
    def set_name(name)
    def get_name()
    def add_mime_type(mime_type)
    def add_pattern(pattern)
    def add_custom(needed, func, data)
    def get_needed()
    def filter(filter_info)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.FileFilter
```

Description

Note

This widget is available in PyGTK 2.4 and above.

A `gtk.FileFilter` is an object that filters files based on a set of rules that it contains. The categories of information that `gtk.FileFilter` uses to accept or reject the file are:

`gtk.FILE_FILTER_FILENAME`	The full path name of the file.
`gtk.FILE_FILTER_URI`	The URI of the file.
`gtk.FILE_FILTER_DISPLAY_NAME`	The simple name of the file as displayed in a file chooser.
`gtk.FILE_FILTER_MIME_TYPE`	The MIME type of the file.

The `add_pattern()` method adds a rule that only uses the display name (`gtk.FILE_FILTER_DISPLAY_NAME`) for filtering. The `add_mime_type()` method adds a rule that only uses the mime type (`gtk.FILE_FILTER_MIME_TYPE`) for filtering. To use the file URI (`gtk.FILE_FILTER_URI`) or filename (`gtk.FILE_FILTER_FILENAME`) you have to create a custom filter rule using a callback function that is registered with the `add_custom()` method.

The pattern rule uses file globbing to match the file display name:

- '*' matches any combination of characters e.g.. "a*c" matches "abc", "a bridge tic", "aaabbbc" and so on.
- '?' matches any single character e.g. "a?c" matches "abc", aZc" and so on but not "abbc".
- '[' and ']' enclose a set of characters that can be matched; ranges of characters can be included by separating the start and end with a dash (e.g. "a–z" include all the lowercase letters). e.g. "a[0–9]c" matches "a3c" and "a9c" but not "a28c" or "abc".
- '\' escapes the next character to allow "*", "?", "[" and "]" to be matched literally.

The MIME type requires an exact match (no pattern matching).

Constructor

```
    gtk.FileFilter()
```

Returns :	a new `gtk.FileFilter`

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new `gtk.FileFilter` with no rules added to it. Such a filter doesn't pass any files, so it's not particularly useful until you add rules with the `add_mime_type()`, `add_pattern()` or `add_custom()` methods. To create a filter that accepts any file, use:

```
  filter = gtk.FileFilter()
  filter.add_pattern("*")
```

Methods

gtk.FileFilter.set_name

```
    def set_name(name)
```

`name` :	the human–readable–name for the filter.

Note

This method is available in PyGTK 2.4 and above.

The `set_name`() method sets the human–readable name of the filter to the string in *`name`*. The string in *`name`* will be displayed in the file chooser user interface if there is a selectable list of filters.

gtk.FileFilter.get_name

```
    def get_name()
```

Returns :	The human–readable name of the filter, or `None`.

Note

This method is available in PyGTK 2.4 and above.

The `get_name`() method returns the human–readable name for the filter or `None` if the name has not been set. See the `set_name()` method.

gtk.FileFilter.add_mime_type

```
    def add_mime_type(mime_type)
```

`mime_type` :	the name of a MIME type

Note

This method is available in PyGTK 2.4 and above.

The `add_mime_type`() method adds a rule allowing the mime type specified by *`mime_type`* to be matched. Sets the needs value to `gtk.FILE_FILTER_MIME_TYPE` (see the get_needed() method for more information).

gtk.FileFilter.add_pattern

```
    def add_pattern(pattern)
```

pattern :	a shell style glob pattern

Note

This method is available in PyGTK 2.4 and above.

The `add_pattern()` method adds a rule allowing the shell style glob pattern specified by *pattern* to filter file names. Sets the needs value to `gtk.FILE_FILTER_DISPLAY_NAME` (see the get_needed() method for more information).

The pattern rule uses file globbing to match the file display name:

- '*' matches any combination of characters e.g.. "a*c" matches "abc", "a bridge tic", "aaabbbc" and so on.
- '?' matches any single character e.g. "a?c" matches "abc", aZc" and so on but not "abbc".
- '[' and ']' enclose a set of characters that can be matched; ranges of characters can be included by separating the start and end with a dash (e.g. "a–z" include all the lowercase letters). e.g. "a[0–9]c" matches "a3c" and "a9c" but not "a28c" or "abc".
- '\' escapes the next character to allow "*", "?", "[" and "]" to be matched literally.

gtk.FileFilter.add_custom

```
    def add_custom(needed, func, data)
```

needed :	a bitfield of flags indicating the information that the custom filter function needs.
func :	a callback function; if the function returns `TRUE`, then the file will be displayed.
data :	the data to pass to *func*

Note

This method is available in PyGTK 2.4 and above.

The `add_custom()` method adds a rule to a filter that allows files to be filtered based on a custom callback function specified by *func*. The bitfield *needed* provides information about what sorts of information that the filter function needs; this allows GTK+ to avoid retrieving expensive information when it isn't needed by the filter. *needed* is a combination of:

`gtk.FILE_FILTER_FILENAME`	The full path name of the file.
`gtk.FILE_FILTER_URI`	The URI of the file.
`gtk.FILE_FILTER_DISPLAY_NAME`	The simple name of the file as displayed in a file chooser.
`gtk.FILE_FILTER_MIME_TYPE`	The MIME type of the file.

The signature of *func* is:

```
  def filefilterfunction(filter_info, data):
```

where *filter_info* is a 4–tuple where each item is either a string or `None`. The strings correspond to: the full pathname of the file, the URI of the file, the display name of the file and the MIME type of the file. *data* is the value passed in as the *data* parameter in the `add_custom()` method. Using a custom filter function is the only way to filter files based on file URIs or full file pathnames.

gtk.FileFilter.get_needed

```
    def get_needed()
```

Returns :	a bitfield of flags indicating the needed fields when calling `filter()`

Note

This method is available in PyGTK 2.4 and above.

The `get_needed`() method returns the information that is needed by the `gtk.FileFilter` to filter the file info using the `filter()`

This method is not typically used by applications; it is intended principally for use in the implementation of `gtk.FileChooser`.

gtk.FileFilter.filter

```
    def filter(filter_info)
```

`filter_info` :	a 4–tuple containing the information about a file.
Returns :	`TRUE` if the file should be displayed

Note

This method is available in PyGTK 2.4 and above.

The `filter`() method tests whether a file should be displayed according to the file filter rules. The 4–tuple *`filter_info`* should include the fields returned from the `get_needed()` method:

- the full pathname of the file if needs includes `gtk.FILE_FILTER_FILENAME`
- the URI of the file is needs includes `gtk.FILE_FILTER_URI`
- the display name (without the path) if the file if needs includes `gtk.FILE_FILTER_DISPLAY_NAME`
- the MIME type of the file if needs includes `gtk.FILE_FILTER_MIME_TYPE`

This method will not typically be used by applications; it is intended principally for use in the implementation of `gtk.FileChooser`.

Prev Up Next
gtk.FileChooserWidget Home gtk.FileSelection

gtk.FileSelection
Prev **The gtk Class Reference** Next

gtk.FileSelection

gtk.FileSelection — a dialog used to prompt the user for a file or directory name

Synopsis

```
class gtk.FileSelection(gtk.Dialog):
    gtk.FileSelection(title=None)
    def set_filename(filename)
    def get_filename()
    def complete(pattern)
    def show_fileop_buttons()
    def hide_fileop_buttons()
    def get_selections()
    def set_select_multiple(select_multiple)
    def get_select_multiple()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Window
            +-- gtk.Dialog
              +-- gtk.FileSelection
```

Properties

"show−fileops"	Read/Write	If `TRUE` display the buttons for creating and manipulating files.
"filename"	Read/Write	The currently selected filename.
"select−multiple"	Read/Write	If `TRUE` allow multiple files to be selected.

Attributes

"dir_list"	Read	The `gtk.TreeView` widget used to display the directories
"file_list"	Read	The `gtk.TreeView` widget used to display the files.
"selection_entry"	Read	The `gtk.Entry` widget used to display the current file selection
"selection_text"	Read	The `gtk.Label` associated with the selection entry.
"main_vbox"	Read	The `gtk.VBox` widget that holds all the widgets of the `gtk.FileSelection` dialog.
"ok_button"	Read	The "OK" button.
"cancel_button"	Read	The "Cancel" button.
"history_pulldown"	Read	The `gtk.OptionMenu` widget that provides the pulldown list of directory paths.
"history_menu"	Read	The `gtk.Menu` part of the *`history_pulldown`*
"fileop_dialog"	Read	The dialog box used to display the `gtk.FileSelection`. It can be customized by adding or removing widgets from it using the standard `gtk.Dialog` methods.
"fileop_entry"	Read	The `gtk.Entry` in the fileops dialog that is created when one of the fileops buttons is clicked.
"fileop_file"	Read	The name of the file to be operated on by the fileops
"fileop_c_dir"	Read	The "New Folder" fileops button.

"fileop_del_file"	Read	The "Delete File" fileops button
"fileop_ren_file"	Read	The "Rename File" fileops button
"button_area"	Read	The fileops `gtk.ButtonBox`.
"action_area"	Read	A `gtk.HBox` that can be used by the application to add buttons, etc.

Description

The `gtk.FileSelection` should be used to retrieve file or directory names from the user. It will create a new dialog window containing a directory list, and a file list corresponding to the current working directory. The filesystem can be navigated using the directory list or the drop−down history menu. Alternatively, the **TAB** key can be used to navigate using filename completion − common in text based editors such as emacs and jed.

File selection dialogs are created with a call to the `gtk.FileSelection()` constructor.

The default filename can be set using the `set_filename()` method and the selected filename retrieved using the `get_filename()` method.

Use the `complete()` method to display files and directories that match a given pattern. This can be used for example, to show only *.txt files, or only files beginning with gtk*.

Simple file operations; create directory, delete file, and rename file, are available from buttons at the top of the dialog. These can be hidden using the `hide_fileop_buttons()` method and shown again using the `show_fileop_buttons()` method.

Constructor

```
gtk.FileSelection(title=None)
```

`title` :	the text to be used as the title of the fileselection dialog.
Returns :	a new fileselection dialog

Creates a new file selection dialog. By default it will contain `gtk.TreeViews` displaying the application's current working directory, and its file listing. File operation buttons that allow the user to create a directory, delete files and rename files, are also present.

Methods

gtk.FileSelection.set_filename

```
def set_filename(filename)
```

`filename` :	a string to set as the default file name.

The `set_filename()` method sets a default path for the file requester. If *`filename`* includes a directory path, then the fileselection will open with that path as its current working directory.

Note the encoding of *`filename`* is the on−disk encoding, which may not be UTF−8.

gtk.FileSelection.get_filename

```
def get_filename()
```

Returns :	currently−selected filename in the on−disk encoding

The `get_filename()` method returns the selected filename in the on−disk encoding, which may or may not be the same as that used by GTK (UTF−8). If no file is selected then the selected directory path is returned.

gtk.FileSelection.complete

```
def complete(pattern)
```

`pattern` :	a string containing a pattern which may or may not match any filenames in the current directory.

The `complete()` method will attempt to match *`pattern`* to valid filenames or subdirectories in the current directory. If a match can be made, the matched filename will appear in the text entry field of the file selection dialog. If a partial match can be made, the "Files" list will contain those file names which have been partially matched, and the "Directories" list those directories which have been partially matched.

gtk.FileSelection.show_fileop_buttons

```
def show_fileop_buttons()
```

The `show_fileop_buttons()` method shows the fileops buttons: "New Folder", "Delete File" and "Rename File".

gtk.FileSelection.hide_fileop_buttons

```
def hide_fileop_buttons()
```

The `hide_fileop_buttons()` method hides the fileops buttons: "New Folder", "Delete File" and "Rename File".

gtk.FileSelection.get_selections

```
def get_selections()
```

Returns :	a tuple containing the selected files.

The `get_selections()` method retrieves a tuple containing the file selections the user has made in the file selection dialog. The first file in the list is equivalent to what the `get_filename()` method would return.

gtk.FileSelection.set_select_multiple

```
def set_select_multiple(select_multiple)
```

`select_multiple` :	If `TRUE` the user is allowed to select multiple files in the file list.

The `set_select_multiple()` method sets the file list selection mode according to the value of *`select_multiple`*. If *`select_multiple`* is `TRUE` the user is allowed to select multiple files in the file list. Use the `get_selections()` method to retrieve the list of selected files.

gtk.FileSelection.get_select_multiple

```
  def get_select_multiple()
```

Returns :	`TRUE` if the user is allowed to select multiple files in the file list

The `get_select_multiple`() method determines whether the user is allowed to make multiple file selection in the file list. If the `get_select_multiple`() method returns `TRUE` the user is allowed to select multiple files in the file list. See the `set_select_multiple()` method.

gtk.Fixed

gtk.Fixed — a container which allows you to position widgets at fixed coordinates

Synopsis

```
class gtk.Fixed(gtk.Container):
    gtk.Fixed()
    def put(widget, x, y)
    def move(widget, x, y)
    def set_has_window(has_window)
    def get_has_window()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Fixed
```

Child Properties

"x"	Read/Write	The x position of the child widget.
"y"	Read/Write	The y position of the child widget.

Description

The `gtk.Fixed` widget is a container that can place child widgets at fixed positions and with fixed sizes, given in pixels. `gtk.Fixed` performs no automatic layout management. For most applications, you should not use this container! It keeps you from having to learn about the other PyGTK containers, but it results in broken applications. With `gtk.Fixed`, the following things will result in truncated text, overlapping widgets, and other display bugs:

- Themes, which may change widget sizes.
- Fonts other than the one you used to write the app will of course change the size of widgets containing text; keep in mind that users may use a larger font because of difficulty reading the default,

or they may be using Windows or the framebuffer port of PyGTK, where different fonts are available.
- Translation of text into other languages changes its size. Also, display of non–English text will use a different font in many cases.

In addition, the fixed widget can't properly be mirrored in right–to–left languages such as Hebrew and Arabic. i.e. normally PyGTK will flip the interface to put labels to the right of the thing they label, but it can't do that with `gtk.Fixed`. So your application will not be usable in right–to–left languages. Finally, fixed positioning makes it kind of annoying to add and remove GUI elements, since you have to reposition all the other elements. This is a long–term maintenance problem for your application. If you know none of these things are an issue for your application, and prefer the simplicity of `gtk.Fixed`, by all means use the widget. But you should be aware of the tradeoffs.

Constructor

```
gtk.Fixed()
```

Returns :	a new fixed widget

Creates a new `gtk.Fixed` widget

Methods

gtk.Fixed.put

```
def put(widget, x, y)
```

`widget` :	the child widget being added*
`x` :	the x position of the widget location
`y` :	the y position of the widget location

The `put`() method adds the child widget specified by *`widget`* to the gtk.Fixed widget at the location specified by *x* and *y*.

gtk.Fixed.move

```
def move(widget, x, y)
```

`widget` :	the child widget
`x` :	the new x position
`y` :	the new y position

The `move`() method moves the child widget specified by *`widget`* to the location specified specified by *x* and *y*.

gtk.Fixed.set_has_window

```
def set_has_window(has_window)
```

`has_window` :	if `TRUE` a separate window should be created

The `set_has_window`() method specifies whether a `gtk.Fixed` widget is created with a separate `gtk.gdk.Window` according to the value of *`has_window`*. If *`has_window`* is `TRUE` the fixed widget will be created with its own separate window. By default, the setting is `FALSE` and the fixed will be created with no separate `gtk.gdk.Window`. This method must be called while the `gtk.Fixed` is not realized, for instance, immediately after the window is created.

gtk.Fixed.get_has_window

```
    def get_has_window()
```

Returns :	`TRUE` if the fixed widget has its own window.

The `get_has_window()` method returns `TRUE` if the `gtk.Fixed` widget has it's own `gtk.gdk.Window`. See the `set_has_window()` method.

gtk.FontButton

gtk.FontButton a button to launch a font selection dialog (new in PyGTK 2.4)

Synopsis

```
class gtk.FontButton(gtk.Button):
    gtk.FontButton(fontname=None)
    def get_title()
    def set_title(title)
    def get_use_font()
    def set_use_font(use_font)
    def get_use_size()
    def set_use_size(use_size)
    def get_font_name()
    def set_font_name(fontname)
    def get_show_style()
    def set_show_style(show_style)
    def get_show_size()
    def set_show_size(show_size)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Button
            +-- gtk.FontButton
```

Properties

"font−name"	Read−Write	The name of the currently selected font.
"show−size"	Read−Write	If `TRUE`, the selected font size will be shown in the label. For a more WYSIWIG way to show the selected size, see the "use−size" property.
"show−style"	Read−Write	If `TRUE`, the name of the selected font style will be shown in the label. For a more WYSIWIG way to show the selected style, see the "use−font" property.
"title"	Read−Write	The title of the font selection dialog.
"use−font"	Read−Write	If `TRUE`, the label will be drawn in the selected font
"use−size"	Read−Write	If `TRUE`, the label will be drawn with the selected font size.

Signal Prototypes

"font−set"	`def callback(`*`fontbutton`*`, `*`user_param1`*`, ...)`

Description

Note

This widget is available in PyGTK 2.4 and above.

The `gtk.FontButton` is a button that displays the currently selected font and, when clicked, opens a `gtk.FontSelectionDialog` to change the font. A `gtk.FontButton` can be used in a preference dialog for selecting a font.

Constructor

```
gtk.FontButton(fontname=None)
```

`fontname` :	the name of the font to display in the font selection dialog
Returns :	a new font button widget

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new `gtk.FontButton` that displays the font specified by *`fontname`* or the currently selected font if *`fontname`* is `None` or not specified.

Methods

gtk.FontButton.get_title

```
def get_title()
```

Returns :	the title string

Note

This method is available in PyGTK 2.4 and above.

The `get_title`() method returns the value of the "title" property that contains the title of the font selection dialog.

gtk.FontButton.set_title

```
def set_title(title)
```

title :	a string containing the font selection dialog title

Note

This method is available in PyGTK 2.4 and above.

The `set_title`() method sets the "title" property to the string specified by *`title`* and sets the title of the `gtk.FontSelectionDialog` if it exists. The "title" property contains the title of the font selection dialog.

gtk.FontButton.get_use_font

```
def get_use_font()
```

Returns :	`TRUE`, if the font button label is written using the selected font.

Note

This method is available in PyGTK 2.4 and above.

The `get_use_font`() method returns the value of the "use–font" property. If "use–font" is `TRUE` the selected font is used in the label.

gtk.FontButton.set_use_font

```
def set_use_font(use_font)
```

use_font :	If `TRUE`, the font button label will be written using the font selected.

Note

This method is available in PyGTK 2.4 and above.

The `set_use_font`() method sets the "use–font" property to the value specified by *`use_font`*. If *`use_font`* is `TRUE`, the font button label will be written using the selected font.

gtk.FontButton.get_use_size

```
def get_use_size()
```

Returns :	`TRUE` if the font button label is written using the selected size.

Note

This method is available in PyGTK 2.4 and above.

The `get_use_size`() method returns the value of the "use–size" property. If the value of "use–size" is `TRUE`, the font button label is written using the selected font size.

gtk.FontButton.set_use_size

```
def set_use_size(use_size)
```

`use_size` :	If `TRUE`, the font button label will be written using the selected size.

Note

This method is available in PyGTK 2.4 and above.

The `set_use_size`() method sets the "use−size" property to the value of `use_size`. If *`use_size`* is `TRUE`, the font button label will be written using the selected size.

gtk.FontButton.get_font_name

```
def get_font_name()
```

Returns :	the font name.

Note

This method is available in PyGTK 2.4 and above.

The `get_font_name`() method returns the value of the "font−name" property that contains the name of the currently selected font.

gtk.FontButton.set_font_name

```
def set_font_name(fontname)
```

`fontname` :	the name of the font to display in the font selection dialog
Returns :	`TRUE` if the font selection dialog exists and the font name could be set

Note

This method is available in PyGTK 2.4 and above.

The `set_font_name`() method sets the "font−name" property to the value of *`fontname`* and updates the currently−displayed font in the font selection dialog. Returns `TRUE` if the font selection dialog exists and *`fontname`* could be set as its font.

gtk.FontButton.get_show_style

```
def get_show_style()
```

Returns :	`TRUE` if the font style will be shown in the label.

Note

This method is available in PyGTK 2.4 and above.

The `get_show_style`() method returns the value of the "show−style" property. If "show−style" is `TRUE`, the name of the font style will be shown in the font button label.

gtk.FontButton.set_show_style

```
def set_show_style(show_style)
```

`show_style` :	if `TRUE`, the font style should be displayed in the font button label.

Note

This method is available in PyGTK 2.4 and above.

The `set_show_style`() method sets the "show−style" property tot he value of *`show_style`*. If *`show_style`* is `TRUE`, the font style will be displayed along with name of the selected font in the font button label.

gtk.FontButton.get_show_size

```
def get_show_size()
```

Returns :	`TRUE` if the font size will be shown in the font button label.

Note

This method is available in PyGTK 2.4 and above.

The `get_show_size`() method returns the value of the "show−size" property. If "show−size" is `TRUE`, the font size will be shown in the font button label.

gtk.FontButton.set_show_size

```
def set_show_size(show_size)
```

`show_size` :	if `TRUE`, the font size should be displayed in the font button label.

Note

This method is available in PyGTK 2.4 and above.

The `set_show_size`() method sets the "show−size" property to to the value of *`show_size`*. If *`show_size`* is `TRUE`, the font size will be displayed along with the name of the selected font in the font button label.

Signals

The "font−set" gtk.FontButton Signal

```
def callback(fontbutton, user_param1, ...)
```

`fontbutton` :	the fontbutton that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "font−set" signal is emitted when the user selects a font. When handling this signal, use the `get_font_name()` method to find out what font was just selected.

gtk.FontSelection

gtk.FontSelection — a widget for selecting fonts.

Synopsis

```
class gtk.FontSelection(gtk.VBox):
    gtk.FontSelection()
    def get_font_name()
    def set_font_name(fontname)
    def get_preview_text()
    def set_preview_text(text)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Box
          +-- gtk.VBox
            +-- gtk.FontSelection
```

Properties

"font–name"	Read/Write	The string that names the font
"font"	Read	The gtk.gdk.Font that is currently selected. This property is deprecated.
"preview–text"	Read/Write	The text to display in order to demonstrate the selected font.

Description

The `gtk.FontSelection` widget lists the available fonts, styles and sizes, allowing the user to select a font. It is used in the `gtk.FontSelectionDialog` widget to provide a dialog box for selecting fonts. The `set_font_name()` method sets the initial font selection. The current font selection is retrieved using the `get_font_name()` method.

The fontselection has a preview area that contains a `gtk.Entry` that displays text using the currently selected font. The preview text can be retrieved with the `get_preview_text()` method and set with the `set_preview_text()` method.

Filters can be used to limit the font selections. There are 2 filters in the `gtk.FontSelection` – a base filter and a user filter. The base filter cannot be changed by the user, so this can be used when the user must choose from the restricted set of fonts (e.g. for a terminal–type application you may want to force the user to select a fixed–width font). The user filter can be changed or reset by the user, by using the Reset Filter button or changing the options on the Filter page of the widget.

Constructor

```
gtk.FontSelection()
```

Returns :	a new fontselection widget

Creates a new `gtk.FontSelection` widget.

Methods

gtk.FontSelection.get_font_name

```
def get_font_name()
```

Returns :	a string containing the selected font name

The `get_font_name()` method returns the name of the currently selected font.

gtk.FontSelection.set_font_name

```
def set_font_name(fontname)
```

`fontname` :	a string containing the name of a font
Returns :	`TRUE` if the font could be set

The `set_font_name()` method sets the currently selected font in the fontselection using the value of *`fontname`*. The method returns `TRUE` if the font is found and can be selected in the fontselection.

gtk.FontSelection.get_preview_text

```
def get_preview_text()
```

Returns :	a string containing the preview text

The `get_preview_text()` method returns a string that contains the text displayed in the preview area entry.

gtk.FontSelection.set_preview_text

```
def set_preview_text(text)
```

`text` :	the new preview text string

The `set_preview_text()` method sets the text to be displayed in the preview area entry using the string in *`text`*.

gtk.FontSelectionDialog

gtk.FontSelectionDialog a dialog for selecting fonts.

Synopsis

```
class gtk.FontSelectionDialog(gtk.Dialog):
    gtk.FontSelectionDialog(title)
    def get_font_name()
    def set_font_name(fontname)
    def get_preview_text()
    def set_preview_text(text)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Window
            +-- gtk.Dialog
              +-- gtk.FontSelectionDialog
```

Attributes

"fontsel"	Read	The `gtk.FontSelection` widget in the dialog.
"main_vbox"	Read	The `gtk.VBox` that contains all the dialog widgets
"action_area"	Read	The `gtk.HBox` containing the action buttons
"ok_button"	Read	The OK button
"apply_button"	Read	The Apply button
"cancel_button"	Read	The Cancel button

Description

The `gtk.FontSelectionDialog` is a dialog box containing a `gtk.FontSelection` widget that the user can use to select a font according to the desired family, style and size.

The `set_font_name()` method sets the initial font selection. The current font selection is retrieved using the `get_font_name()` method. The font selection dialog has a preview area that contains a `gtk.Entry` that displays text using the currently selected font. The preview text can be retrieved with the `get_preview_text()` method and set with the `set_preview_text()` method.

Filters can be used to limit the font selections. There are 2 filters in the `gtk.FontSelectionDialog` – a base filter and a user filter. The base filter cannot be changed by the user, so this can be used when the user must choose from the restricted set of fonts (e.g. for a terminal–type application you may want to force the user to select a fixed–width font). The user filter can be changed or reset by the user, by using the Reset Filter button or changing the options on the Filter page of the widget.

Note

In GTK+ 2.2 and above the `gtk.FontSelectionDialog` does not have filters, a Reset Filter button or a Filter page.

Constructor

```
    gtk.FontSelectionDialog(title)
```

`title` :	a string to be used as the dialog title
Returns :	a new font selection dialog

Creates a new `gtk.FontSelectionDialog` with the title specified by *`title`*.

Methods

gtk.FontSelectionDialog.get_font_name

```
    def get_font_name()
```

Returns :	the currently selected font name or None if no font is selected.

The `get_font_name`() method returns a string containing the currently selected font name or None if no font name is selected.

gtk.FontSelectionDialog.set_font_name

```
    def set_font_name(fontname)
```

`fontname` :	a string containing the font name to be set
Returns :	`TRUE` if the font is found and can be selected

The `set_font_name`() method sets the current font using the value of *`fontname`*. The method returns `TRUE` if the font exists and could be selected.

gtk.FontSelectionDialog.get_preview_text

```
    def get_preview_text()
```

Returns :	a string containing the text in the preview area entry

The `get_preview_text`() method returns a string containing the text in the preview area entry.

gtk.FontSelectionDialog.set_preview_text

```
    def set_preview_text(text)
```

`text` :	a string used to set the text in the preview area entry

The `set_preview_text`() method sets the text in the preview area entry using the string specified by *`text`*.

Prev Up Next
gtk.FontSelection Home gtk.Frame

gtk.Frame

gtk.Frame — a bin with a decorative frame and optional label.

Synopsis

```
class gtk.Frame(gtk.Bin):
    gtk.Frame(label=None)
    def set_label(label)
    def get_label()
    def set_label_widget(label_widget)
    def get_label_widget()
    def set_label_align(xalign, yalign)
    def get_label_align()
    def set_shadow_type(type)
    def get_shadow_type()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Frame
```

Properties

"label"	Read−Write	The text of the frame's label
"label−xalign"	Read−Write	The horizontal alignment of the label widget in the range of 0.0 to 1.0
"label−yalign"	Read−Write	The vertical alignment of the decoration within the label widget height in the range of 0.0 to 1.0
"shadow−type"	Read−Write	The style of the frame's border; one of: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`
"label−widget"	Read−Write	The widget to display in place of the usual frame label.

Description

The `gtk.Frame` widget is a `gtk.Bin` that surrounds its child with a decorative frame and an optional label. If present, the label is drawn in a gap in the top side of the frame. The position of the label can be controlled with `set_label_align()`.

Constructor

```
gtk.Frame(label=None)
```

`label` :	a string to use as the label text or None if no label is required.
Returns :	a new frame widget

Creates a new `gtk.Frame` widget with the label text specified by *`label`*. If *`label`* is `None` no label is created.

Methods

gtk.Frame.set_label

```
def set_label(label)
```

`label` :	a string to be used as the label text

The `set_label`() method sets the text of the label as specified by *`label`*. If *`label`* is `None` the current label is removed.

gtk.Frame.get_label

```
def get_label()
```

Returns :	the text in the label, or `None` if there is no label widget or the label widget is not a `gtk.Label`.

The `get_label`() method returns the text in the label widget. If there is no label widget or the label widget is not a `gtk.Label` the method returns `None`.

gtk.Frame.set_label_widget

```
def set_label_widget(label_widget)
```

`label_widget` :	the new label widget

The `set_label_widget`() method set the label widget (usually to something other than a `gtk.Label` widget) for the frame. This widget will appear embedded in the top edge of the frame as a title.

gtk.Frame.get_label_widget

```
def get_label_widget()
```

Returns :	the label widget, or `None` if there is no label widget.

The `get_label_widget`() method retrieves the label widget for the frame. See `set_label_widget()`.

gtk.Frame.set_label_align

```
def set_label_align(xalign, yalign)
```

`xalign` :	the horizontal alignment of the label widget along the top edge of the frame (in the range of 0.0 to 1.0)
`yalign` :	the vertical alignment of the decoration with respect to the label widget (in the range 0.0 to 1.0)

The `set_label_align()` method sets the alignment of the frame's label widget and decoration (defaults are 0.0 and 0.5) as specified by *`xalign`* and *`yalign`*. The *`xalign`* value specifies the fraction of free horizontal space that is allocated to the left of the label widget. The *`yalign`* value specifies the fraction of label widget height above the decoration.

gtk.Frame.get_label_align

```
    def get_label_align()
```

Returns :	a tuple containing the x and y alignments of the frame's label widget

The `get_label_align()` method returns a tuple containing the X and Y alignment of the frame's label widget and decoration. See the `set_label_align()` method.

gtk.Frame.set_shadow_type

```
    def set_shadow_type(type)
```

`type` :	a shadow type: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`

The `set_shadow_type()` method sets the frame's shadow type to the value of *`type`*. The type must be one of: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`

gtk.Frame.get_shadow_type

```
    def get_shadow_type()
```

Returns :	the current shadow type of the frame.

The get_shadow_type() method retrieves the shadow type of the frame; one of: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`. See `set_shadow_type()`.

Prev	Up	Next
gtk.FontSelectionDialog	Home	gtk.GammaCurve

gtk.GammaCurve

Prev	**The gtk Class Reference**	Next

gtk.GammaCurve

gtk.GammaCurve — subclass of `gtk.Curve` for editing gamma curves.

Synopsis

```
class gtk.GammaCurve(gtk.VBox):
    gtk.GammaCurve()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
```

```
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Box
          +-- gtk.VBox
            +-- gtk.GammaCurve
```

Attributes

"table"	Read	The `gtk.Table` containing the `gtk.GammaCurve` widgets.
"curve"	Read	The `gtk.Curve` widget.
"gamma"	Read	The gamma value (float)
"gamma_dialog"	Read	The `gtk.Dialog` that prompts for the gamma value.
"gamma_text"	Read	The `gtk.Entry` containing the gamma value in the gamma dialog

Description

Note

This widget is considered too specialized or little–used for PyGTK, and will in the future be moved to some other package. If your application needs this widget, feel free to use it, as the widget does work and is useful in some applications; it's just not of general interest. However, it will eventually move out of the PyGTK distribution.

The `gtk.GammaCurve` widget is a variant of `gtk.Curve` specifically for editing gamma curves, which are used in graphics applications such as the Gimp. The `gtk.GammaCurve` widget shows a curve which the user can edit with the mouse just like a `gtk.Curve` widget. On the right of the curve it also displays 5 buttons, 3 of which change between the 3 curve modes (spline, linear and free), and the other 2 set the curve to a particular gamma value, or reset it to a straight line.

Constructor

```
  gtk.GammaCurve()
```

Returns :	a new gammacurve widget

Creates a new `gtk.GammaCurve` widget.

Prev Up Next
gtk.Frame Home gtk.GenericCellRenderer

gtk.HandleBox
Prev **The gtk Class Reference** Next

gtk.HandleBox

gtk.HandleBox a widget for detachable window portions.

Synopsis

```
class gtk.HandleBox(gtk.Bin):
    gtk.HandleBox()
    def set_shadow_type(type)
    def get_shadow_type()
    def set_handle_position(position)
    def get_handle_position()
    def set_snap_edge(edge)
    def get_snap_edge()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.HandleBox
```

Properties

"shadow–type"	Read–Write	The type of shadow; one of: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`
"handle–position"	Read–Write	The position of the handle relative to the child widget; one of: `gtk.POS_LEFT`, `gtk.POS_RIGHT`, `gtk.POS_TOP`, `gtk.POS_BOTTOM`
"snap–edge"	Read–Write	The side of the handlebox that's lined up with the docking point to dock the handlebox; one of: `gtk.POS_LEFT`, `gtk.POS_RIGHT`, `gtk.POS_TOP`, `gtk.POS_BOTTOM`
"snap–edge–set"	Read–Write	If `TRUE`, use the value from "snap–edge"; otherwise, use a value derived from "handle–position". Available in GTK+ 2.2 and above.

Signal Prototypes

"child–attached"	`def callback(handlebox, widget, user_param1, ...)`
"child–detached"	`def callback(handlebox, widget, user_param1, ...)`

Description

The `gtk.HandleBox` widget allows a portion of a window to be "torn off". It is a bin widget which displays its child and a handle that the user can drag to tear off into a separate floating window containing the child widget. A thin ghost is drawn in the original location of the handlebox. The separate window can be dragged back to its original location to be reattached. When reattaching, the ghost and float window, must be aligned along one of the edges called the snap edge that can be specified by the application, or specified automatically using a reasonable default based on the handle position. The snap edge is automatically set as `gtk.POS_TOP` if the handle position is `gtk.POS_RIGHT` or `gtk.POS_LEFT`; otherwise, the snap edge will be set as `gtk.POS_LEFT`.

To make detaching and reattaching the handlebox as minimally confusing as possible to the user, it is important to set the snap edge so that the snap edge does not move when the handlebox is detached. For example, if the handlebox is packed at the bottom of a `gtk.VBox`, then when the handlebox is detached, the bottom edge of the handlebox's allocation will remain fixed as the height of the handlebox shrinks, so the snap edge should be set to `gtk.POS_BOTTOM`.

Constructor

```
gtk.HandleBox()
```

Returns :	a new handlebox widget

Creates a new `gtk.HandleBox` widget.

Methods

gtk.HandleBox.set_shadow_type

```
def set_shadow_type(type)
```

type :	the shadow type: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`

The `set_shadow_type()` method sets the type of shadow to be drawn around the border of the handle box as specified by *`type`*. The value of *`type`* must be one of: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`.

gtk.HandleBox.get_shadow_type

```
def get_shadow_type()
```

Returns :	the type of shadow currently drawn around the handle box.

The `get_shadow_type()` method gets the type of shadow drawn around the handle box. The shadow type is one of: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`. See `set_shadow_type()`.

gtk.HandleBox.set_handle_position

```
def set_handle_position(position)
```

position :	the side of the handlebox where the handle should be drawn.

The `set_handle_position()` method sets the side of the handlebox where the handle is drawn using the value of *`position`*. The value of *`position`* must be one of: `gtk.POS_LEFT`, `gtk.POS_RIGHT`, `gtk.POS_TOP`, `gtk.POS_BOTTOM`

gtk.HandleBox.get_handle_position

```
def get_handle_position()
```

Returns :	the current handle position.

The `get_handle_position()` method gets the handle position of the handle box; one of: `gtk.POS_LEFT`, `gtk.POS_RIGHT`, `gtk.POS_TOP`, `gtk.POS_BOTTOM`. See `set_handle_position()`.

gtk.HandleBox.set_snap_edge

```
    def set_snap_edge(edge)
```

edge :	the edge to use as the snap edge or −1 to have PyGTK automatically pick the snap edge

The `set_snap_edge()` method sets the snap edge of the handlebox to the value specified by *`edge`*. The value of *`edge`* can be one of: `gtk.POS_LEFT`, `gtk.POS_RIGHT`, `gtk.POS_TOP`, `gtk.POS_BOTTOM` or −1 to have the snap edge automatically specified.

The snap edge is the edge of the detached child that must be aligned with the corresponding edge of the "ghost" left behind when the child was detached to reattach the torn−off window. Usually, the snap edge should be chosen so that it stays in the same place on the screen when the handlebox is torn off. If the snap edge is not set, then an appropriate value will be guessed from the handle position. If the handle position is `gtk.POS_RIGHT` or `gtk.POS_LEFT`, then the snap edge will be `gtk.POS_TOP`, otherwise it will be `gtk.POS_LEFT`.

gtk.HandleBox.get_snap_edge

```
    def get_snap_edge()
```

Returns :	the edge used for determining reattachment, or −1 if the snap edge is determined (as per default) from the handle position.

The `get_snap_edge()` method gets the edge used for determining reattachment of the handle box. The return value will be one of: `gtk.POS_LEFT`, `gtk.POS_RIGHT`, `gtk.POS_TOP`, `gtk.POS_BOTTOM` or −1 to indicate the snap edge is automatically selected. See `set_snap_edge()`.

Signals

The "child−attached" gtk.HandleBox Signal

```
    def callback(handlebox, widget, user_param1, ...)
```

`handlebox` :	the widget that received the signal
`widget` :	the child widget
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "child−attached" signal is emitted when the contents of the handlebox are reattached to the main window.

The "child−detached" gtk.HandleBox Signal

```
    def callback(handlebox, widget, user_param1, ...)
```

`handlebox` :	the widget that received the signal
`widget` :	the child widget
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "child−detached" signal is emitted when the contents of the handlebox are detached from the main window.

gtk.HBox

gtk.HBox — a horizontal container box

Synopsis

```
class gtk.HBox(gtk.Box):
    gtk.HBox(homogeneous=FALSE, spacing=0)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Box
          +-- gtk.HBox
```

Description

The `gtk.HBox` is a container subclassed from `gtk.Box` that organizes its child widgets into a single horizontal row. The `gtk.Box` methods are used to manage the order, spacing, width and alignment of the child widgets though all widgets are allocated the same height.

Constructor

```
   gtk.HBox(homogeneous=FALSE, spacing=0)
```

homogeneous :	If `TRUE` all children are given equal space allocations.
spacing :	The additional horizontal space between children in pixels
Returns :	a new hbox widget

Creates a new `gtk.HBox` widget.

gtk.HButtonBox

gtk.HButtonBox — a container for arranging buttons horizontally.

Synopsis

```
class gtk.HButtonBox(gtk.ButtonBox):
    gtk.HButtonBox()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Box
          +-- gtk.ButtonBox
            +-- gtk.HButtonBox
```

Description

The `gtk.HButtonBox` is a container subclassed from `gtk.ButtonBox` that is optimized for the horizontal layout of buttons. The `gtk.HButtonBox` helps provide a consistent layout of buttons in an application by providing default values of spacing, padding and layout style as described in the `gtk.ButtonBox` reference. Buttons are packed into a `gtk.HButtonBox` using the `gtk.Container.add()` method or the `gtk.Box.pack_start()` and `gtk.Box.pack_end()` methods. The spacing between buttons can be set with the `gtk.Box.set_spacing()` method.

Constructor

```
  gtk.HButtonBox()
```

Returns :	a new hbuttonbox widget

Creates a new `gtk.HButtonBox` widget.

Prev	Up	Next
gtk.HBox	Home	gtk.HPaned

gtk.HPaned

Prev **The gtk Class Reference** Next

gtk.HPaned

gtk.HPaned a container with two panes arranged horizontally.

Synopsis

```
class gtk.HPaned(gtk.Paned):
    gtk.HPaned()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
```

```
        +-- gtk.Paned
          +-- gtk.HPaned
```

Description

The `gtk.HPaned` is a container widget subclassed from `gtk.Paned` with two children arranged horizontally. The division between the children can be adjusted by the user by dragging a handle. See the `gtk.Paned` description for more information.

Constructor

```
    gtk.HPaned()
```

Returns :	a new hpaned widget

Creates a new `gtk.HPaned` widget.

gtk.HRuler

gtk.HRuler — a horizontal ruler.

Synopsis

```
class gtk.HRuler(gtk.Ruler):
    gtk.HRuler()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Ruler
        +-- gtk.HRuler
```

Description

Note

This widget is considered too specialized or little–used for PyGTK, and will in the future be moved to some other package. If your application needs this widget, feel free to use it, as the widget does work and is useful in some applications; it's just not of general interest. However, it will eventually move out of the PyGTK distribution.

The `gtk.HRuler` widget is arranged horizontally creating a ruler that is used in conjunction with other widgets such as a text widget. The ruler is used to show the location of the mouse on the window and to show the size of the window in specified units. The available units of measurement are `gtk.PIXELS` (the default), gtk.INCHES and gtk.CENTIMETERS. See the gtk.Ruler description for more information on the methods that are used to manage a gtk.HRuler.

Constructor

```
    gtk.HRuler()
```

Returns :	a new hruler widget

Creates a new gtk.HRuler widget.

gtk.HScale

gtk.HScale — a horizontal slider widget for selecting a value from a range.

Synopsis

```
class gtk.HScale(gtk.Scale):
    gtk.HScale(adjustment=None)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Range
        +-- gtk.Scale
          +-- gtk.HScale
```

Description

The gtk.HScale is subclassed from gtk.Scale to provide a widget that allows a user to select a value using a horizontal slider. See the gtk.Scale description for more information on the methods available to manage a gtk.HScale.

Constructor

```
    gtk.HScale(adjustment=None)
```

adjustment :	a gtk.Adjustment object
Returns :	a new hscale widget

Creates a new gtk.HScale widget and associates a gtk.Adjustment specified by *adjustment*. The default value of *adjustment* is None which creates the hscale with no gtk.Adjustment.

gtk.HScrollbar

gtk.HScrollbar — a horizontal scrollbar widget

Synopsis

```
class gtk.HScrollbar(gtk.Scrollbar):
    gtk.HScrollbar(adjustment=None)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Range
        +-- gtk.Scrollbar
          +-- gtk.HScrollbar
```

Description

The `gtk.HScrollbar` widget is subclassed from `gtk.Scrollbar` to provide a horizontal scrollbar. See `gtk.Scrollbar` for details on the methods available for managing scrollbars. A `gtk.Adjustment` may be specified with the scrollbar at creation (or is created automatically if none is specified) to handle the adjustment of the scrollbar. See `gtk.Adjustment` for details.

Constructor

```
    gtk.HScrollbar(adjustment=None)
```

`adjustment` :	a `gtk.Adjustment` or `None` to automatically create an adjustment
Returns :	a new hscrollbar widget

Creates a new `gtk.HScrollbar` with an associated `gtk.Adjustment` specified by *`adjustment`*. If *`adjustment`* is None or missing a new `gtk.Adjustment` will be created and associated with the scrollbar.

gtk.HSeparator

gtk.HSeparator — a horizontal separator.

Synopsis

```
class gtk.HSeparator(gtk.Separator):
    gtk.HSeparator()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Separator
        +-- gtk.HSeparator
```

Description

The `gtk.HSeparator` widget is a horizontal separator, used to visibly separate the widgets within a window. It displays a horizontal line with a shadow to make it appear sunken into the interface.

Note

The `gtk.HSeparator` widget is not used as a separator within menus. To create a separator in a menu create an empty `gtk.SeparatorMenuItem` widget and add it to the menu with `gtk.MenuShell.append()`.

Constructor

```
   gtk.HSeparator()
```

Returns :	a new horizontal separator widget

Creates a new `gtk.HSeparator` widget.

gtk.IconFactory

gtk.IconFactory — an object that manages a group of icon sets.

Synopsis

```
class gtk.IconFactory(gobject.GObject):
    gtk.IconFactory()
    def add(stock_id, icon_set)
    def lookup(stock_id)
    def add_default()
    def remove_default()
Functions

    def gtk.icon_factory_lookup_default(stock_id)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.IconFactory
```

Description

A gtk.IconFactory manages a group of gtk.IconSets that manage a set of variants (for different sizes and states) of a specific icon. Icons in an icon factory are named by a stock ID, which is a simple string identifying the icon e.g "gtk–ok". Each gtk.Style has a list of gtk.IconFactorys derived from the current theme; those icon factories are consulted first when searching for an icon. If the theme doesn't set a particular icon, the search continues for the icon in a list of default icon factories, maintained by the add_default()() and remove_default()() methods. Applications with icons should add a default icon factory with their icons, which will allow themes to override the icons for the application.

To display an icon, always use gtk.Style.lookup_icon_set() on the widget that will display the icon, or the convenience function gtk.Widget.render_icon(). These functions take the theme into account when looking up the icon to use for a given stock ID.

Constructor

```
gtk.IconFactory()
```

Returns :	a new gtk.IconFactory object

Creates a new gtk.IconFactory.

Methods

gtk.IconFactory.add

```
def add(stock_id, icon_set)
```

stock_id :	an icon name
icon_set :	an icon set

The add() method adds the specified *icon_set* to the icon factory, under the name *stock_id*. the *stock_id* string should include the name of your application, e.g. "myapp–whatever–icon". Normally applications create a gtk.IconFactory, then add it to the list of default factories with the add_default(). Then they pass the *stock_id* to widgets such as gtk.Image to display the icon. Themes can provide an icon with the same name (such as "myapp–whatever–icon") to override your application's default icons. If an icon already existed in the icon factory for *stock_id*, it is unreferenced and replaced with the new *icon_set*.

gtk.IconFactory.lookup

```
def lookup(stock_id)
```

stock_id :	an icon name
Returns :	the icon set named *stock_id*.

The lookup() method looks up *stock_id* in the icon factory, returning an icon set if found, otherwise None. For display to the user, you should use gtk.Style.lookup_icon_set() on the gtk.Style for the widget that will display the icon, instead of using this function directly, so that themes are taken into account.

gtk.IconFactory.add_default

```
    def add_default()
```

The `add_default`() method adds the icon factory to the list of icon factories searched by `gtk.Style.lookup_icon_set()`. This means that, for example, `gtk.Image.set_from_stock()` will be able to find icons in the icon factory. There will normally be an icon factory added for each library or application that comes with icons. The default icon factories can be overridden by themes.

gtk.IconFactory.remove_default

```
    def remove_default()
```

The `remove_default`() method removes an icon factory from the list of default icon factories. Not normally used though you might use it for a library that can be unloaded or shut down.

Functions

gtk.icon_factory_lookup_default

```
    def gtk.icon_factory_lookup_default(stock_id)
```

`stock_id` :	the stock ID to lookup
Returns :	the icon set matching *`stock_id`* or `None` if no icon set matches

The `gtk.icon_factory_lookup_default`() function returns the `gtk.IconSet` that has the name specified by *`stock_id`*. If the icon set cannot be found the function returns `None`.

Prev	Up	Next
gtk.HSeparator	Home	gtk.IconInfo

gtk.IconInfo

gtk.IconInfo — object containing information about and icon in an icon theme (new in PyGTK 2.4)

Synopsis

```
class gtk.IconInfo(gobject.GBoxed):
    def copy()
    def free()
    def get_base_size()
    def get_filename()
    def get_builtin_pixbuf()
    def load_icon()
    def set_raw_coordinates(raw_coordinates)
    def get_embedded_rect()
    def get_attach_points()
    def get_display_name()
```

Description

Note

This object is available in PyGTK 2.4 and above.

A `gtk.IconInfo` object contains information about an icon in a `gtk.IconTheme`. A `gtk.IconInfo` object is created using the `gtk.IconTheme.lookup_icon()` method.

A `gtk.gdk.Pixbuf` can be rendered with the icon using the `load_icon()` method. If you just want to load the pixbuf of an icon you can use the `gtk.IconTheme.load_icon()` method that combines the `gtk.IconTheme.lookup_icon()` method and the `load_icon()` method.

Methods

gtk.IconInfo.copy

```
    def copy()
```

Returns :	the new `gtk.IconInfo`

Note

This method is available in PyGTK 2.4 and above.

The `copy()` method returns a copy of the icon info object.

gtk.IconInfo.free

```
    def free()
```

Note

This method is available in PyGTK 2.4 and above.

The `free()` method frees the icon info and its associated information

gtk.IconInfo.get_base_size

```
    def get_base_size()
```

Returns :	the base size, or 0, if no base size is known for the icon.

Note

This method is available in PyGTK 2.4 and above.

The `get_base_size()` method returns the base size for the icon. The base size is a size for the icon that was specified by the icon theme creator. This may be different than the actual size of image; an example of this is small emblem icons that can be attached to a larger icon. These icons will be given the same base size as the larger icons to which they are attached.

gtk.IconInfo.get_filename

```
def get_filename()
```

Returns : the filename for the icon, or `None` if the `get_builtin_pixbuf()` should be used instead.

Note

This method is available in PyGTK 2.4 and above.

The `get_filename`() method returns the filename for the icon. If the `gtk.ICON_LOOKUP_USE_BUILTIN` flag was passed to the `gtk.IconTheme.lookup_icon()` method, there may be no filename if a builtin icon is returned. In this case, you should use the `get_builtin_pixbuf()` method.

gtk.IconInfo.get_builtin_pixbuf

```
def get_builtin_pixbuf()
```

Returns : the built–in image pixbuf, or `None`.

Note

This method is available in PyGTK 2.4 and above.

The `get_builtin_pixbuf`() method returns the built–in image for this icon, if any. To allow `GTK+` to use built in icon images, you must pass the `gtk.ICON_LOOKUP_USE_BUILTIN` to the `gtk.IconTheme.lookup_icon()` method.

gtk.IconInfo.load_icon

```
def load_icon()
```

Returns : the icon rendered into a `gtk.gdk.Pixbuf`. This may be a newly created icon or a new reference to an internal icon, so you must not modify the icon.

Note

This method is available in PyGTK 2.4 and above.

The `load_icon`() method renders the icon previously looked up in an icon theme using the `gtk.IconTheme.lookup_icon()` method. The icon size will be based on the size passed to the `gtk.IconTheme.lookup_icon()` method. Note that the resulting `gtk.gdk.Pixbuf` may not be exactly this size. An icon theme may have icons that differ slightly from their nominal sizes, and in addition `GTK+` will avoid scaling icons that it considers sufficiently close to the requested size to maintain sharpness.

This method raise the GError exception if an error occurs during rendering of the icon.

gtk.IconInfo.set_raw_coordinates

```
def set_raw_coordinates(raw_coordinates)
```

`raw_coordinates` :

	if `TRUE`, the coordinates of embedded rectangles and attached points should be returned in their original (unscaled) form.

Note

This method is available in PyGTK 2.4 and above.

The `set_raw_coordinates`() method sets the internal raw_coordinates flag to the value of *`raw_coordinates`*. If *`raw_coordinates`* is `TRUE`, the coordinates returned by the `get_embedded_rect()` and `get_attach_points()` methods will be returned in their original form as specified in the icon theme, instead of scaled appropriately for the pixbuf returned by the `load_icon()` method.

Raw coordinates are somewhat strange; they are specified to be with respect to the unscaled pixmap for `PNG` and `XPM` icons, but for `SVG` icons, they are in a 1000x1000 coordinate space that is scaled to the final size of the icon. You can determine if the icon is an `SVG` icon by using the `get_filename()` method, and seeing if it is not `None` and ends in '.svg'.

This method is provided primarily to allow compatibility wrappers for older API's, and is not expected to be useful for applications.

gtk.IconInfo.get_embedded_rect

```
    def get_embedded_rect()
```

Returns :	a `gtk.gdk.Rectangle` or `None`

Note

This method is available in PyGTK 2.4 and above.

The `get_embedded_rect`() method returns a `gtk.gdk.Rectangle` containing the coordinates of a rectangle within the icon that can be used for display of information such as a preview of the contents of a text file. See the `set_raw_coordinates()` method for further information about the coordinate system.

gtk.IconInfo.get_attach_points

```
    def get_attach_points()
```

Returns :	a tuple containing a set of 2−tuples for the attach points

Note

This method is available in PyGTK 2.4 and above.

The `get_attach_points`() method returns a tuple containing the attach points for an icon as a set of 2−tuples. An attach point is a location in the icon that can be used as anchor points for attaching emblems or overlays to the icon.

gtk.IconInfo.get_display_name

```
    def get_display_name()
```

Returns :	the display name for the icon or `None`, if the icon doesn't have a specified display name.

Note

This method is available in PyGTK 2.4 and above.

The `get_display_name()` method returns the display name for an icon. A display name is a string to be used in place of the icon name in a user visible context like a list of icons.

Prev	Up	Next
gtk.IconFactory	Home	gtk.IconSet

gtk.IconSet

Prev **The gtk Class Reference** Next

gtk.IconSet

gtk.IconSet — contains a set of variants for an icon

Synopsis

```
class gtk.IconSet(gobject.GBoxed):
    gtk.IconSet(pixbuf=None)
    def copy()
    def render_icon(style, direction, state, size, widget, detail)
    def add_source(source)
    def get_sizes()
```

Description

A `gtk.IconSet` contains a set of variants for a single icon. The variants provide icons of different sizes and for different widget states. The variants can be added with the `add_source()` method.

Constructor

```
    gtk.IconSet(pixbuf=None)
```

`pixbuf` :	a `gtk.gdk.Pixbuf`
Returns :	a new `gtk.IconSet`

Creates a new `gtk.IconSet` with *`pixbuf`* as the default fallback source image. If *`pixbuf`* is `None` there is no default fallback source image. If you don't add any additional `gtk.IconSource`s to the icon set, all variants of the icon will be created from *`pixbuf`*, using scaling, pixelation, etc. as required to adjust the icon size or make the icon look insensitive/prelighted.

Methods

gtk.IconSet.copy

```
    def copy()
```

Returns :	a new `gtk.IconSet` identical to the first.

The `copy()` method returns a copy of the icon set.

gtk.IconSet.render_icon

```
    def render_icon(style, direction, state, size, widget, detail)
```

style :	a gtk.Style associated with *widget*, or None
direction :	the text direction; one of: gtk.TEXT_DIR_NONE, gtk.TEXT_DIR_LTR, gtk.TEXT_DIR_RTL
state :	the widget state; one of: gtk.STATE_NORMAL, gtk.STATE_ACTIVE, gtk.STATE_PRELIGHT, gtk.STATE_SELECTED, gtk.STATE_INSENSITIVE
size :	the icon size; one of: gtk.ICON_SIZE_INVALID, gtk.ICON_SIZE_MENU, gtk.ICON_SIZE_SMALL_TOOLBAR, gtk.ICON_SIZE_LARGE_TOOLBAR, gtk.ICON_SIZE_BUTTON, gtk.ICON_SIZE_DND, gtk.ICON_SIZE_DIALOG
widget :	the widget that will display the icon, or None
detail :	the detail to pass to the theme engine, or None
Returns :	a gtk.gdk.Pixbuf to be displayed

The render_icon() method renders an icon using gtk.Style.render_icon(). In most cases, gtk.Widget.render_icon() is better, since it automatically provides most of the arguments from the current widget settings. This method never returns None; if the icon can't be rendered (perhaps because an image file fails to load), a default "missing image" icon will be returned instead.

gtk.IconSet.add_source

```
    def add_source(source)
```

source :	a gtk.IconSource

The add_source() method adds the gtk.IconSource specified by *source* to the icon set. Icon sets have a list of gtk.IconSource, which they use as base icons for rendering icons in different states and sizes. Icons are scaled, made to look insensitive, etc. in the render_icon() method, but gtk.IconSet needs base images to work with. The base images and when to use them are described by a gtk.IconSource.

An example of when you'd use this method: a web browser's "Back to Previous Page" icon might point in a different direction in Hebrew and in English; it might look different when insensitive; and it might change size depending on toolbar mode (small or large icons). So a single icon set would contain all those variants of the icon, and you might add a separate source for each one.

You should nearly always add a "default" icon source with all fields wildcarded, which will be used as a fallback if no more specific source matches. gtk.IconSet always prefers more specific icon sources to more generic icon sources. The order in which you add the sources to the icon set does not matter.

The gtk.IconSet() constructor creates a new icon set with a default icon source based on the given pixbuf.

gtk.IconSet.get_sizes

```
    def get_sizes()
```

Returns :	a tuple containing all the icon sizes supported by the icon set

The get_sizes() method returns a tuple containing all the icon sizes this icon set can render.

gtk.IconSource

gtk.IconSource — a source for icon variants

Synopsis

```
class gtk.IconSource(gobject.GBoxed):
    gtk.IconSource()
    def copy()
    def free()
    def set_filename(filename)
    def set_pixbuf(pixbuf)
    def get_filename()
    def get_pixbuf()
    def set_direction_wildcarded(setting)
    def set_state_wildcarded(setting)
    def set_size_wildcarded(setting)
    def get_size_wildcarded()
    def get_state_wildcarded()
    def get_direction_wildcarded()
    def set_direction(direction)
    def set_state(state)
    def set_size(size)
    def get_direction()
    def get_state()
    def get_size()
Functions

    def gtk.icon_size_lookup(icon_size)
    def gtk.icon_size_lookup_for_settings(settings, icon_size)
    def gtk.icon_size_register(name, width, height)
    def gtk.icon_size_register_alias(alias, target)
    def gtk.icon_size_from_name(name)
    def gtk.icon_size_get_name(size)
```

Description

A `gtk.IconSource` contains a `gtk.gdk.Pixbuf` (or an image filename) that serves as the base image for one or more of the icons in a `gtk.IconSet`, along with a specification for which icons in the icon set will be based on that pixbuf or image file. By default, the icon source has all parameters wildcarded. That is, the icon source can be used as the base icon for any desired text direction, widget state, or icon size.

Constructor

```
    gtk.IconSource()
```

Returns :	a new `gtk.IconSource`

Creates a new `gtk.IconSource`. A `gtk.IconSource` contains a `gtk.gdk.Pixbuf` (or image filename) that serves as the base image for one or more of the icons in a `gtk.IconSet`, along with a specification for which icons in the icon set will be based on that pixbuf or image file. By default, the icon source has all parameters wildcarded. That is, the icon source will be used as the base icon for any desired text direction, widget state, or icon size.

Methods

gtk.IconSource.copy

```
def copy()
```

Returns :	a new gtk.IconSource

The copy() method creates a copy of the icon source.

gtk.IconSource.free

```
def free()
```

The free() method frees a dynamically−allocated icon source, along with its filename, size, and pixbuf fields if those are not None.

gtk.IconSource.set_filename

```
def set_filename(filename)
```

filename :	the image file to use

The set_filename() method sets the name of the image file (specified by *filename*) to use as the base image when creating icon variants for a gtk.IconSet. The filename must be absolute.

gtk.IconSource.set_pixbuf

```
def set_pixbuf(pixbuf)
```

pixbuf :	the pixbuf to use as a source

The set_pixbuf() method sets a pixbuf (specified by *pixbuf*) to use as a base image when creating icon variants for a gtk.IconSet. If an icon source has both a filename and a pixbuf set, the pixbuf will take priority.

gtk.IconSource.get_filename

```
def get_filename()
```

Returns :	the image filename

The get_filename() method retrieves the source filename, or None if none is set.

gtk.IconSource.get_pixbuf

```
def get_pixbuf()
```

Returns :	the source pixbuf

The get_pixbuf() method retrieves the source pixbuf, or None if none is set.

gtk.IconSource.set_direction_wildcarded

```
    def set_direction_wildcarded(setting)
```

`setting` :	if `TRUE` wildcard the text direction

The `set_direction_wildcarded()` method determines whether the icon source direction is wildcarded according to the value specified by *`setting`*. If *`setting`* is `TRUE` the text direction is wildcarded and the icon source can be used as the base image for an icon in any text direction (`gtk.TEXT_DIR_NONE`, `gtk.TEXT_DIR_LTR` or `gtk.TEXT_DIR_RTL`). If the text direction is not wildcarded, then the text direction the icon source applies to should be set with the `set_direction()` method and the icon source will only be used with that text direction. Non−wildcarded icon sources (exact matches) are preferred over wildcarded icon sources. An exact match will be used when possible.

gtk.IconSource.set_state_wildcarded

```
    def set_state_wildcarded(setting)
```

`setting` :	if `TRUE` wildcard the widget state

The `set_state_wildcarded()` method determines whether the icon source state is wildcarded according to the value of *`setting`*. If *`setting`* is `TRUE` the widget state is wildcarded and the icon source can be used as the base image for an icon in any widget state (`gtk.STATE_NORMAL`, `gtk.STATE_ACTIVE`, `gtk.STATE_PRELIGHT`, `gtk.STATE_SELECTED` or `gtk.STATE_INSENSITIVE`). If the widget state is not wildcarded, then the widget state the icon source applies to should be set with the `set_state()` method and the icon source will only be used with that specific state. Non−wildcarded icon sources (exact matches) are preferred over wildcarded icon sources. An exact match will be used when possible.

A `gtk.IconSet` will normally transform wildcarded icon source images to produce an appropriate icon for a given state, for example lightening an image on prelight, but will not modify source images that match exactly.

gtk.IconSource.set_size_wildcarded

```
    def set_size_wildcarded(setting)
```

`setting` :	if `TRUE` wildcard the widget state

The `set_size_wildcarded()` determines whether the icon source can be used as the basis for an icon of any size according to the value of *`setting`*. If *`setting`* is `TRUE` the icon size is wildcarded and the icon source can be used as the base image for an icon of any size. If the size is not wildcarded, then the size the icon source applies to should be set with `gtk.IconSource.set_size()` and the icon source will only be used with that specific size. Non−wildcarded icon sources (exact matches) are preferred over wildcarded icon sources. An exact match will be used when possible.

`gtk.IconSet` will normally scale wildcarded source images to produce an appropriate icon at a given size, but will not change the size of source images that match exactly.

gtk.IconSource.get_size_wildcarded

```
    def get_size_wildcarded()
```

Returns :	`TRUE` if this icon source is a base for any icon size variant

The `get_size_wildcarded()` method gets the value set by the `set_size_wildcarded()` method.

gtk.IconSource.get_state_wildcarded

```
    def get_state_wildcarded()
```

Returns :	`TRUE` if this icon source is a base for any widget state variant

The `get_state_wildcarded()` method gets the value set by the `set_state_wildcarded()` method.

gtk.IconSource.get_direction_wildcarded

```
    def get_direction_wildcarded()
```

Returns :	`TRUE` if this icon source is a base for any text direction variant

The `get_direction_wildcarded()` method gets the value set by the `set_direction_wildcarded()` method.

gtk.IconSource.set_direction

```
    def set_direction(direction)
```

`direction` :	the text direction this icon source applies to

The `set_direction()` method sets the text direction according to the value of *`direction`* that the icon source is intended to be used with. The value of direction must be one of: `gtk.TEXT_DIR_NONE`, `gtk.TEXT_DIR_LTR` or `gtk.TEXT_DIR_RTL`.

Setting the text direction on an icon source makes no difference if the text direction is wildcarded. Therefore, you should usually call the `set_direction_wildcarded()` method with a setting of `FALSE` to un−wildcard it in addition to calling this function.

gtk.IconSource.set_state

```
    def set_state(state)
```

`state` :	the widget state this source applies to

The `set_state()` method sets the widget state specified by *`state`* that the icon source is intended to be used with. The value of *`state`* must be one of: `gtk.STATE_NORMAL`, `gtk.STATE_ACTIVE`, `gtk.STATE_PRELIGHT`, `gtk.STATE_SELECTED` or `gtk.STATE_INSENSITIVE`. Setting the widget state on an icon source makes no difference if the state is wildcarded. Therefore, you should usually call the `set_state_wildcarded()` method with a setting of `FALSE` to un−wildcard it in addition to calling this function.

gtk.IconSource.set_size

```
    def set_size(size)
```

`size` :	the icon size this source applies to

The `set_size()` method sets the icon size specified by *`size`* that the icon source is intended to be used with. Setting the icon size for an icon source makes no difference if the size is wildcarded. Therefore, you should usually call the `gtk.IconSource.set_size_wildcarded()` method with a setting of `FALSE` to un−wildcard it in addition to calling this function.

gtk.IconSource.get_direction

```
def get_direction()
```

Returns : the text direction the icon source matches

The `get_direction()` method obtains the text direction this icon source applies to. The return value is only useful and meaningful if the text direction is *not* wildcarded.

gtk.IconSource.get_state

```
def get_state()
```

Returns : the widget state the icon source matches

The `get_state()` method obtains the widget state this icon source applies to. The return value is only useful and meaningful if the widget state is *not* wildcarded.

gtk.IconSource.get_size

```
def get_size()
```

Returns : the icon size this source matches.

The `get_size()` method obtains the icon size this source applies to. The return value is only useful and meaningful if the icon size is *not* wildcarded.

Functions

gtk.icon_size_lookup

```
def gtk.icon_size_lookup(icon_size)
```

`icon_size` : an icon size

Returns : a 2–tuple containing the width and height of the specified *`icon_size`*

The `gtk.icon_size_lookup()` function returns a 2–tuple containing the width and height of the icon size specified by *`icon_size`*. The value of *`icon_size`* must be one of:

- `gtk.ICON_SIZE_MENU`
- `gtk.ICON_SIZE_SMALL_TOOLBAR`
- `gtk.ICON_SIZE_LARGE_TOOLBAR`
- `gtk.ICON_SIZE_BUTTON`
- `gtk.ICON_SIZE_DND`
- `gtk.ICON_SIZE_DIALOG`

or an integer value returned from the `gtk.icon_size_register()` function.

gtk.icon_size_lookup_for_settings

```
def gtk.icon_size_lookup_for_settings(settings, icon_size)
```

`settings` : a `gtk.Settings` object used to determine which user preferences to use.

`icon_size` : an icon size

Returns :

	a 2–tuple containing the width and height of the specified *`icon_size`* or `None` if *`icon_size`* was invalid.

Note

This function is available in PyGTK 2.4 and above.

The `gtk.icon_size_lookup_for_settings()` function returns a 2–tuple containing the width and height of the icon size specified by *`icon_size`* of `None` if *`icon_size`* was not valid. The value of *`icon_size`* must be one of:

- `gtk.ICON_SIZE_MENU`
- `gtk.ICON_SIZE_SMALL_TOOLBAR`
- `gtk.ICON_SIZE_LARGE_TOOLBAR`
- `gtk.ICON_SIZE_BUTTON`
- `gtk.ICON_SIZE_DND`
- `gtk.ICON_SIZE_DIALOG`

or an integer value returned from the `gtk.icon_size_register()` function.

gtk.icon_size_register

```
    def gtk.icon_size_register(name, width, height)
```

`name` :	the name of the icon size
`width` :	the width of the icon size
`height` :	the height of the icon size
Returns :	an integer representing the icon size

The `gtk.icon_size_register()` function registers a new icon size for the specified *`width`* and *`height`* with the specified *`name`* and returns the integer used to represent the icon size.

gtk.icon_size_register_alias

```
    def gtk.icon_size_register_alias(alias, target)
```

`alias` :	an alias for *`target`*
`target` :	an integer representing an existing icon size

The `gtk.icon_size_register_alias()` function registers the specified *`alias`* as another name for the icon size specified by *`target`*.

gtk.icon_size_from_name

```
    def gtk.icon_size_from_name(name)
```

`name` :	a name of an existing icon size
Returns :	the icon size associated with *`name`*

The `gtk.icon_size_from_name()` function returns the integer representing the icon size associated with the specified *`name`*.

gtk.icon_size_get_name

```
    def gtk.icon_size_get_name(size)
```

size :	an integer representing an existing icon size
Returns :	the name associated with the icon size represented by *size*

The `gtk.icon_size_get_name()` function returns the name of the icon size represented by *size*.

gtk.IconTheme

gtk.IconTheme — look up icons by name and size (new in PyGTK 2.4)

Synopsis

```
class gtk.IconTheme(gobject.GObject):
    gtk.IconTheme()
    def set_screen(screen)
    def set_search_path(path)
    def get_search_path()
    def append_search_path(path)
    def prepend_search_path(path)
    def set_custom_theme(theme_name)
    def has_icon(icon_name)
    def lookup_icon(icon_name, size, flags)
    def load_icon(icon_name, size, flags)
    def list_icons(context=None)
    def get_example_icon_name()
    def rescan_if_needed()
    def get_icon_sizes(icon_name)
Functions

    def gtk.icon_theme_get_default()
    def gtk.icon_theme_get_for_screen(screen)
    def gtk.icon_theme_add_builtin_icon(icon_name, size, pixbuf)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.IconTheme
```

Signal Prototypes

"changed"	`def callback(icontheme, user_param1, ...)`

Description

Note

This object is available in PyGTK 2.4 and above.

`gtk.IconTheme` provides a facility for looking up icons by name and size. The main reason for using a name rather than simply providing a filename is to allow different icons to be used depending on what icon theme is selected by the user. The operation of icon themes on Linux and Unix follows the Icon Theme Specification). There is a default icon theme, named "hicolor" where applications should install their icons, but more additional application themes can be installed as operating system vendors and users choose.

Named icons are similar to the Themeable Stock Images facility (see `gtk.IconFactory`, `gtk.IconSet` and `gtk.IconSource` for more detail), and the distinction between the two may be a bit confusing. A few things to keep in mind:

- Stock images usually are used in conjunction with Stock Items, such as `gtk.STOCK_OK` or `gtk.STOCK_OPEN`. Named icons are easier to set up and therefore are more useful for new icons that an application wants to add, such as application icons or window icons.
- Stock images can only be loaded at the symbolic sizes defined by the standard icon sizes (see the `gtk.icon_size_lookup()` function), or by custom sizes defined by the `gtk.icon_size_register()` function, while named icons are more flexible and any pixel size can be specified.
- Because stock images are closely tied to stock items, and thus to actions in the user interface, stock images may come in multiple variants for different widget states or writing directions.

A good rule of thumb is that if there is a stock image for what you want to use, use it, otherwise use a named icon. It turns out that internally stock images are generally defined in terms of one or more named icons. (An example is icons that depend on writing direction; `gtk.STOCK_GO_FORWARD` uses the two themed icons "gtk−stock−go−forward−ltr" and "gtk−stock−go−forward−rtl".)

In many cases, named themes are used indirectly, via `gtk.Image` or stock items, rather than directly, but looking up icons directly is also simple. The `gtk.IconTheme` object acts as a database of all the icons in the current theme. You can create new `gtk.IconTheme` objects, but its much more efficient to use the standard icon theme for the `gtk.gdk.Screen` so that the icon information is shared with other people looking up icons. In the case where the default screen is being used, looking up an icon can be as simple as:

```
icon_theme = gtk.icon_theme_get_default()
try:
    pixbuf = icon_theme.load_icon("my-icon-name", 48, 0)
except gobject.GError, exc:
    print "can't load icon", exc
```

Constructor

```
  gtk.IconTheme()
```

Returns :	the newly created `gtk.IconTheme` object.

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new icon theme object. Icon theme objects are used to lookup up an icon by name in a particular icon theme. Usually, you'll want to use the `gtk.icon_theme_get_default()` or `gtk.icon_theme_get_for_screen()` functions rather than creating a new icon theme object from scratch.

Methods

gtk.IconTheme.set_screen

```
    def set_screen(screen)
```

screen :	a gtk.gdk.Screen

Note

This method is available in PyGTK 2.4 and above.

The `set_screen()` method sets the `gtk.gdk.Screen` for an icon theme to *screen*. The screen is used to track the user's currently configured icon theme, which might be different for different screens.

gtk.IconTheme.set_search_path

```
    def set_search_path(path)
```

path :	a list or tuple of directories that are searched for icon themes

Note

This method is available in PyGTK 2.4 and above.

The `set_search_path()` method sets the search path for the icon theme object to the list of directory names contained in *path*. When looking for an icon theme, `GTK+` will search for a subdirectory of one or more of the directories in *path* with the same name as the icon theme. (Themes from multiple of the path elements are combined to allow themes to be extended by adding icons in the user's home directory.)

In addition if an icon found isn't found either in the current icon theme or the default icon theme, and an image file with the right name is found directly in one of the elements of *path*, then that image will be used for the icon name. (This is a legacy feature, and new icons should be put into the default icon theme, which is called "hicolor", rather than directly on the icon path.)

gtk.IconTheme.get_search_path

```
    def get_search_path()
```

Returns :	a tuple containing the list of directory names that are searched for icon themes

Note

This method is available in PyGTK 2.4 and above.

The `get_search_path()` method returns a tuple containing the current search path. See the `set_search_path()` method for more details.

gtk.IconTheme.append_search_path

```
    def append_search_path(path)
```

path :	a directory name to append to the icon path

Note

This method is available in PyGTK 2.4 and above.

The `append_search_path()` method appends the directory name specified by *`path`* to the search path. See the `set_search_path()` method for more information.

gtk.IconTheme.prepend_search_path

```
def prepend_search_path(path)
```

`path` :	a directory name to prepend to the icon path

Note

This method is available in PyGTK 2.4 and above.

The `prepend_search_path()` method prepends the directory name specified by *`path`* to the search path. See the `set_search_path()` method for more information.

gtk.IconTheme.set_custom_theme

```
def set_custom_theme(theme_name)
```

`theme_name` :	the name of icon theme to use instead of the configured theme

Note

This method is available in PyGTK 2.4 and above.

The `set_custom_theme()` method sets the theme used by the icon theme object to the theme named by *`theme_name`* usually replacing system configured theme. This method cannot be called on the icon theme objects returned from the `gtk.icon_theme_get_default()` and `gtk.icon_theme_get_default()` functions.

gtk.IconTheme.has_icon

```
def has_icon(icon_name)
```

`icon_name` :	the name of an icon
Returns :	`TRUE` if the icon theme includes an icon for *`icon_name`*.

Note

This method is available in PyGTK 2.4 and above.

The `has_icon()` method checks whether the icon theme includes an icon for the icon name specified by *`icon_name`*.

gtk.IconTheme.lookup_icon

```
def lookup_icon(icon_name, size, flags)
```

`icon_name` :	the name of the icon to lookup

size :	the desired icon size
flags :	the flags modifying the behavior of the icon lookup
Returns :	a gtk.IconInfo object containing information about the icon, or None if the icon wasn't found. Free with the gtk.IconInfo.free() method

Note

This method is available in PyGTK 2.4 and above.

The lookup_icon() method looks up the icon specified by *icon_name*, *size* and *flags* and returns a gtk.IconInfo object containing information such as the filename of the icon. The icon can then be rendered into a pixbuf using the gtk.IconInfo.load_icon() method. (the load_icon() method combines these two steps if all you need is the pixbuf.)

The value of flags must be a combination of:

gtk.ICON_LOOKUP_NO_SVG	Never return Scalable Vector Graphics (SVG) icons, even if gdk–pixbuf supports them. Cannot be used together with gtk.ICON_LOOKUP_FORCE_SVG.
gtk.ICON_LOOKUP_FORCE_SVG	Return SVG icons, even if gdk–pixbuf doesn't support them. Cannot be used together with gtk.ICON_LOOKUP_NO_SVG.
gtk.ICON_LOOKUP_USE_BUILTIN	When passed to the lookup_icon() method includes builtin icons as well as files. For a builtin icon, the gtk.IconInfo.get_filename() method returns None and you need to call the get_builtin_pixbuf() method.

gtk.IconTheme.load_icon

```
    def load_icon(icon_name, size, flags)
```

icon_name :	the name of the icon to lookup
size :	the desired icon size. The resulting icon may not be exactly this size; see the gtk.IconInfo.load_icon() method.
flags :	the flags modifying the behavior of the icon lookup
Returns :	a gtk.gdk.Pixbuf containing the rendered icon; this may be a newly created icon or a new reference to an internal icon, so you must not modify the icon.

Note

This method is available in PyGTK 2.4 and above.

The load_icon() method looks up the icon specified by *icon_name* in an icon theme, scales it to the size specified by *size* and renders it into a gtk.gdk.Pixbuf. flags is a combination of: gtk.ICON_LOOKUP_FORCE_SVG, gtk.ICON_LOOKUP_NO_SVG and gtk.ICON_LOOKUP_USE_BUILTIN (see the lookup_icon() method for more details). Since this is a convenience function, if more details about the icon are needed, use the lookup_icon() method followed by the GtkIconInfo.load_icon() method.

This method raise the GError exception if an error occurs during rendering of the icon.

gtk.IconTheme.list_icons

```
def list_icons(context=None)
```

context :	a string identifying a particular type of icon, or `None` to list all icons.
Returns :	a tuple containing the names of all the icons in the theme.

Note

This method is available in PyGTK 2.4 and above.

The `list_icons`() method lists the icons in the current icon theme. Only a subset of the icons can be listed by providing a string specified by *context*. The set of values for the context string is system dependent, but will typically include such values as 'apps' and 'mimetypes'.

gtk.IconTheme.get_example_icon_name

```
def get_example_icon_name()
```

Returns :	the name of an example icon or `None`.

Note

This method is available in PyGTK 2.4 and above.

The `get_example_icon_name`() method returns the name of an icon that is representative of the current theme (for instance, to use when presenting a list of themes to the user.)

gtk.IconTheme.rescan_if_needed

```
def rescan_if_needed()
```

Returns :	`TRUE` if the icon theme has changed and needed to be reloaded.

Note

This method is available in PyGTK 2.4 and above.

The `rescan_if_needed`() method checks to see if the icon theme has changed; if it has, any currently cached information is discarded and will be reloaded next time the icon theme is accessed.

gtk.IconTheme.get_icon_sizes

```
def get_icon_sizes(icon_name)
```

icon_name :	the name of an icon
Returns :	a tuple containing the sizes that the icon is available in.

Note

This method is available in PyGTK 2.6 and above.

The `get_icon_sizes`() method returns a tuple containing the sizes available for the icon named by *icon_name*. A size of −1 means the icon is scalable. If the icon is not found an emty tuple is returned.

Functions

gtk.icon_theme_get_default

```
    def gtk.icon_theme_get_default()
```

Returns :	A unique gtk.IconTheme associated with the default gtk.gdk.Screen. This icon theme is associated with the screen and can be used as long as the screen is open.

Note

This function is available in PyGTK 2.4 and above.

The `gtk.icon_them_get_default()` function returns the icon theme for the default screen. See the `gtk.icon_theme_get_for_screen()` function.

gtk.icon_theme_get_for_screen

```
    def gtk.icon_theme_get_for_screen(screen)
```

`screen` :	a gtk.gdk.Screen object
Returns :	A unique gtk.IconTheme associated with the default gtk.gdk.Screen. This icon theme is associated with the screen and can be used as long as the screen is open.

Note

This function is available in PyGTK 2.4 and above.

The `gtk.icon_theme_get_for_screen()` function returns the icon theme object associated with the `gtk.gdk.Screen` specified by *`screen`*. If this function has not previously been called for the given screen, a new icon theme object will be created and associated with the screen. Icon theme objects are fairly expensive to create, so using this function is usually a better choice than calling the gtk.IconTheme() constructor and setting the screen yourself; by using this function a single icon theme object will be shared between users.

gtk.icon_theme_add_builtin_icon

```
    def gtk.icon_theme_add_builtin_icon(icon_name, size, pixbuf)
```

`icon_name` :	the name of the icon to register
`size` :	the size at which to register the icon (different images can be registered for the same icon name at different sizes.)
`pixbuf` :	a gtk.gdk.Pixbuf that contains the image to use for *`icon_name`*.
Returns :	

Note

This function is available in PyGTK 2.4 and above.

The `gtk.icon_theme_add_builtin_icon()` function registers a built–in icon for icon theme lookups using *`icon_name`* as the icon name, *`size`* as the icon size and *`pixbuf`* as the icon image. The idea of built–in icons is to allow an application or library that uses themed icons to function without requiring specific icon files to be present in the file system. For instance, the default images for all of `GTK+`'s stock icons are registered as built–icons.

In general, if you use `gtk.icon_theme_add_builtin_icon()` you should also install the icon in the icon theme, so that the icon is generally available. This function will generally be used with pixbufs loaded via the `gtk.gdk.pixbuf_new_from_inline()` function.

Signals

The "changed" gtk.IconTheme Signal

```
    def callback(icontheme, user_param1, ...)
```

icontheme :	the icontheme that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

Note

This signal is available in PyGTK 2.4 and above.

The "changed" signal is emitted when the current icon theme is switched or `GTK+` detects that a change has occurred in the contents of the current icon theme.

gtk.IconView

gtk.IconView — a widget which displays a list of icons in a grid (new in PyGTK 2.6)

Synopsis

```
class gtk.IconView(gtk.Container):
    gtk.IconView(model=None)
    def set_model(model=None)
    def get_model()
    def set_text_column(column)
    def get_text_column()
    def set_markup_column(column)
    def get_markup_column()
    def set_pixbuf_column(column)
    def get_pixbuf_column()
    def get_path_at_pos(x, y)
    def selected_foreach(func, data)
    def set_selection_mode(mode)
    def get_selection_mode()
    def set_orientation(orientation)
    def get_orientation()
    def select_path(path)
    def unselect_path(path)
    def path_is_selected(path)
    def get_selected_items()
    def select_all()
    def unselect_all()
    def item_activated(path)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.IconView
```

Properties

"markup–column"	Read–Write	The number of the model column containing markup information to be displayed. If this property and the "text–column" property are both set to column numbers, this overrides the text column. If both are set to –1, no text is displayed. Allowed values: >= –1. Default value: –1. Available in GTK+ 2.6 and above.
"model"	Read–Write	The `gtk.TreeModel` for the icon view. Available in GTK+ 2.6 and above.
"orientation"	Read–Write	How the text and icon of each item are positioned relative to each other. Default value: `gtk.ORIENTATION_VERTICAL`. Available in GTK+ 2.6 and above.
"pixbuf–column"	Read–Write	The number of the model column containing the pixbufs that are displayed. Setting this property to –1 turns off the display of pixbufs. Allowed values: >= –1. Default value: –1. Available in GTK+ 2.6 and above.
"selection–mode"	Read–Write	The selection mode of the icon view. If the mode is `gtk.SELECTION_MULTIPLE`, rubberband selection is enabled, for the other modes, only keyboard selection is possible. Default value: `gtk.SELECTION_SINGLE`. Available in GTK+ 2.6 and above.
"text–column"	Read–Write	The number of the model column containing the text that is displayed. If this property and the "markup–column" property are both set to –1, no text is displayed. Allowed values: >= –1. Default value: –1. Available in GTK+ 2.6 and above.

Style Properties

"selection–box–alpha"	Read	The opacity of the selection box. Default value: 64. Available in GTK+ 2.6 and above.
"selection–box–color"	Read	The color of the selection box. Available in GTK+ 2.6 and above.

Signal Prototypes

"activate−cursor−item"	`def callback(iconview, user_param1, ...)`
"item−activated"	`def callback(iconview, path, user_param1, ...)`
"move−cursor"	`def callback(iconview, step, number, user_param1, ...)`
"select−all"	`def callback(iconview, user_param1, ...)`
"select−cursor−item"	`def callback(iconview, user_param1, ...)`
"selection−changed"	`def callback(iconview, user_param1, ...)`
"set−scroll−adjustments"	`def callback(iconview, hadj, vadj, user_param1, ...)`
"toggle−cursor−item"	`def callback(iconview, user_param1, ...)`
"unselect−all"	`def callback(iconview, user_param1, ...)`

Description

Note

This widget is available in PyGTK 2.6 and above.

The `gtk.IconView` widget provides an alternative view of a `gtk.ListStore` model. It displays the model as a grid of icons with labels. Like `gtk.TreeView`, you can select one or multiple items (depending on the selection mode, see the `set_selection_mode()` method for more information). In addition to selection with the arrow keys, `gtk.IconView` supports rubberband selection, which is controlled by dragging the pointer.

Constructor

```
    gtk.IconView(model=None)
```

`model` :	A `gtk.TreeModel`, or `None`
Returns :	a new `gtk.IconView` widget.

Note

This constructor is available in PyGTK 2.6 and above.

Creates a new `gtk.IconView` widget. If *`model`* is specified and not `None` it should specify a `gtk.TreeModel` to be used as the model.

Methods

gtk.IconView.set_model

```
    def set_model(model=None)
```

`model` :	a `gtk.TreeModel` or `None`

Note

This method is available in PyGTK 2.6 and above.

The `set_model()` method sets the "model" property to the `gtk.TreeModel` specified by *`model`*. If *`model`* is `None` the old model will be unset.

gtk.IconView.get_model

```
def get_model()
```

Returns :	The `gtk.TreeModel` used by the cell view or `None`.

Note

This method is available in PyGTK 2.6 and above.

The `get_model()` method returns the value of the "model" property which contains the `gtk.TreeModel` used by the cell view. If no model is being used this method returns `None`.

gtk.IconView.set_text_column

```
def set_text_column(column)
```

`column` :	The index of a column in the model or −1 to unset the column

Note

This method is available in PyGTK 2.6 and above.

The `set_text_column()` method sets the "text−column" property to the value of *`column`* which should be the index of a column in the model containing strings to be used for text. If *`column`* is −1 the text column is unset.

gtk.IconView.get_text_column

```
def get_text_column()
```

Returns :	The index of a model's column or −1 if unset.

Note

This method is available in PyGTK 2.6 and above.

The `get_text_column()` method returns the value of the "text−column" property which contains the index of the column in the model that provides strings to be used for text. If the "text−column" property contains −1 then no column is used for text.

gtk.IconView.set_markup_column

```
def set_markup_column(column)
```

`column` :	The index of a column in the model or −1.

Note

This method is available in PyGTK 2.6 and above.

The `set_markup_column()` method sets the "markup–column" property to the value specified by *`column`*. The "markup–column" property indicates the index of a column in the model to be used for markup information. If *`column`* is –1 the markup column will be unset. If the markup column is set, it overrides the text column set by the `set_text_column()` method.

gtk.IconView.get_markup_column

```
def get_markup_column()
```

Returns :	the index of the model column containing markup or –1.

Note

This method is available in PyGTK 2.6 and above.

The `get_markup_column` method returns the value of the "markup–column" property which contains the index of the column in the model that contains the Pango markup for text. if the "markup–column" property is –1, the markup column is unset.

gtk.IconView.set_pixbuf_column

```
def set_pixbuf_column(column)
```

`column` :	the index of a model column or –1

Note

This method is available in PyGTK 2.6 and above.

The `set_pixbuf_column()` method sets the "pixbuf–column" property to the value of *`column`*. If column >= 0 the icon view `gtk.gdk.Pixbuf` objects will be retrieved from the specified column. If *`column`* is –1 then no pixbufs will be used,

gtk.IconView.get_pixbuf_column

```
def get_pixbuf_column()
```

Returns :	The index of a column in the model or –1

Note

This method is available in PyGTK 2.6 and above.

The `get_pixbuf_column()` returns the value of the "pixbuf–column" property which contains the index of the model column use to to retrieve `gtk.gdk.Pixbuf` objects to display. If the "pixbuf–column" property is –1 no pixbufs will be retrieved.

gtk.IconView.get_path_at_pos

```
def get_path_at_pos(x, y)
```

x :	the x coordinate
y :	the y coordinate
Returns :	the path at the specified point or `None`.

Note

This method is available in PyGTK 2.6 and above.

The `get_path_at_pos()` method returns the model path of the icon located at the coordinates specified by (*x*, *y*). This method return None if there is no icon at the specified location.

gtk.IconView.selected_foreach

```
def selected_foreach(func, data)
```

func :	a callback function or method.
data :	User data to pass to *func*.

Note

This method is available in PyGTK 2.6 and above.

The `selected_foreach()` method calls the callback function or method specified by *func* for each selected icon. The signature of *func* is:

```
def func(iconview, path, user_data):
```

where *iconview* is the `gtk.IconView`, *path* is the tree path of the icon row and *user_data* is data.

gtk.IconView.set_selection_mode

```
def set_selection_mode(mode)
```

mode :	The selection mode.

Note

This method is available in PyGTK 2.6 and above.

The `set_selection_mode()` method sets the "selection−mode" property to the value of *mode*. *mode* should be one of the GTK Selection Mode Constants.

gtk.IconView.get_selection_mode

```
def get_selection_mode()
```

Returns :	the selection mode.

Note

This method is available in PyGTK 2.6 and above.

The `get_selection_mode()` method returns the value of the "selection−mode" property which contains one of the GTK Selection Mode Constants.

gtk.IconView.set_orientation

```
    def set_orientation(orientation)
```

`orientation` :	the relative position of the icon and text.

Note

This method is available in PyGTK 2.6 and above.

The `set_orientation()` method sets the "orientation" property to the value of *`orientation`*. *`orientation`* should contain one of the GTK Orientation Constants. The "orientation" property indicates the relative positioning of the icon and text.

gtk.IconView.get_orientation

```
    def get_orientation(, )
```

Returns :	the relative position of the icon and text.

Note

This method is available in PyGTK 2.6 and above.

The `get_orientation()` method returns the value of the "orientation" property that indicates the relative position between the icon and text. See the `set_orientation()` method for more information.

gtk.IconView.select_path

```
    def select_path(path)
```

`path` :	a path indicating the icon to be selected

Note

This method is available in PyGTK 2.6 and above.

The `select_path` method selects the icon with the tree path specified by *`path`*.

gtk.IconView.unselect_path

```
    def unselect_path(path)
```

`path` :	a path indicating the icon to be unselected

Note

This method is available in PyGTK 2.6 and above.

The `unselect_path` method unselects the icon with the tree path specified by *`path`*.

gtk.IconView.path_is_selected

```
    def path_is_selected(path)
```

`path` :	a path of an icon.
Returns :	`TRUE` if *`path`* is selected.

Note

This method is available in PyGTK 2.6 and above.

The `path_is_selected`() method returns `TRUE` if the icon with the tree path specified by *`path`* is selected.

gtk.IconView.get_selected_items

```
    def get_selected_items()
```

Returns :	a list of the paths of the selected icons.

Note

This method is available in PyGTK 2.6 and above.

The `get_selected_items`() method returns a list of the paths of the selected icons.

gtk.IconView.select_all

```
    def select_all()
```

Note

This method is available in PyGTK 2.6 and above.

The `select_all`() method selects all the icons if the selection mode is set to `gtk.SELECTION_MULTIPLE`.

gtk.IconView.unselect_all

```
    def unselect_all()
```

Note

This method is available in PyGTK 2.6 and above.

The `unselect_all`() method unselects all the icons.

gtk.IconView.item_activated

```
    def item_activated(path)
```

`path` :	a tree path pointing to an icon row.

Note

This method is available in PyGTK 2.6 and above.

The `item_activated()` method activates the icon pointed to by the path specified by *`path`*.

Signals

The "activate−cursor−item" gtk.IconView Signal

```
def callback(iconview, user_param1, ...)
```

`iconview` :	the widget that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled.

The "activate−cursor−item" signal is emitted when an icon is selected and the **Enter** key is pressed.

The "item−activated" gtk.IconView Signal

```
def callback(iconview, path, user_param1, ...)
```

`iconview` :	the widget that received the signal
`path` :	the path to the activated icon item.
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "item−activated" signal is emitted when an icon item is activated by the user double clicking an icon item, pressing the **Enter** key when an icon item is selected or via a call to the `item_activated()` method.

The "move−cursor" gtk.IconView Signal

```
def callback(iconview, step, number, user_param1, ...)
```

`iconview` :	the widget that received the signal
`step` :	the step size to move the cursor
`number` :	the number of steps to move
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled.

The "move−cursor" signal is emitted when the cursor is moved using the keyboard keys: **Up**, **Down**, **Control−p**, **Control−n**, **Home**, **End**, **Page_Up**, **Page_Down**, **Right**, **Left** with various **Shift** and **Control** combinations. *`step`* will be one of the GTK Movement Step Constants.

The "select−all" gtk.IconView Signal

```
def callback(iconview, user_param1, ...)
```

`iconview` :	the widget that received the signal

user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "select–all" signal is emitted when all icon items are selected by pressing **Control–a**.

The "select–cursor–item" gtk.IconView Signal

```
    def callback(iconview, user_param1, ...)
```

iconview :	the widget that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "select–cursor–item" signal is emitted when the icon item at the cursor is selected by the user pressing the **Space** key.

The "selection–changed" gtk.IconView Signal

```
    def callback(iconview, user_param1, ...)
```

iconview :	the widget that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "selection–changed" signal is emitted when the selection is changed either by user action or program method calls.

The "set–scroll–adjustments" gtk.IconView Signal

```
    def callback(iconview, hadj, vadj, user_param1, ...)
```

iconview :	the widget that received the signal
hadj :	the new horizontal gtk.Adjustment
vadj :	the new vertical gtk.Adjustment
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "set–scroll–adjustments" signal is emitted when the new horizontal and veritcal scroll gtk.Adjustment objects are set.

The "toggle–cursor–item" gtk.IconView Signal

```
    def callback(iconview, user_param1, ...)
```

iconview :	the widget that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "toggle–cursor–item" signal is emitted when the user presses **Control–Space**.

The "unselect–all" gtk.IconView Signal

```
    def callback(iconview, user_param1, ...)
```

iconview :	the widget that received the signal

user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "unselect−all" signal is emitted when the user presses **Control−Shift−a**.

Prev Up Next
gtk.IconTheme Home gtk.Image
gtk.Image
Prev The gtk Class Reference Next

gtk.Image

gtk.Image — A widget displaying an image

Synopsis

```
class gtk.Image(gtk.Misc):
    gtk.Image()
    def set_from_pixmap(pixmap, mask)
    def set_from_image(gdk_image, mask)
    def set_from_file(filename)
    def set_from_pixbuf(pixbuf)
    def set_from_stock(stock_id, size)
    def set_from_icon_set(icon_set, size)
    def set_from_animation(animation)
    def get_storage_type()
    def get_pixmap(pixmap, mask)
    def get_image(gdk_image, mask)
    def get_pixbuf()
    def get_stock(stock_id, size)
    def get_icon_set(icon_set, size)
    def get_animation()
    def get_icon_name()
    def set_from_icon_name(icon_name, size)
    def set_pixel_size(pixel_size)
    def get_pixel_size()
```

```
Functions

    def gtk.image_new_from_stock(stock_id, size)
    def gtk.image_new_from_icon_set(icon_set, size)
    def gtk.image_new_from_animation(animation)
    def gtk.image_new_from_icon_name(icon_name, size)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Misc
        +-- gtk.Image
```

Properties

"file"	Write	A filename containing the image data. Default value: `None`
"icon–name"	Read–Write	The name of the icon in the icon theme. If the icon theme is changed, the image will be updated automatically. Default value: `None`. Available in GTK 2.6 and above.
"icon–set"	Read–Write	the `gtk.IconSet` to display
"icon–size"	Read–Write	the size to use for a stock icon, named icon or icon set. Allowed values: >= 0. Default value: 4
"image"	Read–Write	a `gtk.gdk.Image` to display
"mask"	Read–Write	a bitmap `gtk.gdk.Pixmap` to use with a `gtk.gdk.Image` or a `gtk.gdk.Pixmap`
"pixbuf"	Read–Write	the `gtk.gdk.Pixbuf` to display
"pixbuf–animation"	Read–Write	the `gtk.gdk.PixbufAnimation` to display
"pixel–size"	Read–Write	a fixed size overriding the "icon–size" property for images of type `gtk.IMAGE_ICON_NAME`. Allowed values: >= –1. Default value: –1. Available in GTK+ 2.6 and above.
"pixmap"	Read–Write	a `gtk.gdk.Pixmap` to display
"stock"	Read–Write	the stock ID for a stock image to display. Default value: `None`
"storage–type"	Read	the type of the image data; one of the GTK Image Type Constants: `gtk.IMAGE_EMPTY`, `gtk.IMAGE_PIXMAP`, `gtk.IMAGE_IMAGE`, `gtk.IMAGE_PIXBUF`, `gtk.IMAGE_STOCK`, `gtk.IMAGE_ICON_SET` or `gtk.IMAGE_ANIMATION`

Description

The `gtk.Image` widget displays an image. Various kinds of objects can be displayed as an image; usually, you would load a `gtk.gdk.Pixbuf` ("pixel buffer") from a file, and then display it. The convenience method `set_from_file()` will read the file and automatically create a pixbuf that is added to the `gtk.Image` widget. If the file isn't loaded successfully, the image will contain a "broken image" icon similar to that used in many web browsers. If you want to handle errors in loading the file yourself, for example by displaying an error message, then load the image with `gtk.gdk.pixbuf_new_from_file()`, then add it to the `gtk.Image` with the `set_from_pixbuf()` method. The image file may contain an animation; if so, the `gtk.Image` will display an animation (`gtk.gdk.PixbufAnimation`) instead of a static image.

`gtk.Image` is a subclass of `gtk.Misc`, which implies that you can align it (center, left, right) and add padding to it, using the `gtk.Misc` methods. `gtk.Image` is a "no window" widget (has no `gtk.gdk.Window` of its own), so by default does not receive events. If you want to receive events on the image, such as button clicks, place the image inside a `gtk.EventBox`, then connect to the event signals on the event box.

When handling events on the event box, keep in mind that coordinates in the image may be different from event box coordinates due to the alignment and padding settings on the image (see `gtk.Misc`). The simplest way to solve this is to set the alignment to 0.0 (left/top), and set the padding to zero. Then the origin of the image will be the same as the origin of the event box.

Sometimes an application will want to avoid depending on external data files, such as image files. GTK+ comes with a program to avoid this, called **gdk–pixbuf–csource**. This program allows you to convert an image into a C variable declaration, which can then be loaded into a `gtk.gdk.Pixbuf` using the `gtk.gdk.pixbuf_new_from_inline()` function. This can also be used in `PyGTK` by modifying the output using an editor or the **sed** command as follows:

```
  gdk-pixbuf-csource imagefile | \
        sed -n -e 's/{/[/' -e 's/};/]/' -e '/".*"/p' >outfile
```

You can edit `outfile` into a Python file where it will be evaluated to a list containing a string. The string can be used directly with the `gtk.gdk.pixbuf_new_from_inline()` function to create a pixbuf that can be used in a `gtk.Image`.

Constructor

```
    gtk.Image()
```

Returns :	a newly created `gtk.Image` widget.

Creates a new empty `gtk.Image` widget.

Methods

gtk.Image.set_from_pixmap

```
    def set_from_pixmap(pixmap, mask)
```

`pixmap` :	a `gtk.gdk.Pixmap`
`mask` :	a bitmap `gtk.gdk.Pixmap` or `None`

The `set_from_pixmap()` method sets the image data from *`pixmap`* using *`mask`*.

gtk.Image.set_from_image

```
    def set_from_image(gdk_image, mask)
```

`gdk_image` :	a `gtk.gdk.Image` or `None`
`mask` :	a bitmap `gtk.gdk.Pixmap` or `None`

The `set_from_image()` method sets the image data from *`image`* using *`mask`*. A `gtk.gdk.Image` is a client−side image buffer in the pixel format of the current display. If *`image`* is `None` the current image data will be removed.

gtk.Image.set_from_file

```
    def set_from_file(filename)
```

`filename` :	a filename or `None`

The `set_from_file()` method sets the image data from the contents of the file named *`filename`*. If the file isn't found or can't be loaded, the resulting `gtk.Image` will display a "broken image" icon. This function never returns None, it always returns a valid `gtk.Image` widget. If the file contains an animation, the image will contain an animation. If *`filename`* is `None` the current image data will be removed.

If you need to detect failures to load the file, use `gtk.gdk.pixbuf_new_from_file()` to load the file yourself, then create the `gtk.Image` from the pixbuf. (Or for animations, use the gtk.gdk.PixbufAnimation() constructor.

gtk.Image.set_from_pixbuf

```
    def set_from_pixbuf(pixbuf)
```

pixbuf :	a gtk.gdk.Pixbuf or None

The set_from_pixbuf() method sets the image data using *pixbuf*. Note that this function just creates an gtk.Image from *pixbuf*. The gtk.Image created will not react to state changes. Should you want that, you should use the set_from_icon_set() method If *pixbuf* is None the current image data will be removed.

gtk.Image.set_from_stock

```
    def set_from_stock(stock_id, size)
```

stock_id :	a stock icon name
size :	a stock icon size

The set_from_stock() method sets the image data from the stock item identified by *stock_id*. Sample stock icon names are gtk.STOCK_OPEN and gtk.STOCK_OK. Stock icon sizes are gtk.ICON_SIZE_MENU, gtk.ICON_SIZE_SMALL_TOOLBAR, gtk.ICON_SIZE_LARGE_TOOLBAR, gtk.ICON_SIZE_BUTTON, gtk.ICON_SIZE_DND and gtk.ICON_SIZE_DIALOG. If the stock icon name isn't known, a "broken image" icon will be displayed instead. You can register your own stock icon names, see the gtk.IconFactory.add_default() and gtk.IconFactory.add() methods.

The stock icons are described in the Stock Items reference.

gtk.Image.set_from_icon_set

```
    def set_from_icon_set(icon_set, size)
```

icon_set :	a gtk.IconSet
size :	a stock icon size

The set_from_icon_set() method sets the image data from icon_set with the size specified by *size*. Stock icon sizes are gtk.ICON_SIZE_MENU, gtk.ICON_SIZE_SMALL_TOOLBAR, gtk.ICON_SIZE_LARGE_TOOLBAR, gtk.ICON_SIZE_BUTTON, gtk.ICON_SIZE_DND and gtk.ICON_SIZE_DIALOG.

gtk.Image.set_from_animation

```
    def set_from_animation(animation)
```

animation :	the gtk.gdk.PixbufAnimation

The set_from_animation() method sets the image data from *animation*.

gtk.Image.get_storage_type

```
    def get_storage_type()
```

Returns :	the type of the image representation being used

The get_storage_type() method gets the type of representation being used by the gtk.Image to store image data. If the gtk.Image has no image data, the return value will be gtk.IMAGE_EMPTY. The image type is one of: gtk.IMAGE_EMPTY, gtk.IMAGE_PIXMAP, gtk.IMAGE_IMAGE, gtk.IMAGE_PIXBUF, gtk.IMAGE_STOCK, gtk.IMAGE_ICON_SET or gtk.IMAGE_ANIMATION.

gtk.Image.get_pixmap

```
def get_pixmap()
```

Returns : a tuple containing the pixmap (or `None`) and the mask (or `None`)

The `get_pixmap()` method returns a tuple containing the pixmap and mask being displayed by the `gtk.Image`. Either or both the pixmap and mask may be `None`. If the storage type of the image is not either `gtk.IMAGE_EMPTY` or `gtk.IMAGE_PIXMAP` the ValueError exception will be raised.

gtk.Image.get_image

```
def get_image()
```

Returns : a tuple containing a `gtk.gdk.Image` and a mask bitmap

The `get_image()` method returns a tuple containing the `gtk.gdk.Image` and mask being displayed by the `gtk.Image`. One or both of the `gtk.gdk.Image` and mask may be `None`. If the storage type of the image is not either of `gtk.IMAGE_EMPTY` or `gtk.IMAGE_IMAGE` the ValueError exception will be raised.

gtk.Image.get_pixbuf

```
def get_pixbuf()
```

Returns : the displayed pixbuf, or `None` if the image is empty

The `get_pixbuf()` method gets the `gtk.gdk.Pixbuf` being displayed by the `gtk.Image`. The return value may be None if no image data is set. If the storage type of the image is not either `gtk.IMAGE_EMPTY` or `gtk.IMAGE_PIXBUF` the ValueError exception will be raised.

gtk.Image.get_stock

```
def get_stock()
```

Returns : a tuple containing the stock icon name and the stock icon size of the image data

The `get_stock()` method returns a tuple containing the stock icon identifier (may be `None`) and size being displayed by the `gtk.Image`. The size will be one of: `gtk.ICON_SIZE_MENU`, `gtk.ICON_SIZE_SMALL_TOOLBAR`, `gtk.ICON_SIZE_LARGE_TOOLBAR`, `gtk.ICON_SIZE_BUTTON`, `gtk.ICON_SIZE_DND` or `gtk.ICON_SIZE_DIALOG`. If the storage type of the image is not either `gtk.IMAGE_EMPTY` or `gtk.IMAGE_STOCK` the ValueError exception will be raised.

gtk.Image.get_icon_set

```
def get_icon_set()
```

Returns : a tuple containing a `gtk.IconSet` and a stock icon size

The `get_icon_set()` method returns a tuple containing the icon set (may be `None`) and size being displayed by the `gtk.Image`. The size will be one of: `gtk.ICON_SIZE_MENU`, `gtk.ICON_SIZE_SMALL_TOOLBAR`, `gtk.ICON_SIZE_LARGE_TOOLBAR`, `gtk.ICON_SIZE_BUTTON`, `gtk.ICON_SIZE_DND` or `gtk.ICON_SIZE_DIALOG`. If the storage type of the image is not either `gtk.IMAGE_EMPTY` or `gtk.IMAGE_ICON_SET` the ValueError exception will be raised.

gtk.Image.get_animation

```
def get_animation()
```

Returns :	the displayed animation, or `None` if the image is empty

The `get_animation()` method gets the `gtk.gdk.PixbufAnimation` (may be None if there is no image data) being displayed by the `gtk.Image`. If the storage type of the image is not either `gtk.IMAGE_EMPTY` or `gtk.IMAGE_ANIMATION` the ValueError exception will be raised.

gtk.Image.get_icon_name

```
def get_icon_name()
```

Returns :	a 2–tuple containing the name and size of the displayed icon.

Note

This method is available in PyGTK 2.6 and above.

The `get_icon_name()` method returns a 2–tuple containing the values of the "icon–name" and "icon–size" properties respectively if the "icon–name" property is not `None`. If the "icon–name" property is `None` the 2–tuple returned will be:

```
(None, <enum GTK_ICON_SIZE_INVALID of type GtkIconSize>)
```

gtk.Image.set_from_icon_name

```
def set_from_icon_name(icon_name, size)
```

`icon_name` :	an icon name
`size` :	a stock icon size

Note

This method is available in PyGTK 2.6 and above.

The `set_from_icon_name()` method sets the "icon–name" and "icon–size" properties to the values of *`icon_name`* and *`size`* respectively. *`icon_name`* should be the name of an icon in the current icon theme. If *`icon_name`* isn't known, a "broken image" icon will be displayed instead. If the current icon theme is changed, the icon will be updated appropriately.

gtk.Image.set_pixel_size

```
def set_pixel_size(pixel_size)
```

`pixel_size` :	the new pixel size to be used for named icons

Note

This method is available in PyGTK 2.6 and above.

The `set_pixel_size()` method sets the "pixel–size" property to the value specified by *`pixel_size`*. If the pixel size is set to a value != –1 the "pixel–size" property is used instead of the icon size set by the `set_from_icon_name()` method.

gtk.Image.get_pixel_size

```
    def get_pixel_size()
```

Returns :	the pixel size used for named icons.

Note

This method is available in PyGTK 2.6 and above.

The `get_pixel_size()` method returns the value of the "pixel−size" property which specifies the pixel size to be used for named icons.

Functions

gtk.image_new_from_stock

```
    def gtk.image_new_from_stock(stock_id, size)
```

`stock_id` :	a stock icon name
`size` :	an integer representing an icon size
Returns :	a new `gtk.Image` displaying the stock icon

The `gtk.image_new_from_stock()` function returns a new `gtk.Image` displaying the stock icon specified by *`stock_id`* with the specified *`size`*. Sample stock icon names are `gtk.STOCK_OPEN`, `gtk.STOCK_OK` – see the `set_from_stock()` method for detailed information on the `PyGTK` stock icons. . Sample stock sizes are `gtk.ICON_SIZE_MENU`, `gtk.ICON_SIZE_SMALL_TOOLBAR` – see the `gtk.icon_size_lookup()` function for more detail. If the stock icon name isn't known, a "broken image" icon will be displayed instead. You can register your own stock icon names, see the `gtk.IconFactory.add_default()` and `gtk.IconFactory.add()` methods.

gtk.image_new_from_icon_set

```
    def gtk.image_new_from_icon_set(icon_set, size)
```

`icon_set` :	a `gtk.IconSet` object
`size` :	an integer representing an icon size
Returns :	a new `gtk.Image` object

The `gtk.image_new_from_icon_set()` function returns a new `gtk.Image` created from the `gtk.IconSet` specified by *`icon_set`* with the specified *`size`*. Sample stock sizes are `gtk.ICON_SIZE_MENU`, `gtk.ICON_SIZE_SMALL_TOOLBAR` – see the `gtk.icon_size_lookup()` function for more detail. Instead of using this function, usually it's better to create a `gtk.IconFactory`, put your icon sets in the icon factory, add the icon factory to the list of default factories with the `add_default()` method, and then use the `gtk.image_new_from_stock()` function. This will allow themes to override the icon you ship with your application.

gtk.image_new_from_animation

```
    def gtk.image_new_from_animation(animation)
```

`animation` :	a `gtk.gdk.PixbufAnimation` object
Returns :	a new `gtk.Image` object

The `gtk.image_new_from_animation()` function returns a new `gtk.Image` object containing the `gtk.gdk.PixbufAnimation` specified by *`animation`*.

gtk.image_new_from_icon_name

```
    def gtk.image_new_from_icon_name(icon_name, size)
```

`icon_name` :	an icon name
`size` :	a stock icon size
Returns :	a new `gtk.Image` widget.

Note

This function is available in PyGTK 2.6 and above.

The `gtk.image_new_from_icon_name()` function returns a new `gtk.Image` object displaying the named theme icon specified by *`icon_name`* with the icon size specified by *`size`*. If the icon name isn't known, a "broken image" icon will be displayed instead. If the current icon theme is changed, the icon will be updated appropriately. The "icon−name" and "icon−size" properties are also set by this function.

Prev	Up	Next
gtk.IconView	Home	gtk.ImageMenuItem

	gtk.ImageMenuItem	
Prev	**The gtk Class Reference**	Next

gtk.ImageMenuItem

gtk.ImageMenuItem — a menuitem that displays an image with an accel label

Synopsis

```
class gtk.ImageMenuItem(gtk.MenuItem):
    gtk.ImageMenuItem(stock_id=None, accel_group=None)
    def set_image(image)
    def get_image()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Item
            +-- gtk.MenuItem
              +-- gtk.ImageMenuItem
```

Properties

"image"	Read−Write	the child widget that is next to the text in the menu item

Description

The `gtk.ImageMenuItem` widget is a subclass of `gtk.MenuItem` that contains an image widget and a `gtk.AccelLabel` though a `gtk.ImageMenuItem` can be created that has no image and an empty label.

Constructor

```
    gtk.ImageMenuItem(stock_id=None, accel_group=None)
```

`stock_id` :	the stock icon ID or `None` if no image is needed
`accel_group` :	the accel group to add the accel label mnemonic to
Returns :	a new `gtk.ImageMenuItem` widget

Creates a new `gtk.ImageMenuItem` with a stock label and image specified by *`stock_id`*. If *`stock_id`* is not a stock item then the image will be the "broken image" and the label text will be the string in *`stock_id`*. The label text will be parsed for underscore characters to indicate the mnemonic character for the accelerator.

If *`stock_id`* specifies a stock item and *`accel_group`* specifies a `gtk.AccelGroup` the accelerator is added to *`accel_group`*.

Methods

gtk.ImageMenuItem.set_image

```
    def set_image(image)
```

`image` :	a widget to set as the image for the menu item.

The `set_image`() method sets the image of the image menu item to the widget specified in *`image`*.

gtk.ImageMenuItem.get_image

```
    def get_image()
```

Returns :	the image in the image menu item

The `get_image`() method gets the widget that is currently set as the image of image menu item. See `set_image()`.

Prev Up Next
gtk.Image Home gtk.IMContext

gtk.IMContext
Prev **The gtk Class Reference** Next

gtk.IMContext

gtk.IMContext — an abstract base class defining a generic input method interface

Synopsis

```
class gtk.IMContext(gtk.Object):
    def set_client_window(window)
    def get_preedit_string()
    def filter_keypress(key)
    def focus_in()
    def focus_out()
    def reset()
    def set_cursor_location(area)
    def set_use_preedit(use_preedit)
    def set_surrounding(text, len, cursor_index)
    def get_surrounding()
    def delete_surrounding(offset, n_chars)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.IMContext
```

Signal Prototypes

"commit"	`def callback(`*`imcontext, string, user_param1, ...`*`)`
"delete–surrounding"	`def callback(`*`imcontext, offset, n_chars, user_param1, ...`*`)`
"preedit–changed"	`def callback(`*`imcontext, user_param1, ...`*`)`
"preedit–end"	`def callback(`*`imcontext, user_param1, ...`*`)`
"preedit–start"	`def callback(`*`imcontext, user_param1, ...`*`)`
"retrieve–surrounding"	`def callback(`*`imcontext, user_param1, ...`*`)`

Description

The `gtk.IMContext` is an abstract base class used to provide objects to manage the context for input methods used to support text input in various natural languages. The character input for some languages (e.g. Chinese, Japanese and Korean) cannot be easily input using standard keyboards so multiple keystrokes are used to input a single character. Input methods are used to help this process by providing feedback of the characters input and managing the context and valid combinations. The`gtk.IMContext` subclasses manage the required context information for applications and widgets such as the `gtk.Entry` and `gtk.TextView` widgets.

The `gtk.IMContext` manages the context of the text surrounding the cursor and the preedit text that provides feedback about the in–process multiple character composition.

Methods

gtk.IMContext.set_client_window

```
    def set_client_window(window)
```

`window` : the client window. This may be `None` to indicate that the previous client window no longer exists.

The `set_client_window()` method set the client window for the input context; this is the `gtk.gdk.Window` in which the input appears. This window is used in order to correctly position status windows, and may also be used for purposes internal to the input method.

gtk.IMContext.get_preedit_string

```
def get_preedit_string()
```

Returns :	a tuple containing the preedit string, the attribute list and the position of cursor (in bytes) within the preedit string.

The `get_preedit_string()` method returns a tuple containing: the current preedit string for the input context, a list of attributes to apply to the string and the cursor position within the string. This string should be displayed inserted at the insertion point.

gtk.IMContext.filter_keypress

```
def filter_keypress(event)
```

`key` :	the key event
Returns :	`TRUE` if the input method handled the keystroke.

The `filter_keypress()` method allows an input method to internally handle a key press event. If this function returns `TRUE`, then no further processing should be done for this keystroke.

gtk.IMContext.focus_in

```
def focus_in()
```

The `focus_in()` method notifies the input method that the widget to which this input context corresponds has gained focus. The input method may, for example, change the displayed feedback to reflect this change.

gtk.IMContext.focus_out

```
def focus_out()
```

The `focus_out()` method notifies the input method that the widget to which this input context corresponds has lost focus. The input method may, for example, change the displayed feedback or reset the context state to reflect this change.

gtk.IMContext.reset

```
def reset()
```

The `reset()` method notifies the input method that a significant change in context (such as a change in cursor position) has been made. This will typically cause the input method to clear the preedit state.

gtk.IMContext.set_cursor_location

```
def set_cursor_location(area)
```

`area` :	new location

The `set_cursor_location()` method notifies the input method that a change in cursor position has been made.

gtk.IMContext.set_use_preedit

```
    def set_use_preedit(use_preedit)
```

use_preedit :	if TRUE the IM context should use the preedit string.

The `set_use_preedit()` method sets the use preedit setting to the value of use_preedit. If *use_preedit* is `TRUE` (the default) the IM context should use the preedit string to display feedback. If *use_preedit* is `FALSE` the IM context may use some other method to display feedback, such as displaying it in a child of the root window.

gtk.IMContext.set_surrounding

```
    def set_surrounding(text, len, cursor_index)
```

text :	the text surrounding the insertion point, as UTF–8. the preedit string should not be included within it.
len :	the length of *text*, or –1 to calculate the length of *text*.
cursor_index :	the byte index of the insertion cursor within *text*.

The `set_surrounding()` method sets surrounding context around the insertion point and preedit string. This function is expected to be called in response to the "retrieve_surrounding" signal, and will likely have no effect if called at other times.

gtk.IMContext.get_surrounding

```
    def get_surrounding()
```

Returns :	a tuple containing the UTF–8 encoded string of text holding context around the insertion point and the byte index of the insertion cursor within the string, or None if no surrounding context was retrieved.

The `get_surrounding()` method returns a tuple containing the text surrounding the cursor and the byte index of the cursor within the text. Input methods typically want context in order to constrain input text based on existing text; this is important for languages such as Thai where only some sequences of characters are allowed.

This function is implemented by emitting the "retrieve_surrounding" signal on the input method; in response to this signal, a widget should provide as much context as is available, up to an entire paragraph, by calling `set_surrounding()`. Note that there is no obligation for a widget to respond to the "retrieve_surrounding" signal, so input methods must be prepared to function without context.

gtk.IMContext.delete_surrounding

```
    def delete_surrounding(offset, n_chars)
```

offset :	the offset from cursor position in chars; a negative value means start before the cursor.
n_chars :	the number of characters to delete.
Returns :	`TRUE` if the signal was handled.

The delete_surrounding() method asks the widget that the input context is attached to to delete characters around the cursor position by emitting the "delete_surrounding" signal. Note that *offset* and *n_chars* are in characters not in bytes, which differs from the usage other places in the `gtk.IMContext` class.

In order to use this function, you should first call `get_surrounding()` to get the current context, and call this function immediately afterward to make sure that you know what you are deleting. You should also account for the fact that even if the signal was handled, the input context might not have deleted all the characters that were requested to be deleted.

This function is used by an input method that wants to make substitutions in the existing text in response to new input. It is not useful for applications.

Signals

The "commit" gtk.IMContext Signal

```
def callback(imcontext, string, user_param1, ...)
```

imcontext :	the imcontext that received the signal
string:	the text to be committed
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "commit" signal is emitted when the text is ready to be displayed.

The "delete–surrounding" gtk.IMContext Signal

```
def callback(imcontext, user_param1, ...)
```

imcontext :	the imcontext that received the signal
offset :	the offset from the cursor position of the text to be deleted
n_chars :	the number of characters to be deleted
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled.

The "delete–surrounding" signal is emitted when the input method needs to delete the context text.

The "preedit–changed" gtk.IMContext Signal

```
def callback(imcontext, user_param1, ...)
```

imcontext :	the imcontext that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "preedit–changed" signal is emitted when the preedit text has changed.

The "preedit–end" gtk.IMContext Signal

```
def callback(imcontext, user_param1, ...)
```

imcontext :	the imcontext that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "preedit–end" signal is emitted when preediting is completed.

The "preedit−start" gtk.IMContext Signal

```
    def callback(imcontext, user_param1, ...)
```

imcontext :	the imcontext that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "preedit−start" signal is emitted when preediting is started.

The "retrieve−surrounding" gtk.IMContext Signal

```
    def callback(imcontext, user_param1, ...)
```

imcontext :	the imcontext that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled.

The "retrieve−surrounding" signal is emitted when the input method requires the context surrounding the cursor. The callback should set the input method surrounding context by calling the `set_surrounding()` method. The method returns `TRUE` if the signal was handled by the callback.

gtk.IMContextSimple

gtk.IMContextSimple — an input method context object that supports "simple" input methods

Synopsis

```
class gtk.IMContextSimple(gtk.IMContext):
    gtk.IMContextSimple()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.IMContext
      +-- gtk.IMContextSimple
```

Description

The `gtk.IMContextSimple` class is a subclass of `gtk.IMContext` that provides context support for "simple" input methods. `gtk.IMContextSimple` does direct keysym to unicode translation and table−driven composition.

Constructor

```
    gtk.IMContextSimple()
```

Returns :	a new gtk.IMContextSimple.

Creates a new gtk.IMContextSimple object.

Methods

gtk.IMMulticontext

gtk.IMMulticontext — an input method context object that manages the use of multiple input method contexts for a widget

Synopsis

```
class gtk.IMMulticontext(gtk.IMContext):
    gtk.IMMulticontext()
    def append_menuitems(menushell)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.IMContext
      +-- gtk.IMMulticontext
```

Description

The gtk.IMMulticontext class is a subclass of gtk.IMContext that manages the use of multiple input method contexts for a widget including the ability to switch between contexts on the fly. A gtk.IMMulticontext object will proxy the method calls and signals to and from the object implementing the input method.

Constructor

```
    gtk.IMMulticontext()
```

Returns :	a new gtk.IMMulticontext object.

Creates a new gtk.IMMulticontext object.

Methods

gtk.IMMulticontext.append_menuitems

```
  def append_menuitems(menushell)
```

menushell :	a gtk.MenuShell widget

The append_menuitems() method adds menuitems for various available input methods to a menu; the menuitems, when selected, will switch the input method for the context and the global default input method.

Prev	Up	Next
gtk.IMContextSimple	Home	gtk.InputDialog

gtk.InputDialog

Prev	**The gtk Class Reference**	Next

gtk.InputDialog

gtk.InputDialog a dialog for configuring devices for the XInput extension.

Synopsis

```
class gtk.InputDialog(gtk.Dialog):
    gtk.InputDialog()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Window
            +-- gtk.Dialog
              +-- gtk.InputDialog
```

Signal Prototypes

"disable–device"	def callback(*inputdialog*, *deviceid*, *user_param1*, *...*)
"enable–device"	def callback(*inputdialog*, *deviceid*, *user_param1*, *...*)

Description

Note

This widget is considered too specialized or little–used for PyGTK, and will in the future be moved to some other package. If your application needs this widget, feel free to use it, as the widget does work and is useful in some applications; it's just not of general interest. However, it will eventually move out of the PyGTK distribution.

gtk.InputDialog displays a dialog which allows the user to configure XInput extension devices. For each device, they can control the mode of the device (disabled, screen–relative, or window–relative), the mapping of axes to coordinates, and the mapping of the devices macro keys to key press events. gtk.InputDialog contains two buttons to which the application can connect; one for closing the dialog, and one for saving the changes. No actions are bound to these by default. The changes that the user makes take effect immediately.

Constructor

```
  gtk.InputDialog()
```

Returns :	a new gtk.InputDialog widget

Creates a new gtk.InputDialog.

Signals

The "disable–device" gtk.InputDialog Signal

```
  def callback(inputdialog, deviceid, user_param1, ...)
```

inputdialog :	the inputdialog that received the signal
deviceid :	the ID of the newly disabled device.
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "disable–device" signal is emitted when the user changes the mode of a device from a `gtk.gdk.MODE_SCREEN` or `gtk.gdk.MODE_WINDOW` to `gtk.gdk.MODE_ENABLED`.

The "enable–device" gtk.InputDialog Signal

```
  def callback(inputdialog, deviceid, user_param1, ...)
```

inputdialog :	the inputdialog that received the signal
deviceid :	the ID of the newly disabled device.
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "enable–device" signal is emitted when the user changes the mode of a device from `gtk.gdk.MODE_DISABLED` to a `gtk.gdk.MODE_SCREEN` or `gtk.gdk.MODE_WINDOW`.

Prev Up Next
gtk.IMMulticontext Home gtk.Invisible

gtk.Invisible
Prev The gtk Class Reference Next

gtk.Invisible

gtk.Invisible — internally–used widget which is not displayed.

Synopsis

```
class gtk.Invisible(gtk.Widget):
    gtk.Invisible()
    def set_screen(screen)
    def get_screen()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Invisible
```

Properties

"screen"	Read–Write	The screen where this window will be displayed.

Description

The `gtk.Invisible` widget is used internally in GTK, and is probably not useful for application developers. It is used for reliable pointer grabs and selection handling in the code for drag–and–drop.

Constructor

```
    gtk.Invisible()
```

Returns :	a new `gtk.Invisible` widget

Creates a new `gtk.Invisible` widget.

Methods

gtk.Invisible.set_screen

```
    def set_screen(screen)
```

`screen` :	a `gtk.gdk.Screen` object

Note

This method is available in PyGTK 2.2 and above.

The `set_screen()` method sets the `gtk.gdk.Screen` (specified by *`screen`*) where the `gtk.Invisible` object will be displayed.

gtk.Invisible.get_screen

```
    def get_screen()
```

Returns :, :,	the associated `gtk.gdk.Screen`

Note

This method is available in PyGTK 2.2 and above.

The `get_screen()` method returns the associated `gtk.gdk.Screen` object.

gtk.Item

gtk.Item abstract base class for `gtk.MenuItem`

Synopsis

```
class gtk.Item(gtk.Bin):
    def select()
    def deselect()
    def toggle()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Item
```

Signal Prototypes

"deselect"	`def callback(item, user_param1, ...)`
"select"	`def callback(item, user_param1, ...)`
"toggle"	`def callback(item, user_param1, ...)`

Description

The `gtk.Item` widget is an abstract base class for `gtk.MenuItem`.

Methods

gtk.Item.select

```
    def select()
```

The `select()` method emits the "select" signal on the item.

gtk.Item.deselect

```
def deselect()
```

The `deselect()` method emits the "deselect" signal on the item.

gtk.Item.toggle

```
def toggle()
```

The `toggle()` method emits the "toggle" signal on the item.

Signals

The "deselect" gtk.Item Signal

```
def callback(item, user_param1, ...)
```

item :	the item that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "deselect" signal is emitted when the item is deselected.

The "select" gtk.Item Signal

```
def callback(item, user_param1, ...)
```

item :	the item that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "select" signal is emitted when the item is selected.

The "toggle" gtk.Item Signal

```
def callback(item, user_param1, ...)
```

item :	the item that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "toggle" signal is emitted when the item is toggled.

Prev	Up	Next
gtk.Invisible	Home	gtk.ItemFactory

gtk.ItemFactory

Prev	**The gtk Class Reference**	Next

gtk.ItemFactory

gtk.ItemFactory — creates menus, menubars and option menus from a data description.

Synopsis

```
class gtk.ItemFactory(gtk.Object):
    gtk.ItemFactory(container_type, path, accel_group=None)
    def construct(container_type, path, accel_group)
    def get_item(path)
    def get_widget(path)
    def get_widget_by_action(action)
    def get_item_by_action(action)
    def create_items(entries, callback_data=None)
    def delete_item(path)
    def popup(x, y, mouse_button, time=GDK_CURRENT_TIME)
Functions

    def gtk.item_factory_from_widget(widget)
    def gtk.item_factory_path_from_widget(widget)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.ItemFactory
```

Description

Warning

`gtk.ItemFactory` is deprecated in PyGTK 2.4 and above. The `gtk.UIManager` should be used instead.

The `gtk.ItemFactory` provides a convenient way to create and manage menus, menubars and option menus from a data description. The data description is a tuple or list containing a list of entry tuples that each describe an individual menu, menuitem, etc. Each entry tuple may contain the following entry fields though only the path must be specified; the other fields are optional:

- A path that defines the logical position of the menu or menuitem in the menu hierarchy. a path is similar to a file path in that it starts with a slash (/) character and all components are joined by a slash (/) character. The last component may have an underscore that indicates that the following character is to be used as the accelerator mnemonic.
- An accelerator that defines a key sequence that will activate the menuitem. The key sequence is a set of zero or more modifiers followed by a single key. The modifier keys are:

"<alt>"	**Alt–L**or **Alt–R**
"<ctl>", "<ctrl>", "<control>",	**Ctrl–L**or **Ctrl–R**
"<shift>", "<shft>",	**Shift–L**or **Shift–R**

- A callback function or method that will be invoked when the menu item is activated or the accelerator key sequence is pressed. The callback function is defined as either:

```
    def callback(callback_data, callback_action, widget)
    def callback(callback_action, widget)
```

where *`callback_action`* is the callback action defined below, *`callback_data`* is the data passed with the `create_items()` method and *`widget`* is the menuitem widget. The second definition must be used if no *`callback_data`* argument is specified by the call to `create_items()`.

- A callback action that is an arbitrary integer value.
- An item type is a string describing the type of the item:

"None", "", "Item",	a simple menu item
"<Title>"	a title item
"<ImageItem>"	an image item
"<StockItem>"	an item holding a stock image
"<CheckItem>"	a check item
"<ToggleItem>"	a toggle item
"<RadioItem>"	a radio item
<path>	the path of a radio item group to add item to
"<Separator>"	a separator
"<Tearoff>"	a tearoff separator
"<Branch>"	an item to hold sub items
"<LastBranch>"	a right justified item to hold sub items

- extra data that is either a `gtk.gdk.Pixbuf` or a stock ID

Constructor

```
    gtk.ItemFactory(container_type, path, accel_group=None)
```

`container_type` :	the kind of menu to create; one of: `gtk.MenuBar`, `gtk.Menu` or `gtk.OptionMenu`.
`path` :	the path of the new item factory, a string of the form `"<name>"`
`accel_group` :	a `gtk.AccelGroup` to which the accelerators for the menu items will be added, or `None` to create a new one
Returns :	a new `gtk.ItemFactory`

Creates a new `gtk.ItemFactory` object.

Methods

gtk.ItemFactory.construct

```
    def construct(container_type, path, accel_group)
```

`container_type` :	the kind of menu to create; one of: `gtk.MenuBar`, `gtk.Menu` or `gtk.OptionMenu`.
`path` :	the path of the item factory, a string of the form `"<name>"`
`accel_group` :	a `gtk.AccelGroup` to which the accelerators for the menu items will be added, or `None` to create a new one

The `construct()` method initializes an item factory.

gtk.ItemFactory.get_item

```
    def get_item(path)
```

`path` :	the path to the menu item
Returns :	the menu item with the specified *`path`*, or `None` if *`path`* doesn't lead to a menu item

The `get_item`() method returns the menu item that corresponds to *`path`*. If the widget corresponding to *`path`* is a menu item that opens a submenu, then the item is returned. If you are interested in the submenu, use the `get_widget()` method instead.

gtk.ItemFactory.get_widget

```
    def get_widget(path)
```

`path` :	the path to the widget
Returns :	the widget associated with the specified *`path`*, or `None` if *`path`* doesn't lead to a widget

The `get_widget`() method returns the widget that corresponds to *`path`*. If the widget corresponding to *`path`* is a menu item that opens a submenu, then the submenu is returned. If you are interested in the menu item, use `get_item()` instead.

gtk.ItemFactory.get_widget_by_action

```
    def get_widget_by_action(action)
```

`action` :	a callback action value
Returns :	the widget that corresponds to the given action, or `None` if no widget was found

The `get_widget_by_action`() method returns the widget that is associated with the specified *`action`*. If there are multiple items with the same *`action`*, the result is undefined.

gtk.ItemFactory.get_item_by_action

```
    def get_item_by_action(action)
```

`action` :	a callback action value
Returns :	the menu item that corresponds to the specified *`action`*, or `None` if no menu item was found

The `get_item_by_action`() returns the menu item that is associated with the specified *`action`*.

gtk.ItemFactory.create_items

```
    def create_items(entries, callback_data=None)
```

`entries` :	a tuple or list of item factory entries
`callback_data` :	optional data passed to the callback functions of all entries

The `create_items`() method creates the menu items from the specified item factory *`entries`*. If no *`callback_data`* is specified it will not be passed to the callback functions i.e the callback functions will be passed one less argument.

gtk.ItemFactory.delete_item

```
    def delete_item(path)
```

`path` :	a path

The `delete_item()` method deletes the menu item that was created with the specified *`path`*.

gtk.ItemFactory.popup

```
    def popup(x, y, mouse_button, time=0L)
```

x :	the x position
y :	the y position
mouse_button :	the mouse button that was pressed to initiate this action
time :	an optional timestamp for this action; default is 0L which means use the current time

The `popup()` method pops up the menu constructed from the item factory at the specified location (*x*, *y*). This method is generally invoked in response to a "button_press_event" so the arguments are retrieved from the event information.

Functions

gtk.item_factory_from_widget

```
    def gtk.item_factory_from_widget(widget)
```

widget :	a `gtk.Widget` object
Returns :	the `gtk.ItemFactory` that created *`widget`*

The `gtk.item_factory_from_widget()` function returns the `gtk.ItemFactory` object that created the `gtk.Widget` specified by *`widget`*.

gtk.item_factory_path_from_widget

```
    def gtk.item_factory_path_from_widget(widget)
```

widget :	a `gtk.Widget` object
Returns :	the full path to the `gtk.ItemFactory` that created *`widget`*

The `gtk.item_factory_path_from_widget()` function returns the full path to the `gtk.ItemFactory` that created the `gtk.Widget` specified by *`widget`*.

gtk.Label

gtk.Label — a widget that displays a limited amount of read–only text

Synopsis

```
class gtk.Label(gtk.Misc):
    gtk.Label(str=None)
    def set_text(str)
    def get_text()
    def set_attributes(attrs)
    def get_attributes()
    def set_label(str)
    def get_label()
    def set_markup(str)
    def set_use_markup(setting)
    def get_use_markup()
    def set_use_underline(setting)
    def get_use_underline()
    def set_markup_with_mnemonic(str)
    def get_mnemonic_keyval()
    def set_mnemonic_widget(widget)
    def get_mnemonic_widget()
    def set_text_with_mnemonic(str)
    def set_justify(jtype)
    def get_justify()
    def set_pattern(pattern)
    def set_line_wrap(wrap)
    def get_line_wrap()
    def set_selectable(setting)
    def get_selectable()
    def select_region(start_offset, end_offset)
    def get_selection_bounds()
    def get_layout()
    def get_layout_offsets()
    def set_ellipsize(mode)
    def get_ellipsize()
    def set_width_chars(n_chars)
    def get_width_chars()
    def set_single_line_mode(single_line_mode)
    def get_single_line_mode()
    def get_max_width_chars()
    def set_max_width_chars(n_chars)
    def get_angle()
    def set_angle(angle)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Misc
        +-- gtk.Label
```

Properties

"angle"	Read–Write	The angle that the baseline of the label makes with the horizontal, in degrees, measured counterclockwise. An angle of 90 reads from from bottom to top, an angle of 270, from top to bottom. Ignored if the label is selectable, wrapped, or ellipsized. Allowed values: [0,360] Default value: 0. Available in GTK+ 2.6 and above.
"attributes"	Read–Write	A list of Pango style attributes to apply to the text of the label.
"cursor–position"	Read	The current position of the insertion cursor in chars. Allowed values: >= 0. Default value: 0
"ellipsize"	Read–Write	

		The preferred place to ellipsize the string, if the label does not have enough room to display the entire string, specified as one of the Pango Ellipsize Mode Constants. Note that setting this property to a value other than `pango.ELLIPSIZE_NONE` has the side−effect that the label requests only enough space to display the ellipsis "...". In particular, this means that ellipsizing labels don't work well in notebook tabs, unless the tab's "tab−expand" property is set to `TRUE`. Other means to set a label's width are with the `gtk.Widget.set_size_request()` and `set_width_chars()` methods. Default value: `pango.ELLIPSIZE_NONE`. Available in GTK+ 2.6 and above.
"justify"	Read−Write	The alignment of the lines in the text of the label relative to each other. The possible values are: `gtk.JUSTIFY_LEFT`, `gtk.JUSTIFY_RIGHT`, `gtk.JUSTIFY_CENTER`, `gtk.JUSTIFY_FILL`. This does NOT affect the alignment of the label within its allocation. Default value: `gtk.JUSTIFY_LEFT`
"label"	Read−Write	The text of the label. Default value: `None`
"max−width−chars"	Read−Write	The desired maximum width of the label, in characters. If this property is set to −1, the width will be calculated automatically, otherwise the label will request space for no more than the requested number of characters. If the "width−chars" property is set to a positive value, then the "max−width−chars" property is ignored. Allowed values: >= −1. Default value: −1. Available in GTK+ 2.6 and above.
"mnemonic−keyval"	Read	The mnemonic accelerator key for this label. Default value: 16777215
"mnemonic−widget"	Read−Write	The widget to be activated when the label's mnemonic key is pressed.
"pattern"	Write	A string with _ characters in positions used to identify to characters in the text to underline. Default value: `None`
"selectable"	Read−Write	If `TRUE`, the label text can be selected with the mouse. Default value: `FALSE`
"selection−bound"	Read	The position of the opposite end of the selection from the cursor in chars. Allowed values: >= 0. Default value: 0.
"single−line−mode"	Read−Write	If `TRUE` the label is in single line mode. In single line mode, the height of the label does not depend on the actual text, it is always set to ascent + descent of the font. This can be an advantage in situations where resizing the label because of text changes would be distracting, e.g. in a statusbar. Default value: `FALSE`. Available in GTK+ 2.6 and above.

"use−markup"	Read−Write	If `TRUE`, the text of the label includes XML markup. Default value: `FALSE`
"use−underline"	Read−Write	If `TRUE`, an underscore in the text indicates the next character should be used for the mnemonic accelerator key. Default value: `FALSE`
"width−chars"	Read−Write	The desired width of the label, in characters. If this property is set to −1, the width will be calculated automatically, otherwise the label will request either 3 characters or the property value, whichever is greater. Allowed values: >= −1. Default value: −1. Available in GTK+ 2.6 and above.
"wrap"	Read−Write	If `TRUE`, wrap lines if the text becomes too wide. Default value: `FALSE`

Signal Prototypes

"copy−clipboard"	`def callback(`*`label, user_param1, ...`*`)`
"move−cursor"	`def callback(`*`label, step, count, extend_selection, user_param1, ...`*`)`
"populate−popup"	`def callback(`*`label, menu, user_param1, ...`*`)`

Description

The `gtk.Label` is a widget class that displays a limited amount of read−only text. Labels are used by several widgets (e.g. `gtk.Button`, and its subclasses, `gtk.MenuItem`, etc.) to provide text display as well as by applications to display messages, etc, to the user. Most of the functionality of a `gtk.Label` is directed at modifying the style and layout of the text within the widget allocation. A `gtk.Label` is a "windowless" object which means that it cannot receive events directly. A `gtk.EventBox` can be used to provide event handling capabilities to a `gtk.Label` widget if needed.

Mnemonics

Label text may be specified with embedded underscore characters that are used to indicate that the following character should be underlined and used as the mnemonic accelerator (if it's the first underlined character). The `set_text_with_mnemonic()` method is used to parse the label text for a mnemonic characters. Mnemonics automatically activate any activatable widget the label is inside, such as a `gtk.Button`; if the label is not inside an activatable widget, you have to tell the label about the target using the `set_mnemonic_widget()` method. Here's a simple example where the label is inside a button:

```
  # Pressing Alt+H will activate this button
  button = gtk.Button()
  label = gtk.Label("_Hello")
  label.set_use_underline(True)
  button.add(label)
```

As a convenience you can create a button with a mnemonic label as follows:

```
  # Pressing Alt+H will activate this button
  button = gtk.Button(label="_Hello", use_underline=True)
```

To create a mnemonic for a widget alongside the label, such as a `gtk.Entry`, you have to point the label at the entry with the `set_mnemonic_widget()` method:

```
# Pressing Alt+H will focus the entry
entry = gtk.Entry()
label = gtk.Label("_Hello")
label.set_use_underline(True)
label.set_mnemonic_widget(entry)
```

Markup (styled text)

To make it easy to format text in a label (changing colors, fonts, etc.), the label text can be provided in the Pango markup format which is a simple XML markup format. The `gtk.Label.set_markup()` method sets the label using text in valid markup format (e.g. '<', '>' and '&' characters must be replaced by &lt;, &gt; and &amp; respectively. For example:

```
label = gtk.Label()
label.set_markup("<small>Small text</small>");
```

The markup passed to the `set_markup()` method must be valid. For example, the literal <>& characters must be escaped as &lt;, &gt;, and &amp;. If you pass text obtained from the user, file, or a network to the `set_markup()` method, you'll want to escape it with the Python Library `xml.sax.saxutils.escape()` function.

Markup strings are just a convenient way to set the `pango.AttrList` on a label. Using the `set_attributes()` method may be a simpler way to set attributes in some cases. Be careful though; `pango.AttrList` tends to cause internationalization problems, unless you're applying attributes to the entire string because specifying the start_index and end_index for a `pango.Attribute` requires knowledge of the exact string being displayed, so translations will cause problems.

Selectable labels

Labels can be made selectable with the `set_selectable()` method. Selectable labels allow the user to copy the label contents to the clipboard. Only labels that contain useful–to–copy information such as error messages should be made selectable.

Text layout

A label can contain any number of paragraphs, but will have performance problems if it contains more than a small number. Paragraphs are separated by newlines or other paragraph separators understood by Pango.

Labels can automatically wrap text if you call the `set_line_wrap()` method.

The `set_justify()` method sets how the lines in a label align with one another. If you want to set how the label as a whole aligns in its available space, see the `gtk.Misc.set_alignment()` method.

Constructor

```
gtk.Label(str=None)
```

`str` :	The text of the label or `None` for a blank label
Returns :	the new `gtk.Label` widget

Creates a new `gtk.Label` with the text specified by *`str`* inside it. You can pass `None` to get a blank label.

Methods

gtk.Label.set_text

```
def set_text(str)
```

str :	The new text for the label.

The `set_text()` method sets the text within the `gtk.Label` widget. It replaces any text that was there before and will clear any previously set mnemonic accelerators.

gtk.Label.get_text

```
def get_text()
```

Returns :	the text in the label widget.

The `get_text()` method fetches the text from a label widget, as displayed on the screen. This does not include any Pango markup or embedded underscore characters indicating mnemonics. (See `get_label()`).

gtk.Label.set_attributes

```
def set_attributes(attrs)
```

attrs :	a `pango.AttrList`

The `set_attributes()` method applies a `pango.AttrList` list of attributes to the label text. The attributes set with this function will be ignored if either the "use–underline" or "use–markup" attributes is `TRUE`.

gtk.Label.get_attributes

```
def get_attributes()
```

Returns :	the attribute list, or `None` if no attributes were set.

The `get_attributes()` method returns the attribute list that was set on the label using `set_attributes()`, if any. This function does not reflect attributes that come from the labels markup (see `set_markup()`).

gtk.Label.set_label

```
def set_label(str)
```

str :	the new text (including mnemonics or markup) to set for the label

The `set_label()` method sets the text of the label. The label is parsed for embedded underscores and Pango markup depending on the values of the "use–underline" and "use–markup" properties.

gtk.Label.get_label

```
def get_label()
```

Returns :	the text of the label widget.

The `get_label()` method returns the text from a label widget including any Pango markup and embedded underscores indicating mnemonics. (See `get_text()` that just returns the text).

gtk.Label.set_markup

```
def set_markup(str)
```

`str` :	a markup string

The `set_markup()` method parses *`str`*, which is marked up with the Pango text markup language, and sets the label's text and attribute list.

gtk.Label.set_use_markup

```
def set_use_markup(setting)
```

`setting` :	if `TRUE` the label's text should be parsed for markup.

The `set_use_markup()` method sets the "use−markup" property to the value of *`setting`*. If `TRUE` the text of the label should be parsed as markup.

gtk.Label.get_use_markup

```
def get_use_markup()
```

Returns :	`TRUE` if the label's text will be parsed for markup.

The `get_user_markup()` method returns the value of the "use−markup" property. If `TRUE` the label's text is parsed as markup. See `set_use_markup()`.

gtk.Label.set_use_underline

```
def set_use_underline(setting)
```

`setting` :	if `TRUE` underscores in the text indicate mnemonics

The `set_use_underline()` method sets the "use−underline" property to the value of *`setting`*. If *`setting`* is `TRUE`, an underscore in the text indicates the next character should be used for the mnemonic accelerator key.

gtk.Label.get_use_underline

```
def get_use_underline()
```

Returns :	`TRUE` if an embedded underscore in the label indicates the mnemonic accelerator.

The `get_use_underline()` method returns the value of the "use−underline" property. If `TRUE` an embedded underscore in the label indicates the next character is a mnemonic. See `set_use_underline()`.

gtk.Label.set_markup_with_mnemonic

```
def set_markup_with_mnemonic(str)
```

`str` :	a markup string including embedded underscores

The `set_markup_with_mnemonic()` method parses *`str`* as markup, setting the label's text and attribute list based on the parse results. If characters in *`str`* are preceded by an underscore, they are underlined indicating that they represent a mnemonic accelerator. The mnemonic key can be used to activate another widget, chosen automatically, or explicitly using the `set_mnemonic_widget()` method.

gtk.Label.get_mnemonic_keyval

```
def get_mnemonic_keyval()
```

Returns :	a keyval, or the void symbol keyval

The `get_mnemonic_keyval()` method returns the value of the "mnemonic−keyval" property that contains the keyval used for the mnemonic accelerator if one has been set on the label. If there is no mnemonic set up it returns the void symbol keyval.

gtk.Label.set_mnemonic_widget

```
def set_mnemonic_widget(widget)
```

`widget` :	the widget to be activated when the mnemonic is pressed

The `set_mnemonic_widget()` method sets the "mnemonic−widget" property using the value of *`widget`*. This method associates the label mnemonic with a widget that will be activated when the mnemonic accelerator is pressed. When the label is inside a widget (like a `gtk.Button` or a `gtk.Notebook` tab) it is automatically associated with the correct widget, but sometimes (i.e. when the target is a `gtk.Entry` next to the label) you need to set it explicitly using this function. The target widget will be activated by emitting "mnemonic_activate" on it.

gtk.Label.get_mnemonic_widget

```
def get_mnemonic_widget()
```

Returns :	the target of the label's mnemonic, or `None` if none has been set and the default algorithm will be used.

The `get_mnemonic_widget()` method retrieves the value of the "mnemonic−widget" property which is the target of the mnemonic accelerator of this label. See `set_mnemonic_widget()`.

gtk.Label.set_text_with_mnemonic

```
def set_text_with_mnemonic(str)
```

`str` :	the label text with embedded underscore characters indicating the mnemonic characters

The `set_text_with_mnemonic()` method sets the label's text from the string *`str`*. If characters in *`str`* are preceded by an underscore, they are underlined indicating that they represent a mnemonic accelerator. The mnemonic key can be used to activate another widget, chosen automatically, or explicitly using the `set_mnemonic_widget()` method.

gtk.Label.set_justify

```
def set_justify(jtype)
```

`jtype` :	justification type

The `set_justify()` method sets the alignment of the lines in the text of the label relative to each other using the value of *`jtype`*. The possible values of *`jtype`* are: `gtk.JUSTIFY_LEFT`, `gtk.JUSTIFY_RIGHT`, `gtk.JUSTIFY_CENTER` and `gtk.JUSTIFY_FILL`. `gtk.JUSTIFY_LEFT` is the default value when the widget is first created. If you want to set the alignment of the label as a whole, use the `gtk.Misc.set_alignment()` method instead. The `set_justify()` has no effect on labels containing only a single line.

gtk.Label.get_justify

```
def get_justify()
```

Returns :	the label justification

The `get_justify()` method returns the justification of the label; one of: `gtk.JUSTIFY_LEFT`, `gtk.JUSTIFY_RIGHT`, `gtk.JUSTIFY_CENTER` or `gtk.JUSTIFY_FILL`. See `set_justify()`.

gtk.Label.set_pattern

```
def set_pattern(pattern)
```

`pattern` :	the pattern of underlines

The `set_pattern()` method sets the "pattern" property with the value of *`pattern`*. The pattern contains an underscore or space for each character in the label text. Any characters omitted are assumed to be spaces. For example, if the label text is "XXX Label" and the pattern is "___" then only the "XXX" will be underlined.

gtk.Label.set_line_wrap

```
def set_line_wrap(wrap)
```

`wrap` :	if TRUE the label lines will wrap if too big for the widget size.

The `set_wrap()` method sets the "wrap" property tot he value of *`wrap`*. If *`wrap`* is `TRUE` the label text will wrap if it is wider than the widget size; otherwise, the text gets cut off at the edge of the widget.

gtk.Label.get_line_wrap

```
def get_line_wrap()
```

Returns :	`TRUE` if the lines of the label are automatically wrapped.

The `get_line_wrap()` method returns the value of the "wrap" property. If "wrap" is `TRUE` the lines in the label are automatically wrapped. See `set_line_wrap()`.

gtk.Label.set_selectable

```
def set_selectable(setting)
```

`setting` :	if `TRUE` allow the text in the label to be selected

The `set_selectable()` method sets the "selectable" property with the value of *`setting`*. If *`setting`* is `TRUE` the user is allowed to select text from the label, for copy–and–paste.

gtk.Label.get_selectable

```
def get_selectable()
```

Returns :	`TRUE` if the user can select the label text

The `get_selectable()` method gets the value of the "selectable" property set by the `set_selectable()` method.

gtk.Label.select_region

```
def select_region(start_offset, end_offset)
```

start_offset :	start offset in characters
end_offset :	end offset in characters

The `select_region()` method selects a range of characters in the label, if the label is selectable. The selected region is the range of characters between *start_offset* and *end_offset*. See `set_selectable()`. If the label is not selectable, this method has no effect. If *start_offset* or *end_offset* are −1, then the end of the label will be substituted.

gtk.Label.get_selection_bounds

```
def get_selection_bounds()
```

Returns :	a tuple containing the start and end character offsets of the selection

The `get_selection_bounds()` method returns a tuple that contains the start and end character offsets of the selected text in the label if the selection exists. If there is no selection or the label is not selectable, an empty tuple is returned.

gtk.Label.get_layout

```
def get_layout()
```

Returns :	the `pango.Layout` for this label

The `get_layout()` method returns the `pango.Layout` used to display the label. The layout is useful to e.g. convert text positions to pixel positions, in combination with `get_layout_offsets()`.

gtk.Label.get_layout_offsets

```
def get_layout_offsets()
```

Returns :	a tuple containing the X offset of the layout, or `None` and the Y offset of layout, or `None`

The `get_layout_offsets()` method returns a tuple containing the coordinates where the label will draw the `pango.Layout` representing the text in the label. This method is useful for converting mouse events into coordinates inside the `pango.Layout`, e.g. to take some action if some part of the label is clicked. Of course you will need to create a `gtk.EventBox` to receive the events, and pack the label inside it, since labels are a "windowless" (`gtk.NO_WINDOW`) widget. Remember when using the `pango.Layout` functions you need to convert to and from pixels using `pango.PIXELS()` or `pango.SCALE`.

gtk.Label.set_ellipsize

```
def set_ellipsize(mode)
```

mode :	one of the Pango Ellipsize Mode Constants to use

Note

This method is available in PyGTK 2.6 and above.

The `set_ellipsize()` method sets the "ellipsize" property to the value of *mode*. *mode* should be one of the Pango Ellipsize Mode Constants. The "ellipsize" property specifies if and where an ellipse should be used if there is not enough room for the label text.

gtk.Label.get_ellipsize

```
  def get_ellipsize()
```

Returns :	the current ellipsize mode

Note

This method is available in PyGTK 2.6 and above.

The `get_ellipsize()` method returns the value of the "ellipsize" property which contains one of the Pango Ellipsize Mode Constants. The "ellipsize" property specifies if and where an ellipse should be used if there is not enough room for the label text.

gtk.Label.set_width_chars

```
  def set_width_chars(n_chars)
```

`n_chars` :	the new desired width, in characters.

Note

This method is available in PyGTK 2.6 and above.

The `set_width_chars()` method sets the "width–chars" property to the value of *`n_chars`*. The "width–chars" property specifies the desired width of the label in characters.

gtk.Label.get_width_chars

```
  def get_width_chars()
```

Returns :	the desired width of the label in characters.

Note

This method is available in PyGTK 2.6 and above.

The `get_width_chars()` method returns the value of the "width–chars" property that specifies the desired width of the label in characters.

gtk.Label.set_single_line_mode

```
  def set_single_line_mode(single_line_mode)
```

`single_line_mode` :	if `TRUE` the label is in single line mode.

Note

This method is available in PyGTK 2.6 and above.

The `set_single_line_mode()` method sets the "single–line–mode" property to the value of *`single_line_mode`*. If *`single_line_mode`* is `TRUE` the label is in single line mode where the height of the label does not depend on the actual text, it is always set to ascent + descent of the font.

gtk.Label.get_single_line_mode

```
def get_single_line_mode()
```

Returns :

Note

This method is available in PyGTK 2.6 and above.

The `get_single_line_mode`() method returns the value of the "single−line−mode" property. See the `set_single_line_mode()` method for more information.

gtk.Label.set_max_width_chars

```
def set_max_width_chars(n_chars)
```

`n_chars` :	the new desired maximum width, in characters.

Note

This method is available in PyGTK 2.6 and above.

The `set_max_width_chars`() method sets the "max−width−chars" property to the value of *`n_chars`*.

gtk.Label.get_max_width_chars

```
def get_max_width_chars()
```

Returns :

Note

This method is available in PyGTK 2.6 and above.

The `get_max_width_chars`() method returns the value of the "max−width−chars" property which is the desired maximum width of the label in characters.

gtk.Label.set_angle

```
def set_angle(angle)
```

`angle` :	the angle that the baseline of the label makes with the horizontal, in degrees, measured counterclockwise

Note

This method is available in PyGTK 2.6 and above.

The `set_angle`() method sets the "angle" property to the value of *`angle`*. *`angle`* is the angle of rotation for the label. An angle of 90 reads from from bottom to top, an angle of 270, from top to bottom. The angle setting for the label is ignored if the label is selectable, wrapped, or ellipsized.

gtk.Label.get_angle

```
def get_angle()
```

Returns :	

Note

This method is available in PyGTK 2.6 and above.

The `get_angle`() method returns the value of the "angle" property. See the `set_angle()` method for more information.

Signals

The "copy−clipboard" gtk.Label Signal

```
def callback(label, user_param1, ...)
```

`label` :	the label that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "copy−clipboard" signal is emitted when text is copied from the label to the clipboard.

The "move−cursor" gtk.Label Signal

```
def callback(label, step, count, extend_selection, user_param1, ...)
```

`label` :	the label that received the signal
`step` :	the step size of the move: `gtk.MOVEMENT_LOGICAL_POSITIONS`, `gtk.MOVEMENT_VISUAL_POSITIONS`, `gtk.MOVEMENT_WORDS`, `gtk.MOVEMENT_DISPLAY_LINES`, `gtk.MOVEMENT_DISPLAY_LINE_ENDS`, `gtk.MOVEMENT_PARAGRAPHS`, `gtk.MOVEMENT_PARAGRAPH_ENDS`, `gtk.MOVEMENT_PAGES` and `gtk.MOVEMENT_BUFFER_ENDS`
`count` :	the number of steps to take
`extend_selection` :	if `TRUE` extend the range of the selection
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "move−cursor" signal is emitted when the cursor is being moved *`count`* steps or size `step`. The step size is one of:

```
gtk.MOVEMENT_LOGICAL_POSITIONS,    move by graphemes
gtk.MOVEMENT_VISUAL_POSITIONS,     move by graphemes
gtk.MOVEMENT_WORDS,                move by words
gtk.MOVEMENT_DISPLAY_LINES,        move by lines(wrapped lines)
gtk.MOVEMENT_DISPLAY_LINE_ENDS,    move to line ends(wrapped lines)
gtk.MOVEMENT_PARAGRAPHS,           move by paragraphs(newline-ended lines)
gtk.MOVEMENT_PARAGRAPH_ENDS,       move to ends of a paragraph
gtk.MOVEMENT_PAGES,                move by pages
gtk.MOVEMENT_BUFFER_ENDS           move to ends of the buffer
```

If *`extend_selection`* is `TRUE` the selection will be extended to include the text moved over.

The "populate−popup" gtk.Label Signal

```
  def callback(label, menu, user_param1, ...)
```

label :	the label that received the signal
menu :	the menu to be populated
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "populate−popup" signal is emitted when a menu needs to be populated on the fly.

Prev Up Next
gtk.ItemFactory Home gtk.Layout
gtk.Layout
Prev The gtk Class Reference Next

gtk.Layout

gtk.Layout infinite scrollable area containing child widgets and custom drawing

Synopsis

```
class gtk.Layout(gtk.Container):
    gtk.Layout(hadjustment=None, vadjustment=None)
    def put(child_widget, x, y)
    def move(child_widget, x, y)
    def set_size(width, height)
    def get_size()
    def get_hadjustment()
    def get_vadjustment()
    def set_hadjustment(adjustment)
    def set_vadjustment(adjustment)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Layout
```

Properties

"hadjustment"	Read−Write	The gtk.Adjustment for the horizontal position.
"vadjustment"	Read−Write	The gtk.Adjustment for the vertical position.
"width"	Read−Write	the layout width
"height"	Read−Write	the layout height

Child Properties

"x"	Read−Write	the X position of the child

"y"	Read−Write	the Y position of the child

Attributes

"bin_window"	Read	the window of a layout to draw into

Signal Prototypes

"set−scroll−adjustments"	`def callback(`*`layout, hadjustment, vadjustment, user_param1, ...`*`)`

Description

The `gtk.Layout` is a simple container widget similar to the `gtk.Fixed` container widget. Like the `gtk.Fixed` the `gtk.Layout` places a child widget at a specific position within the container. The `gtk.Layout` offers two features beyond the `gtk.Fixed` widget:

- a very large width and height for the container − limited by the size of an unsigned integer.
- horizontal and vertical adjustments can be specified for use with scrollbars, etc.

The `gtk.Layout` can also be drawn on similar to drawing on a `gtk.DrawingArea`. When handling expose events on a `gtk.Layout`, you must draw to the window specified by the `bin_window` attribute rather than the widget `window` attribute.

Constructor

```
    gtk.Layout(hadjustment=None, vadjustment=None)
```

`hadjustment` :	horizontal adjustment, or `None`
`vadjustment` :	vertical adjustment, or `None`
Returns :	a new `gtk.Layout`

Creates a new `gtk.Layout`. Usually the *`hadjustment`* and *`vadjustment`* arguments are not specified or are specified as `None`, so that new adjustments are created.

Methods

gtk.Layout.put

```
    def put(child_widget, x, y)
```

`child_widget` :	the child widget
`x` :	the X position of child widget
`y` :	the Y position of child widget

The `put`() method adds *`child_widget`* to the layout and places its upper, left corner at the position specified by *`x`* and *`y`*.

gtk.Layout.move

```
    def move(child_widget, x, y)
```

child_widget :	a current child of the layout
x :	the X position to move *child_widget* to
y :	the Y position to move *child_widget* to

The move() method moves a current child of the layout (specified by *child_widget*) to the new position specified by *x* and *y*. The upper, left corner of *child_widget* will be placed at (*x*, *y*).

gtk.Layout.set_size

```
    def set_size(width, height)
```

width :	width of the layout area
height :	height of the layout area

The set_size() method sets the size of the virtual area of the layout to the values specified by *width* and *height*. The "width" and "height" properties are also set by this method.

gtk.Layout.get_size

```
    def get_size()
```

Returns :	a tuple containing the width and height set on *layout*

The get_size() method returns a tuple that contains the width and height of the virtual size that has been set on the layout. See set_size().

gtk.Layout.get_hadjustment

```
    def get_hadjustment()
```

Returns :	a horizontal adjustment

The get_hadjustment() method returns the value of the "hadjustment" property that contains the horizontal adjustment object associated with the layout. This function should only be called after the layout has been placed in a gtk.ScrolledWindow or has otherwise been configured for scrolling. See gtk.ScrolledWindow, gtk.Scrollbar, gtk.Adjustment for details.

gtk.Layout.get_vadjustment

```
    def get_vadjustment()
```

Returns :	a vertical adjustment

The get_vadjustment() method returns the "vadjustment" property that contains the vertical adjustment object associated with the layout. This function should only be called after the layout has been placed in a gtk.ScrolledWindow or has otherwise been configured for scrolling. See gtk.ScrolledWindow, gtk.Scrollbar, gtk.Adjustment for details.

gtk.Layout.set_hadjustment

```
    def set_hadjustment(adjustment)
```

adjustment :	a horizontal adjustment

The `set_hadjustment()` method sets the horizontal adjustment for the layout (and the "hadjustment" property) to the value of *`adjustment`*. See `gtk.ScrolledWindow`, `gtk.Scrollbar`, `gtk.Adjustment` for details.

gtk.Layout.set_vadjustment

```
    def set_vadjustment(adjustment)
```

`adjustment` :	a vertical adjustment

The `set_vadjustment()` method sets the vertical adjustment for the layout (and the "vadjustment" property) to the value of *`adjustment`*. See `gtk.ScrolledWindow`, `gtk.Scrollbar`, `gtk.Adjustment` for details.

Signals

The "set–scroll–adjustments" gtk.Layout Signal

```
    def callback(layout, hadjustment, vadjustment, user_param1, ...)
```

`layout` :	the layout that received the signal
`hadjustment` :	the horizontal adjustment associated with the layout.
`vadjustment` :	the horizontal adjustment associated with the layout.
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "set–scroll–adjustments" signal is emitted when one of the adjustments associated with a layout is changed.

Prev	Up	Next
gtk.Label	Home	gtk.ListStore

gtk.ListStore

gtk.ListStore a list model to use with a `gtk.TreeView`

Synopsis

```
class gtk.ListStore(gobject.GObject, gtk.TreeModel, gtk.TreeDragSource, gtk.TreeDragDest, gtk.T
    gtk.ListStore(column_type, ...)
    def set_column_types(type, ...)
    def set_value(iter, column, value)
    def set(iter, column_num, value, ...)
    def remove(iter)
    def insert(position, row=None)
    def insert_before(sibling, row=None)
    def insert_after(sibling, row=None)
    def prepend(row=None)
    def append(row=None)
    def clear()
    def iter_is_valid(iter)
    def reorder(new_order)
    def swap(a, b)
    def move_after(iter, position)
    def move_before(iter, position)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.ListStore (implements gtk.TreeModel, gtk.TreeDragSource, gtk.TreeDragDest, gtk.TreeSc
```

Description

The `gtk.ListStore` object is a list model for use with a `gtk.TreeView` widget. It implements the `gtk.TreeModel` interface, the `gtk.TreeSortable` and the tree drag and drop interfaces.

Constructor

```
  gtk.ListStore(column_type, ...)
```

`column_type` :	the column type of the first column
`...` :	optional types for succeeding columns
Returns :	a new `gtk.ListStore`

Creates a new list store as with one or more columns with the type specified by the arguments passed to the constructor. For example, `gtk.ListStore(gobject.TYPE_INT, gobject.TYPE_STRING, gtk.gdk.Pixbuf);` will create a new `gtk.ListStore` with three columns, of type int, string and `gtk.gdk.Pixbuf` respectively. The built–in `GObject` types are:

- `gobject.TYPE_BOOLEAN`
- `gobject.TYPE_BOXED`
- `gobject.TYPE_CHAR`
- `gobject.TYPE_DOUBLE`
- `gobject.TYPE_ENUM`
- `gobject.TYPE_FLAGS`
- `gobject.TYPE_FLOAT`
- `gobject.TYPE_INT`
- `gobject.TYPE_INT64`
- `gobject.TYPE_INTERFACE`
- `gobject.TYPE_INVALID`
- `gobject.TYPE_LONG`
- `gobject.TYPE_NONE`
- `gobject.TYPE_OBJECT`
- `gobject.TYPE_PARAM`
- `gobject.TYPE_POINTER`
- `gobject.TYPE_PYOBJECT`
- `gobject.TYPE_STRING`
- `gobject.TYPE_UCHAR`
- `gobject.TYPE_UINT`
- `gobject.TYPE_UINT64`
- `gobject.TYPE_ULONG`

The column types can be any `GObject` type including those that are PyGTK objects or application defined objects that are subclassed from the `GObject` class.

Methods

gtk.ListStore.set_column_types

```
def set_column_types(type, ...)
```

type :	the type of the first column
... :	zero or more type specifications

Note

This method is available in PyGTK 2.2 and above.

The `set_column_types()` method sets the liststore columns to the types specified by *type* and any additional type parameters. This method is meant primarily for classes that inherit from `gtk.ListStore`, and should only be used when constructing a new `gtk.ListStore`. It will not function after a row has been added, or a method on the `gtk.TreeModel` interface is called.

gtk.ListStore.set_value

```
def set_value(iter, column, value)
```

iter :	a valid `gtk.TreeIter` for the row being modified
column :	the column number to modify
value :	the new value for the cell

The `set_value()` method sets the data in the cell specified by *iter* and *column*. The type of *value* must be convertible to the type of the column.

gtk.ListStore.set

```
def set(iter, column_num, value, ...)
```

iter :	A valid `gtk.TreeIter` for the row being modified
column_num :	the number of the column to modify
value :	the new cell value
... :	additional optional sets of column number – value pairs

The `set()` method sets the value of one or more cells in the row referenced by *iter*. The argument list should contain integer column numbers, each followed by the value to be set (the value must be convertible to the type of the cell column). For example, to set column 0 with type `gobject.TYPE_STRING` to "Foo", you would write:

```
liststore.set(iter, 0, "Foo")
```

gtk.ListStore.remove

```
def remove(iter)
```

iter :	A valid `gtk.TreeIter` for the row
Returns :	`TRUE` if *iter* is still valid.

The `remove()` method removes the row specified by *iter* from the list store and returns TRUE if *iter* is still valid. After being removed, *iter* is set to be the next valid row, or is invalidated if it pointed to the last row.

Note

Prior to PyGTK 2.4 this method returned a new gtk.TreeIter that is a copy of *iter*.

gtk.ListStore.insert

```
def insert(position, row=None)
```

position :	the integer position to insert the new row
row :	an optional list or tuple containing ordered column values to set on the row or None
Returns :	A gtk.TreeIter pointing at the new row

The insert() method creates a new row at the location specified by *position*. If *position* is larger than the number of rows on the list, then the new row will be appended to the list. The row will be empty if *row* is not specified or is None. If *row* is specified it must contain a list or tuple of ordered column values (e.g. [gobject.TYPE_STRING, gobject.TYPE_INT]) that are used to set the values in the cells of the new row. Alternatively, the application can fill in row cell values using the set() or set_value() methods.

gtk.ListStore.insert_before

```
def insert_before(sibling, row=None)
```

sibling :	A valid gtk.TreeIter or None
row :	an optional list or tuple containing ordered column values to set on the row or None
Returns :	A gtk.TreeIter pointing at the new row

The insert_before() method inserts a new row before the row specified by the gtk.TreeIter *sibling*. The row will be empty if *row* is not specified or is None. If *row* is specified it must contain a list or tuple of ordered column values (e.g. [gobject.TYPE_STRING, gobject.TYPE_INT]) that are used to set the values in the cells of the new row. Alternatively, the application can fill in row cell values using the set() or set_value() methods.

In PyGTK 2.4, if *sibling* is None the row will be appended to the liststore.

gtk.ListStore.insert_after

```
def insert_after(sibling, row=None)
```

sibling :	A valid gtk.TreeIter or None
row :	an optional list or tuple containing ordered column values to set on the row or None
Returns :	A gtk.TreeIter pointing at the new row

The insert_after() method inserts a new row after the row specified by the gtk.TreeIter *sibling*. The row will be empty if *row* is not specified or is None. If *row* is specified it must contain a list or tuple of ordered column values (e.g. [gobject.TYPE_STRING, gobject.TYPE_INT]) that are used to set the values in the cells of the new row. Alternatively, the application can fill in row cell values using the set() or set_value() methods.

In PyGTK 2.4, if *sibling* is None the row will be prepended to the liststore.

gtk.ListStore.prepend

```
def prepend(row=None)
```

row :	an optional list or tuple containing ordered column values to set on the row or `None`
Returns :	A `gtk.TreeIter` pointing at the new row

The `prepend()` method prepends a new row to the liststore. The row will be empty if *`row`* is not specified or is `None`. If *`row`* is specified it must contain a list or tuple of ordered column values (e.g. `[gobject.TYPE_STRING, gobject.TYPE_INT]`) that are used to set the values in the cells of the new row. Alternatively, the application can fill in row cell values using the `set()` or `set_value()` methods.

gtk.ListStore.append

```
def append(row=None)
```

row :	an optional list or tuple containing ordered column values to set on the row or `None`
Returns :	A `gtk.TreeIter` pointing at the new row

The `append()` method appends a new row to the liststore. The row will be empty if *`row`* is not specified or is `None`. If *`row`* is specified it must contain a list or tuple of ordered column values (e.g. `[gobject.TYPE_STRING, gobject.TYPE_INT]`) that are used to set the values in the cells of the new row. Alternatively, the application can fill in row cell values using the `set()` or `set_value()` methods.

gtk.ListStore.clear

```
def clear()
```

The `clear()` method removes all rows from the liststore.

gtk.ListStore.iter_is_valid

```
def iter_is_valid(iter)
```

iter :	A `gtk.TreeIter`.
Returns :	`TRUE` if the iter is valid, `FALSE` if the iter is invalid.

Note

This method is available in PyGTK 2.2 and above.

Warning

This method is slow. Only use it for debugging and/or testing purposes.

The `iter_is_valid()` method checks if the `gtk.TreeIter` specified by *`iter`* is a valid iter for this `gtk.ListStore`.

gtk.ListStore.reorder

```
def reorder(new_order)
```

`new_order` :	a list of integers mapping the new position of each child to its old position before the re–ordering, i.e. *`new_order`*`[newpos] = oldpos`.

Note

This method is available in PyGTK 2.2 and above.

The `reorder()` method reorders the `gtk.ListStore` items to follow the order indicated by *`new_order`*. Note that this method only works with unsorted stores.

gtk.ListStore.swap

```
    def swap(a, b)
```

`a` :	A `gtk.TreeIter`.
`b` :	Another `gtk.TreeIter`.

Note

This method is available in PyGTK 2.2 and above.

The `swap()` method swaps the liststore rows specified by the `gtk.TreeIter`s *`a`* and *`b`*. Note that this method only works with unsorted stores.

gtk.ListStore.move_after

```
    def move_after(iter, position)
```

`iter` :	A `gtk.TreeIter`.
`position` :	A `gtk.TreeIter` or `None`.

Note

This method is available in PyGTK 2.2 and above.

The `move_after()` method moves the liststore row referenced by *`iter`* to the position after the row referenced by *`position`*. Note that this method only works with unsorted stores. If *`position`* is `None`, the row referenced by *`iter`* will be moved to the start of the list.

gtk.ListStore.move_before

```
    def move_before(iter, position)
```

`iter` :	A `gtk.TreeIter`.
`position` :	A `gtk.TreeIter`, or `None`.

Note

This method is available in PyGTK 2.2 and above.

The `move_before()` method moves the liststore row referenced by *`iter`* to the position before the row referenced by *`position`*. Note that this method only works with unsorted stores. If *`position`* is `None`, the row referenced by *`iter`* will be moved to the end of the list.

Prev Up Next
gtk.Layout Home gtk.Menu

gtk.Menu

Prev **The gtk Class Reference** Next

gtk.Menu

gtk.Menu — a drop down menu widget.

Synopsis

```
class gtk.Menu(gtk.MenuShell):
    gtk.Menu()
    def popup(parent_menu_shell, parent_menu_item, func, button, activate_time)
    def reposition()
    def popdown()
    def get_active()
    def set_active(index)
    def set_accel_group(accel_group)
    def get_accel_group()
    def set_accel_path(accel_path)
    def attach_to_widget(attach_widget, detach_func)
    def detach()
    def get_attach_widget()
    def set_tearoff_state(torn_off)
    def get_tearoff_state()
    def set_title(title)
    def get_title()
    def reorder_child(child, position)
    def set_screen(screen)
    def attach(child, left_attach, right_attach, top_attach, bottom_attach)
    def set_monitor(monitor_num)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.MenuShell
          +-- gtk.Menu
```

Properties

"tearoff−state"	Read−Write	If `TRUE` the menu is torn−off. Default value: `FALSE`. Available in GTK+ 2.6 and above.
"tearoff−title"	Read−Write	A title that may be displayed by the window manager when this menu is torn−off. Default value: "".

Child Properties

Note

These properties are available in GTK+ 2.4 and above.

"bottom−attach"	Read−Write	The row number to attach the bottom of the child to. Allowed values: >= −1. Default value: −1.
"left−attach"	Read−Write	The column number to attach the left side of the child to. Allowed values: >= −1. Default value: −1.

"right−attach"	Read−Write	The column number to attach the right side of the child to. Allowed values: >= −1. Default value: −1.
"top−attach"	Read−Write	The row number to attach the top of the child to. Allowed values: >= −1. Default value: −1.

Style Properties

Note

These properties are available in GTK+ 2.4 and above.

"horizontal−offset"	Read	When the menu is a submenu, position it this number of pixels offset horizontally. Default value: −2.
"vertical−offset"	Read	When the menu is a submenu, position it this number of pixels offset vertically. Default value: 0.
"vertical−padding"	Read	Extra space at the top and bottom of the menu. Allowed values: >= 0. Default value: 1.

Signal Prototypes

"move−scroll"	`def callback(menu, type, user_param1, ...)`

Description

A `gtk.Menu` is a `gtk.MenuShell` that implements a drop down menu consisting of a list of `gtk.MenuItem` objects which can be navigated and activated by the user to perform application functions. A `gtk.Menu` is most commonly dropped down by activating a `gtk.MenuItem` in a `gtk.MenuBar` or popped up by activating a `gtk.MenuItem` in another `gtk.Menu`. A `gtk.Menu` can also be popped up by activating a `gtk.OptionMenu`. Other composite widgets such as the `gtk.Notebook` can pop up a `gtk.Menu` as well. Applications can display a `gtk.Menu` as a popup menu by calling the popup() method.

Constructor

```
gtk.Menu()
```

Returns :	a `gtk.Menu` widget

Creates a new `gtk.Menu` widget.

Methods

gtk.Menu.popup

```
def popup(parent_menu_shell, parent_menu_item, func, button, activate_time)
```

`parent_menu_shell` :	the menu shell containing the triggering menu item or `None`.
`parent_menu_item` :	the menu item whose activation triggered the popup or `None`.
`func` :	a user supplied function used to position the menu or `None`.
`button` :	the mouse button which was pressed to initiate the event.

`activate_time` :	the time at which the activation event occurred.

The `popup`() method displays a menu and makes it available for selection. Applications can use this function to display context–sensitive menus, and will typically supply `None` for the *`parent_menu_shell`*, *`parent_menu_item`* and *`func`* parameters. The default menu positioning function will position the menu at the current pointer position. The *`button`* and *`activate_time`* values should be the mouse button that was pressed to trigger the menu popup and the time the button was pressed. These values can usually be retrieved from the "button_press_event".

gtk.Menu.reposition

```
    def reposition()
```

The `reposition`() method repositions the menu on the screen according to the internal position function.

gtk.Menu.popdown

```
    def popdown()
```

The `popdown`() method removes the menu from the screen.

gtk.Menu.get_active

```
    def get_active()
```

Returns :	the `gtk.MenuItem` that was last selected in the menu. If a selection has not yet been made, the first menu item is selected.

The `get_active`() method returns the selected menu item from the menu. This is used by the `gtk.OptionMenu`.

gtk.Menu.set_active

```
    def set_active(index)
```

`index` :	the index of the menu item to select. Index values start from 0.

The `set_active`() method selects the menu item within the menu at the location specified by *`index`*. This is used by the `gtk.OptionMenu` and is not useful for applications.

gtk.Menu.set_accel_group

```
    def set_accel_group(accel_group)
```

`accel_group` :	a `gtk.AccelGroup`

The `set_accel_group`() method associates the `gtk.AccelGroup` specified by *`accel_group`* with the menu. The accelerator group should also be added to all windows using this menu by calling the `gtk.Window.add_accel_group()` method.

gtk.Menu.get_accel_group

```
    def get_accel_group()
```

Returns :	the `gtk.AccelGroup` associated with the menu.

The `get_accel_group()` method returns the `gtk.AccelGroup` that holds the global accelerators for the menu.

gtk.Menu.set_accel_path

```
def set_accel_path(accel_path)
```

`accel_path` :	a valid accelerator path

The `set_accel_path()` method sets an accelerator path (specified by *`accel_path`*) for this menu to be used to construct accelerator paths for its menu items. This is a convenience method used to avoid calling the `gtk.MenuItem.set_accel_path()` method on each menu item that should support runtime user changeable accelerators. Instead, by just calling `set_accel_path()` on their parent, each menu item of this menu, that contains a label describing its purpose, automatically gets an accel path assigned. For example, calling:

```
menu.set_accel_path("<main>/File")
```

for a menu containing menu items "New" and "Exit", will assign its items the accel paths: `"<main>/File/New"` and `"<main>/File/Exit"`. Assigning accel paths to menu items enables the user to change their accelerators at runtime.

gtk.Menu.attach_to_widget

```
def attach_to_widget(attach_widget, detach_func)
```

`attach_widget` :	the widget that the menu will be attached to.
`detach_func` :	the user supplied callback function that will be called when the menu calls the `detach()` method.

The `attach_to_widget()` method attaches the menu to the widget specified by *`attach_widget`* and provides a callback function specified by *`detach_func`* that will be invoked when the menu calls the `detach()` method during its destruction.

gtk.Menu.detach

```
def detach()
```

The `detach()` method detaches the menu from the widget to which it had been attached. See `attach_to_widget()`().

gtk.Menu.get_attach_widget

```
def get_attach_widget()
```

Returns :	the widget that the menu is attached to.

The `get_attach_widget()` method returns the `gtk.Widget` that the menu is attached to.

gtk.Menu.set_tearoff_state

```
def set_tearoff_state(torn_off)
```

`torn_off` :	If `TRUE`, the menu is displayed as a tearoff menu.

The `set_tearoff_state()` method sets the tearoff state of the menu to the value of *`torn_off`*. If *`torn_off`* is `TRUE` the menu is displayed as a tearoff menu; if *`torn_off`* is `FALSE` the menu is displayed as a drop down menu which persists as long as the menu is active.

gtk.Menu.get_tearoff_state

```
def get_tearoff_state()
```

Returns :	`TRUE` if the menu is currently torn off.

The `get_tearoff_state()` method returns whether the menu is torn off. See `set_tearoff_state()`.

gtk.Menu.set_title

```
def set_title(title)
```

`title` :	a string containing the title for the menu.

The `set_title()` method sets the title text (from the value of *`title`*) to be used for the menu when it is shown as a tearoff menu.

gtk.Menu.get_title

```
def get_title()
```

Returns :	the title of the menu, or `None` if the menu has no title set on it.

The `get_title()` method returns the title of the menu or `None` of no title is set. See `set_title()`.

gtk.Menu.reorder_child

```
def reorder_child(child, position)
```

`child` :	the `gtk.MenuItem` to move.
`position` :	the new position to place child. Positions are numbered starting from 0

The `reorder_child()` method moves the menuitem specified by *`child`* to a new position within the menu specified by *`position`*.

gtk.Menu.set_screen

```
def set_screen(screen)
```

`screen` :	a `gtk.gdk.Screen`, or `None` if the screen should be determined by the widget the menu is attached to.

Note

This method is available in PyGTK 2.2 and above.

The `set_screen()` method sets the `gtk.gdk.Screen` specified by *`screen`* on which the menu will be displayed. If *`screen`* is `None` the screen is determined by the widget that the menu is attached to.

gtk.Menu.attach

```
def attach(child, left_attach, right_attach, top_attach, bottom_attach)
```

`child` :	a `gtk.MenuItem`.

`left_attach` :	The column number to attach the left side of the item to.
`right_attach` :	The column number to attach the right side of the item to.
`top_attach` :	The row number to attach the top of the item to.
`bottom_attach` :	The row number to attach the bottom of the item to.

Note

This method is available in PyGTK 2.4 and above.

The `attach()` method adds a new `gtk.MenuItem` specified by *`child`* to a (table) menu. The number of 'cells' that an item will occupy is specified by *`left_attach`*, *`right_attach`*, *`top_attach`* and *`bottom_attach`*. These each represent the leftmost, rightmost, uppermost and lower column and row numbers of the table. (Columns and rows are indexed from zero).

Note that this function is not related to the `detach()` method.

gtk.Menu.set_monitor

```
    def set_monitor(monitor_num)
```

`monitor_num` :	the number of the monitor on which the menu should be popped up

Note

This method is available in PyGTK 2.4 and above.

The `set_monitor()` method informs `GTK+` on which monitor a menu should be popped up. See the `gtk.gdk.Screen.get_monitor_geometry()` method for more information.

This method should be called from a menu positioning function if the menu should not appear on the same monitor as the pointer. This information can't be reliably inferred from the coordinates returned by a menu positioning function, since, for very long menus, these coordinates may extend beyond the monitor boundaries or even the screen boundaries.

Signals

The "move_scroll" gtk.Menu Signal

```
    def callback(menu, type, user_param1, ...)
```

`menu` :	the menu that received the signal
`type` :	the type of scroll that is requested
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.2 and above.

The "move_scroll" signal is emitted when the user attempts to scroll the menu. *`type`* should be one of the GTK Scroll Step Constants.

Prev Up Next

gtk.MenuBar

gtk.MenuBar a widget that displays gtk.MenuItem widgets horizontally

Synopsis

```
class gtk.MenuBar(gtk.MenuShell):
    gtk.MenuBar()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.MenuShell
          +-- gtk.MenuBar
```

Style Properties

"shadow–type"	Read	The style of bevel around the menubar
"internal–padding"	Read	Amount of border space between the menubar shadow and the menu items

Description

The gtk.MenuBar is a subclass of gtk.MenuShell which contains one or more gtk.MenuItem widgets. A gtk.MenuBar displays the menu items horizontally in an application window or dialog.

Constructor

```
  gtk.MenuBar()
```

Returns :	a new gtk.MenuBar widget

Creates a new gtk.MenuBar widget.

Prev	Up	Next
gtk.Menu	Home	gtk.MenuItem

gtk.MenuItem

gtk.MenuItem the widget used for an item in menus

Synopsis

```
class gtk.MenuItem(gtk.Item):
    gtk.MenuItem(label=None, use_underline=TRUE)
    def set_submenu(submenu)
    def get_submenu()
    def remove_submenu()
    def select()
    def deselect()
    def activate()
    def toggle_size_request()
    def toggle_size_allocate(allocation)
    def set_right_justified(right_justified)
    def get_right_justified()
    def set_accel_path(accel_path)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Item
            +-- gtk.MenuItem
```

Style Properties

"selected–shadow–type"	Read	The shadow type when the item is selected

Signal Prototypes

"activate"	`def callback(menuitem, user_param1, ...)`
"activate–item"	`def callback(menuitem, user_param1, ...)`
"toggle–size–allocate"	`def callback(menuitem, allocation, user_param1, ...)`
"toggle–size–request"	`def callback(menuitem, requisition, user_param1, ...)`

Description

The `gtk.MenuItem` widget implements the appearance and behavior of menu items. The `gtk.MenuItem` and its derived widget subclasses are the only valid children of menus.

When menu items are selected and activated by a user they can:

- display a popup menu if they have an associated submenu
- invoke an associated function or method

As a `gtk.MenuItem` is a subclass of `gtk.Bin` it can hold any valid child widget.

Constructor

```
gtk.MenuItem(label=None, use_underline=TRUE)
```

`label` :	a string to be used as the text of the menu item or `None`
`use_underline` :	if `TRUE`, an underscore in the label text indicates the next character should be underlined and used for the mnemonic accelerator key if it is the first character so marked. Available in PyGTK 2.4 and above.
Returns :	a new `gtk.MenuItem` widget

Creates a new `gtk.MenuItem` widget and sets the text of the menu item label to the value of *`label`*. If *`label`* is `None` no label is created for the menu item. The text of label is parsed for underscore characters that indicate that the next character is a mnemonic accelerator.

In PyGTK 2.4 and above the *`use_underline`* parameter is available and defaults to `TRUE`. If *`use_underline`* is set to `FALSE` the label text will not be parsed for mnemonic characters.

Methods

gtk.MenuItem.set_submenu

```
    def set_submenu(submenu)
```

`submenu` :	the menu to use as the submenu

The `set_submenu()` method sets the menu specified by *`submenu`* as the submenu for the menu item.

gtk.MenuItem.get_submenu

```
    def get_submenu()
```

Returns :	the submenu for this menu item, or `None` if there is no submenu.

The `get_submenu()` method returns the submenu widget associated with this menu item. If there is no submenu this method returns `None`. See `set_submenu()`.

gtk.MenuItem.remove_submenu

```
    def remove_submenu()
```

The `remove_submenu()` method removes the submenu associated with the menu item.

gtk.MenuItem.select

```
    def select()
```

The `select()` method emits the "select" signal on the menu item.

gtk.MenuItem.deselect

```
    def deselect()
```

The `deselect()` method emits the "deselect" signal on the menu item.

gtk.MenuItem.activate

```
    def activate()
```

The `activate`() method emits the "activate" signal on the menu item.

gtk.MenuItem.toggle_size_request

```
    def toggle_size_request()
```

Returns :	the size requisition

Note

This method is available in PyGTK 2.4 and above.

The `toggle_size_request`() method emits the "toggle−size−request" signal on the menuitem and returns the size requested for the menuitem.

gtk.MenuItem.toggle_size_allocate

```
    def toggle_size_allocate(allocation)
```

`allocation` :	the allocation size for the menu item

The `toggle_size_allocate`() method emits the "toggle−size−allocate" signal on the menu item.

gtk.MenuItem.set_right_justified

```
    def set_right_justified(right_justified)
```

`right_justified` :	if `TRUE` the menu item will appear at the far right if added to a menu bar.

The `set_right_justified`() method sets the justification of the menu item according to the value of *`right_justified`*. If *`right_justified`* is `TRUE` the menu item will appear at the right side of a menu bar. If the widget layout is reversed for a right−to−left language like Hebrew or Arabic, right−justified−menu−items appear on the left.

gtk.MenuItem.get_right_justified

```
    def get_right_justified()
```

Returns :	`TRUE` if the menu item will appear at the far right if added to a menu bar.

The `get_right_justified`() method gets the justification of the menu item. If `TRUE` the menu item appears justified at the right side of the menu bar.

gtk.MenuItem.set_accel_path

```
    def set_accel_path(accel_path)
```

`accel_path` :	the accelerator path, corresponding to this menu item

The `set_accel_path`() method sets the accelerator path on the menu item. The accelerator path provides access to the menu item's accelerator allowing user changes to be identified and saved to persistent storage. See also the `gtk.Menu.set_accel_path()` method for a more convenient variant of this function. This method is a convenience wrapper that handles calling `gtk.Widget.set_accel_path()` with the appropriate accelerator group for the menu item.

Signals

The "activate" gtk.MenuItem Signal

```
    def callback(menuitem, user_param1, ...)
```

menuitem :	the menuitem that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "activate" signal is emitted when the menu item is activated.

The "activate–item" gtk.MenuItem Signal

```
    def callback(menuitem, user_param1, ...)
```

menuitem :	the menuitem that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "activate–item" signal is emitted when the menu item is activated, but also if the menu item has a submenu. For normal applications, the relevant signal is "activate".

The "toggle–size–allocate" gtk.MenuItem Signal

```
    def callback(menuitem, allocation, user_param1, ...)
```

menuitem :	the menuitem that received the signal
allocation :	the size allocation for the menuitem
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "toggle–size–allocate" signal is emitted when the size allocation is changed.

The "toggle–size–request" gtk.MenuItem Signal

```
    def callback(menuitem, requisition, user_param1, ...)
```

menuitem :	the menuitem that received the signal
requisition :	the pointer to the location to put the size request
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "toggle–size–request" signal is emitted when a new size request is needed.

Prev	Up	Next
gtk.MenuBar	Home	gtk.MenuShell

gtk.MenuShell

Prev	**The gtk Class Reference**	Next

gtk.MenuShell

gtk.MenuShell — a base class for menu objects.

Synopsis

```
class gtk.MenuShell(gtk.Container):
    def append(child)
    def prepend(child)
    def insert(child, position)
    def deactivate()
    def select_item(menu_item)
    def deselect()
    def activate_item(menu_item, force_deactivate)
    def select_first(search_sensitive)
    def cancel()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.MenuShell
```

Signal Prototypes

"activate−current"	def callback(*menushell*, *force_hide*, *user_param1*, ...)
"cancel"	def callback(*menushell*, *user_param1*, ...)
"cycle−focus"	def callback(*menushell*, *direction*, *user_param1*, ...)
"deactivate"	def callback(*menushell*, *user_param1*, ...)
"move−current"	def callback(*menushell*, *direction*, *user_param1*, ...)
"selection−done"	def callback(*menushell*, *user_param1*, ...)

Description

A `gtk.MenuShell` is the abstract base class used to derive the `gtk.Menu` and `gtk.MenuBar` subclasses. A `gtk.MenuShell` is a container of `gtk.MenuItem` objects arranged in a list which can be navigated, selected, and activated by the user to perform application functions. A `gtk.MenuItem` can have a submenu associated with it, allowing for nested hierarchical menus.

Methods

gtk.MenuShell.append

```
    def append(child)
```

child :	The `gtk.MenuItem` to add.

The `append`() method adds a new `gtk.MenuItem` specified by *child* to the end of the menu shell's item list.

gtk.MenuShell.prepend

```
    def prepend(child)
```

child :	The gtk.MenuItem to add.

The `prepend()` method adds a new gtk.MenuItem specified by *child* to the beginning of the menu shell's item list.

gtk.MenuShell.insert

```
    def insert(child, position)
```

child :	The gtk.MenuItem to add.
position :	The position in the item list where child should be added. Positions are numbered starting from 0.

The `insert()` method adds a new gtk.MenuItem specified by *child* to the menu shell's item list at the position specified by *position*.

gtk.MenuShell.deactivate

```
    def deactivate()
```

The `deactivate()` method deactivates the menu shell. Typically this results in the menu shell being removed from the screen.

gtk.MenuShell.select_item

```
    def select_item(menu_item)
```

menu_item :	The gtk.MenuItem to select.

The `select_item()` method selects the menu item specified by *menu_item* from the menu shell.

gtk.MenuShell.deselect

```
    def deselect()
```

The `deselect()` method deselects the currently selected item from the menu shell, if any.

gtk.MenuShell.activate_item

```
    def activate_item(menu_item, force_deactivate)
```

menu_item :	The gtk.MenuItem to activate.
force_deactivate :	If `TRUE`, force the deactivation of the menu shell after the menu item is activated.

The `activate_item()` method activates the menu item specified by *menu_item*. If *force_deactivate* is `TRUE` the menushell is forcibly deactivated after *menu_item* is activated.

gtk.MenuShell.select_first

```
    def select_first(search_sensitive)
```

search_sensitive :	

	if `TRUE`, search for the first selectable menu item, otherwise select nothing if the first item isn't sensitive.

Note

This method is available in PyGTK 2.2 and above.

The `select_first`() method selects the first visible or selectable child of the menu shell if *`search_sensitive`* is `TRUE`. Don't select tearoff items unless the only item is a tearoff item. If *`search_sensitive`* is `FALSE` select nothing if the first item isn't sensitive. *`search_sensitive`* should be `FALSE` if the menu is being popped up initially.

gtk.MenuShell.cancel

```
    def cancel()
```

Note

This method is available in PyGTK 2.4 and above.

The `cancel`() method cancels the selection within the menu shell.

Signals

The "activate−current" gtk.MenuShell Signal

```
    def callback(menushell, force_hide, user_param1, ...)
```

`menushell` :	the menushell that received the signal
`force_hide` :	if TRUE, hide the menu after activating the menu item.
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "activate−current" signal is emitted to activate the current menu item in the menushell.

The "cancel" gtk.MenuShell Signal

```
    def callback(menushell, user_param1, ...)
```

`menushell` :	the menushell that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "cancel" signal is emitted to cancel the selection in the menushell. Also causes the "selection−done" signal to be emitted.

The "cycle−focus" gtk.MenuShell Signal

```
    def callback(menushell, user_param1, ...)
```

`menushell` :	the menushell that received the signal
`direction` :	the direction to cycle the focus; one of: `gtk.DIR_TAB_FORWARD`,

	`gtk.DIR_TAB_BACKWARD`, `gtk.DIR_UP`, `gtk.DIR_DOWN`, `gtk.DIR_LEFT` or `gtk.DIR_RIGHT`
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "cycle−focus" signal is emitted when an action occurs requesting the focus move to the next menubar.

The "deactivate" gtk.MenuShell Signal

```
    def callback(menushell, user_param1, ...)
```

`menushell` :	the menushell that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "deactivate" signal is emitted when the menushell is deactivated.

The "move−current" gtk.MenuShell Signal

```
    def callback(menushell, direction, user_param1, ...)
```

`menushell` :	the menushell that received the signal
`direction` :	the direction to move; one of: `gtk.MENU_DIR_PARENT`, `gtk.MENU_DIR_CHILD`, `gtk.MENU_DIR_NEXT` or `gtk.MENU_DIR_PREV`
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "move−current" signal is emitted when the current menu item is to be moved in the direction specified by *`direction`* which is one of:

`gtk.MENU_DIR_PARENT`	To the parent menu shell.
`gtk.MENU_DIR_CHILD`	To the submenu, if any, associated with the item.
`gtk.MENU_DIR_NEXT`	To the next menu item.
`gtk.MENU_DIR_PREV`	To the previous menu item.

The "selection−done" gtk.MenuShell Signal

```
    def callback(menushell, user_param1, ...)
```

`menushell` :	the menushell that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "selection−done" signal is emitted when a selection has been completed within a menu shell.

Prev | Up | Next
gtk.MenuItem | Home | gtk.MenuToolButton

gtk.MenuToolButton
Prev | **The gtk Class Reference** | Next

gtk.MenuToolButton

gtk.MenuToolButton A gtk.ToolItem containing a button with an additional dropdown menu (new in PyGTK 2.6)

Synopsis

```
class gtk.MenuToolButton(gtk.ToolButton):
    gtk.MenuToolButton(stock_id)
    gtk.MenuToolButton(icon_widget, label)
    def set_menu(menu)
    def get_menu()
    def set_arrow_tooltip(tooltips, tip_text, tip_private=None)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.ToolItem
            +-- gtk.ToolButton
              +-- gtk.MenuToolButton
```

Properties

"menu"	Read−Write	The dropdown gtk.Menu.

Signal Prototypes

"show−menu"	`def callback(menutoolbutton, user_param1, ...)`

Description

Note

This widget is available in PyGTK 2.6 and above.

A gtk.MenuToolButton is a gtk.ToolItem that contains a button and a small additional button with an arrow. When clicked, the arrow button pops up a dropdown menu.

Constructor

gtk.MenuToolButton

```
gtk.MenuToolButton(stock_id)
```

stock_id :	
Returns :	a new gtk.MenuToolButton

Note

This constructor is available in PyGTK 2.6 and above.

Creates a new gtk.MenuToolButton using from the stock item specified by *stock_id*. The new gtk.MenuToolButton will contain the icon and label associated with *stock_id*.

gtk.MenuToolButton

```
gtk.MenuToolButton(icon_widget, label)
```

icon_widget :	a widget that will be used as icon widget, or None
label :	a string that will be used as label, or None
Returns :	a new gtk.MenuToolButton

Note

This constructor is available in PyGTK 2.6 and above.

Creates a new gtk.MenuToolButton using the icon specified by *icon_widget* and the label specified by *label*.

Methods

gtk.MenuToolButton.set_menu

```
def set_menu(menu)
```

menu :	a gtk.Menu

Note

This method is available in PyGTK 2.6 and above.

The set_menu() method sets the "menu" property to the gtk.Menu specified by *menu*.

gtk.MenuToolButton.get_menu

```
def get_menu()
```

Returns :	the associated gtk.Menu

Note

This method is available in PyGTK 2.6 and above.

The get_menu() method returns the value of the "menu" property that contains the associated gtk.Menu.

gtk.MenuToolButton.set_arrow_tooltip

```
def set_arrow_tooltip(tooltips, tip_text, tip_private=None)
```

tooltips :	A gtk.Tooltips object.

tip_text :	The text to use as the tooltip or `None`
tip_private :	Opitonal private tooltip text or `None`. Defaults to `None`.

Note

This method is available in PyGTK 2.6 and above.

The `set_arrow_tooltip`() method sets the tooltip data specified by *`tip_text`* for the arrow button using the `gtk.Tooltips` object specified by *`tooltips`*.

Signals

The "show−menu" gtk.MenuToolButton Signal

```
  def callback(menutoolbutton, user_param1, ...)
```

menutoolbutton :	the menutoolbutton that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.6 and above.

The "show−menu" signal is emitted before the dropdown menu is displayed.

gtk.MessageDialog

gtk.MessageDialog — a convenient message window

Synopsis

```
class gtk.MessageDialog(gtk.Dialog):
    gtk.MessageDialog(parent=None, flags=0, type=gtk.MESSAGE_INFO, buttons=gtk.BUTTONS_NONE, me
    def set_markup(str)
    def format_secondary_text(message_format)
    def format_secondary_markup(message_format)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Window
            +-- gtk.Dialog
            +-- gtk.MessageDialog
```

Properties

"buttons"	Write–Construct	The buttons shown in the message dialog. One of the GTK Buttons Type Constants: `gtk.BUTTONS_NONE`, `gtk.BUTTONS_OK`, `gtk.BUTTONS_CLOSE`, `gtk.BUTTONS_CANCEL`, `gtk.BUTTONS_YES_NO`, `gtk.BUTTONS_OK_CANCEL`. Default value: `gtk.BUTTONS_NONE`
"message–type"	Read–Write–Construct	The type of message. One of the GTK Message Type Constants: `gtk.MESSAGE_INFO`, `gtk.MESSAGE_WARNING`, `gtk.MESSAGE_QUESTION` or `gtk.MESSAGE_ERROR`. Default value: `gtk.MESSAGE_INFO`

Style Properties

"message–border"	Read–Write	The width of border around the label and image in the message dialog. Allowed values: >= 0. Default value: 12.

Attributes

"image"	Read	The stock ID image
"label"	Read	The label widget that contains the message text.

Description

The `gtk.MessageDialog` presents a dialog with an image representing the type of message (Error, Question, etc.) alongside some message text. It's simply a convenience widget; you could construct the equivalent of `gtk.MessageDialog` from `gtk.Dialog` without too much effort, but `gtk.MessageDialog` saves time.

The `gtk.MessageDialog` types are listed in the GTK Message Type Constants.

A selection of predefined button sets is available for use in a message dialog. See the GTK Buttons Type Constants.

See the `gtk.Dialog` reference page for additional methods to be used with the `gtk.MessageDialog`.

Constructor

```
    gtk.MessageDialog(parent=None, flags=0, type=gtk.MESSAGE_INFO, buttons=gtk.BUTTONS_NONE, mes
```

parent :	the transient parent, or None if none
flags :	the dialog flags – a combination of: `gtk.DIALOG_MODAL`, `gtk.DIALOG_DESTROY_WITH_PARENT` or 0 for no flags
type :	the type of message: `gtk.MESSAGE_INFO`, `gtk.MESSAGE_WARNING`, `gtk.MESSAGE_QUESTION` or `gtk.MESSAGE_ERROR`.
buttons :	

	the predefined set of buttons to use: `gtk.BUTTONS_NONE`, `gtk.BUTTONS_OK`, `gtk.BUTTONS_CLOSE`, `gtk.BUTTONS_CANCEL`, `gtk.BUTTONS_YES_NO`, `gtk.BUTTONS_OK_CANCEL`
`message_format` :	a string containing the message text or None
Returns :	a new `gtk.MessageDialog` widget

Creates a new `gtk.MessageDialog`, which is a simple dialog with an icon indicating the dialog type (error, warning, etc.) specified by *`type`* and some text (*`message_format`*) the user may want to see. *`parent`* if specified indicates the transient parent of the dialog. The *`flags`* allow the specification special dialog characteristics: make the dialog modal (`gtk.DIALOG_MODAL`) and destroy the dialog when the parent is destroyed (`gtk.DIALOG_DESTROY_WITH_PARENT`). When the user clicks a button a "response" signal is emitted with response IDs. *`buttons`* specifies the set of predefined buttons to use: `gtk.BUTTONS_NONE`, `gtk.BUTTONS_OK`, `gtk.BUTTONS_CLOSE`, `gtk.BUTTONS_CANCEL`, `gtk.BUTTONS_YES_NO`, `gtk.BUTTONS_OK_CANCEL`. See `gtk.Dialog` for more details.

Methods

gtk.MessageDialog.set_markup

```
    def set_markup(str)
```

`str` :	a markup string (see the Pango markup language reference)

Note

This method is available in PyGTK 2.4 and above.

The `set_markup()` method sets the text of the message dialog to the contents of *`str`*. If *`str`* contains text marked up with Pango markup (see The Pango Markup Language), it will be displayed with those attributes. Note the '<', '>' and '&' characters must be replaced with '&lt;', '&gt;' and '&amp;' respectively to be displayed literally.

gtk.MessageDialog.format_secondary_text

```
    def format_secondary_text(message_format)
```

`message_format` :	The text to be displayed as the secondary text or `None`.

Note

This method is available in PyGTK 2.6 and above.

The `format_secondary_text()` method sets the secondary text of the message dialog to the text specified by *`message_format`*. Note that setting a secondary text makes the primary text bold, unless you have provided explicit markup.

gtk.MessageDialog.format_secondary_markup

```
    def format_secondary_markup(message_format)
```

`message_format` :	A string containing the pango markup to use as secondary text.

Note

This method is available in PyGTK 2.6 and above.

The `format_secondary_markup()` method sets the secondary text to the markup text specified by *`message_format`*. Note that setting a secondary text makes the primary text become bold, unless you have provided explicit markup.

gtk.Misc

gtk.Misc   a base class for widgets with alignments and padding.

Synopsis

```
class gtk.Misc(gtk.Widget):
    def set_alignment(xalign, yalign)
    def get_alignment()
    def set_padding(xpad, ypad)
    def get_padding()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Misc
```

Properties

"xalign"	Read–Write	The horizontal alignment, from 0.0 to 1.0
"yalign"	Read–Write	The vertical alignment, from 0.0 to 1.0
"xpad"	Read–Write	The amount of space to add on the left and right of the widget, in pixels
"ypad"	Read–Write	The amount of space to add above and below the widget, in pixels

Description

The `gtk.Misc` widget is an abstract widget used to derive subclasses which have alignment and padding attributes. The horizontal and vertical padding attributes allow extra space to be added around the widget. The horizontal and vertical alignment attributes enable the widget to be positioned within its allocated area. The alignment values represent the fraction of available free space (allocation minus the widget size) to place to the left or above the widget for x and y alignment respectively. Note that if the widget is added to a container in such a way that it expands automatically to fill its allocated area, the alignment settings will have no effect.

Methods

gtk.Misc.set_alignment

```
    def set_alignment(xalign, yalign)
```

xalign :	the horizontal alignment
yalign :	the vertical alignment

The `set_alignment()` method sets the alignment of the widget in its allocated space. *`xalign`* sets the fraction (0.0−1.0) of free space (horizontal allocation minus widget width) to place to the left of the widget. *`yalign`* sets the fraction (0.0−1.0) of free space (vertical allocation minus widget height) to place above the widget.

gtk.Misc.get_alignment

```
    def get_alignment()
```

Returns :	a tuple containing the X and Y alignments of the widget

The `get_alignment()` method returns a tuple containing the X and Y alignments of the widget within its allocation. See `set_alignment()`.

gtk.Misc.set_padding

```
    def set_padding(xpad, ypad)
```

xpad :	the amount of space to add on the left and right of the widget, in pixels.
ypad :	the amount of space to add on the top and bottom of the widget, in pixels.

The `set_padding()` method sets the amount of space to add around the widget. The *`xpad`* value specifies the number of pixels of padding to add to the left and right of the widget. The *`yalign`* value specifies the number of pixels to add above and below the widget.

gtk.Misc.get_padding

```
    def get_padding()
```

Returns :	a tuple containing the horizontal and vertical padding in pixels for the widget.

The `get_padding()` method returns a tuple containing the padding in the horizontal and vertical directions of the widget. See `set_padding()`.

Prev Up Next
gtk.MessageDialog Home gtk.Notebook

gtk.Notebook

gtk.Notebook — a tabbed notebook container.

Synopsis

```
class gtk.Notebook(gtk.Container):
    gtk.Notebook()
    def append_page(child, tab_label=None)
    def append_page_menu(child, tab_label=None, menu_label=None)
    def prepend_page(child, tab_label=None)
    def prepend_page_menu(child, tab_label=None, menu_label=None)
    def insert_page(child, tab_label=None, position=-1)
    def insert_page_menu(child, tab_label=None, menu_label=None, position=-1)
    def remove_page(page_num)
    def get_current_page()
    def get_nth_page(page_num)
    def get_n_pages()
    def page_num(child)
    def set_current_page(page_num)
    def next_page()
    def prev_page()
    def set_show_border(show_border)
    def get_show_border()
    def set_show_tabs(show_tabs)
    def get_show_tabs()
    def set_tab_pos(pos)
    def get_tab_pos()
    def set_scrollable(scrollable)
    def get_scrollable()
    def popup_enable()
    def popup_disable()
    def get_tab_label(child)
    def set_tab_label(child, tab_label=None)
    def set_tab_label_text(child, tab_text)
    def get_tab_label_text(child)
    def get_menu_label(child)
    def set_menu_label(child, menu_label=None)
    def set_menu_label_text(child, menu_text)
    def get_menu_label_text(child)
    def query_tab_label_packing(child)
    def set_tab_label_packing(child, expand, fill, pack_type)
    def reorder_child(child, position)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Notebook
```

Properties

"enable–popup"	Read–Write	If `TRUE`, pressing the right mouse button on the notebook pops up a menu that you can use to go to a page
"homogeneous"	Read–Write	If `TRUE`, tabs should have homogeneous sizes
"page"	Read–Write	The index of the current page
"scrollable"	Read–Write	If `TRUE`, scroll arrows are added if there are too many tabs to fit
"show–border"	Read–Write	If `TRUE`, the border should be shown
"show–tabs"	Read–Write	If `TRUE`, tabs should be shown
"tab–border"	Write	The width of the border around the tab labels

"tab–hborder"	Read–Write	The width of the horizontal border of tab labels
"tab–pos"	Read–Write	The side of the notebook that holds the tabs: `gtk.POS_LEFT`, `gtk.POS_RIGHT`, `gtk.POS_TOP` or `gtk.POS_BOTTOM`
"tab–vborder"	Read–Write	The width of the vertical border of tab labels

Child Properties

"menu–label"	Read–Write	The string displayed in the child's menu entry
"position"	Read–Write	The index of the child in the parent
"tab–expand"	Read–Write	If `TRUE`, expand the child's tabs
"tab–fill"	Read–Write	If `TRUE`, the child's tab should fill the allocated area
"tab–label"	Read–Write	The string displayed on the child's tab label
tab–pack""	Read–Write	A pack type indicating whether the child is packed with reference to the start or end of the parent: `gtk.PACK_START` or `gtk.PACK_END`.

Style Properties

Note

These style properties are available in PyGTK 2.4 and above.

"has–backward–stepper"	Read–Write	If `TRUE` the standard backward arrow button is displayed.
"has–forward–stepper"	Read–Write	If `TRUE` the standard forward arrow button is displayed.
"has–secondary–backward–stepper"	Read–Write	If `TRUE` a second backward arrow button is displayed on the opposite end of the tab area.
"has–secondary–forward–stepper"	Read–Write	If `TRUE` a second forward arrow button is displayed on the opposite end of the tab area.

Attributes

"tab_pos"	Read	The side of the notebook that holds the tabs: `gtk.POS_LEFT`, `gtk.POS_RIGHT`, `gtk.POS_TOP` or `gtk.POS_BOTTOM`

Signal Prototypes

```
"change–current–page" def callback(notebook, offset, user_param1, ...)
"focus–tab"           def callback(notebook, type, user_param1, ...)
"move–focus–out"      def callback(notebook, direction_type, user_param1, ...)
"select–page"         def callback(notebook, move_focus, user_param1, ...)
"switch–page"         def callback(notebook, page, page_num, user_param1, ...)
```

Description

The `gtk.Notebook` widget is a `gtk.Container` whose children are overlapping pages that can be switched between by using tab labels along one edge. There are many configuration options for the `gtk.Notebook`. You can choose on which edge the tabs appear (see `set_tab_pos()`); whether the notebook should be made bigger or scrolling arrows added if there are too many tabs to fit (see `set_scrollable()`); and, whether there will be a popup menu allowing the users to switch pages. (see `popup_enable()`, `popup_disable()`).

The `gtk.Notebook` is particularly useful for displaying a large number of application controls that can be grouped into several functional areas. The typical example is the user preferences dialog in some applications. For example, a notebook page can be setup for each of font selection, color selection, formating, etc.

Constructor

```
gtk.Notebook()
```

Returns :	the newly created `gtk.Notebook`

Creates a new `gtk.Notebook` widget with no pages.

Methods

gtk.Notebook.append_page

```
def append_page(child, tab_label=None)
```

child :	the `gtk.Widget` to use as the contents of the page.
tab_label :	the `gtk.Widget` to be used as the label for the page.
Returns :	in PyGTK 2.0 and 2.2 returns `None`. In PyGTK 2.4 and above returns the index number of the page in the notebook

The `append_page()` method appends a page to the notebook using the widget specified by *`child`* and the widget specified by *`tab_label`* as the label on the tab.

In PyGTK 2.4 and above *`tab_label`* can be `None` to use a default label. Also if *`tab_label`* is not specified it will default to `None`.

gtk.Notebook.append_page_menu

```
def append_page_menu(child, tab_label=None, menu_label=None)
```

child :	the `gtk.Widget` to use as the contents of the page.
tab_label :	the `gtk.Widget` to be used as the label for the page.
menu_label :	the widget to use as a label for the page–switch menu, if that is enabled.
Returns :	in PyGTK 2.0 and 2.2 returns `None`. In PyGTK 2.4 and above returns the index number of the page in the notebook

The `append_page_menu()` method appends a page to the notebook and specifying the widget to use as the label in the popup menu. *`child`* specifies the widget to use as the contents of the page; *`tab_label`* specifies the widget to be used as the tab label; and, *`menu_label`* specifies the widget to use in the popup menu.

In PyGTK 2.4 and above *tab_label* can be None to use a default label. If *tab_label* is a gtk.Label or None and *menu_label* is None then the menu label will have the same text as the tab label. Otherwise, *menu_label* must be specified and not None. These parameters will default to None in PyGTK 2.4 as well.

gtk.Notebook.prepend_page

```
def prepend_page(child, tab_label=None)
```

child :	the gtk.Widget to use as the contents of the page.
tab_label :	the gtk.Widget to be used as the label for the page.
Returns :	in PyGTK 2.0 and 2.2 returns None. In PyGTK 2.4 and above returns the index number of the page in the notebook

The prepend_page() method prepends a page to the notebook. *child* specifies the widget to use as the contents of the page and *tab_label* specifies the widget to be used as the tab label.

In PyGTK 2.4 and above *tab_label* can be None to use a default label. Also if *tab_label* is not specified it will default to None.

gtk.Notebook.prepend_page_menu

```
def prepend_page_menu(child, tab_label=None, menu_label=None)
```

child :	the gtk.Widget to use as the contents of the page.
tab_label :	the gtk.Widget to be used as the label for the page.
menu_label :	the widget to use as a label for the page–switch menu, if that is enabled.
Returns :	in PyGTK 2.0 and 2.2 returns None. In PyGTK 2.4 and above returns the index number of the page in the notebook

The prepend_page_menu() method prepends a page to the notebook specifying the widget to use as the label in the popup menu. *child* specifies the widget to use as the contents of the page; *tab_label* specifies the widget to use as the tab label; and, *menu_label* specifies the widget to use in the popup menu.

In PyGTK 2.4 and above *tab_label* can be None to use a default label. If *tab_label* is a gtk.Label or None and *menu_label* is None then the menu label will have the same text as the tab label. Otherwise, *menu_label* must be specified and not None. These parameters will default to None in PyGTK 2.4 as well.

gtk.Notebook.insert_page

```
def insert_page(child, tab_label=None, position=None)
```

child :	the gtk.Widget to use as the contents of the page.
tab_label :	the gtk.Widget to be used as the label for the page.
position :	the index (starting at 0) at which to insert the page, or –1 to append the page after all other pages.
Returns :	in PyGTK 2.0 and 2.2 returns None. In PyGTK 2.4 and above returns the index number of the page in the notebook

The insert_page() method inserts a page into the notebook at the location specified by *position* (0 is the first page). *child* is the widget to use as the contents of the page and *tab_label* specifies the widget to be used as the tab label. If *position* is –1 the page is appended to the notebook. In PyGTK 2.4 and above if *tab_label* is None a default label if "page N" is used.

In PyGTK 2.4 and above *tab_label* can be `None` to use a default label. Also if *tab_label* is not specified it will default to `None`.

In Pygtk 2.4 and above *position* will default to −1 if not specified.

gtk.Notebook.insert_page_menu

```
    def insert_page_menu(child, tab_label=None, menu_label=None, position=None)
```

child :	the gtk.Widget to use as the contents of the page.
tab_label :	the gtk.Widget to be used as the label for the page.
menu_label :	the widget to use as a label for the page−switch menu, if that is enabled.
position :	the index (starting at 0) at which to insert the page, or −1 to append the page after all other pages.
Returns :	in PyGTK 2.0 and 2.2 returns `None`. In PyGTK 2.4 and above returns the index number of the page in the notebook

The `insert_page_menu`() method inserts a page into the notebook at the location specified by *position*. *child* specifies the widget to use as the contents of the page; *tab_label* specifies the widget to use as the tab label; and *menu_label* specifies the widget to use as the label in the popup menu.

In PyGTK 2.4 and above *tab_label* can be `None` to use a default label. If *tab_label* is a gtk.Label or `None` and *menu_label* is `None` then the menu label will have the same text as the tab label. Otherwise, *menu_label* must be specified and not `None`. These parameters will default to `None` in PyGTK 2.4 as well.

In Pygtk 2.4 and above *position* will default to −1 if not specified.

gtk.Notebook.remove_page

```
    def remove_page(page_num)
```

page_num :	the index of a notebook page, starting from 0. If −1, the last page will be removed.

The `remove_page`() method removes from the notebook the page at the location specified by *index*. The value of *index* starts from 0. If *index* is −1 the last page of the notebook will be removed.

gtk.Notebook.get_current_page

```
    def get_current_page()
```

Returns :	the index (starting from 0) of the current page in the notebook. If the notebook has no pages, then −1 will be returned.

The `get_current_page`() method returns the page index of the current page numbered from 0.

gtk.Notebook.get_nth_page

```
    def get_nth_page(page_num)
```

page_num :	the index of a page in the notebook
Returns :	the child widget, or `None` if *page_num* is out of bounds.

The `get_nth_page`() method returns the child widget contained in the page with the index specified by *page_num*. If *page_num* is out of bounds for the page range of the notebook this method returns `None`.

gtk.Notebook.get_n_pages

```
def get_n_pages()
```

Returns :	the number of pages in the notebook.

Note

This method is available in PyGTK 2.4 and above.

The `get_n_pages()` method returns the number of pages in a notebook.

gtk.Notebook.page_num

```
def page_num(child)
```

`child` :	a `gtk.Widget`
Returns :	the index of the page containing *`child`*, or –1 if *`child`* is not in the notebook.

The `page_num()` method returns the index of the page which contains the widget specified by *`child`* or `None` if no page contains *`child`*.

gtk.Notebook.set_current_page

```
def set_current_page(page_num)
```

`page_num` :	the index of the page to switch to, starting from 0. If negative, the last page will be used. If greater than the number of pages in the notebook, nothing will be done.

The `set_current_page()` method switches to the page number specified by *`page_num`*. If *`page_num`* is negative the last page is selected.

gtk.Notebook.next_page

```
def next_page()
```

The `next_page()` method switches to the next page. Nothing happens if the current page is the last page.

gtk.Notebook.prev_page

```
def prev_page()
```

The `prev_page()` method switches to the previous page. Nothing happens if the current page is the first page.

gtk.Notebook.set_show_border

```
def set_show_border(show_border)
```

`show_border` :	if `TRUE` a bevel should be drawn around the notebook.

The `show_border()` method sets the "show–border" property to the value of *`show_border`*. If *`show_border`* is `TRUE` a bevel will be drawn around the notebook pages. This only has a visual effect when the tabs are not shown. See `set_show_tabs()`.

gtk.Notebook.get_show_border

```
def get_show_border()
```

Returns :	`TRUE` if the bevel should be drawn

The `get_show_border()` method returns the value of the "show−border" property. If "show−border" is `TRUE` a bevel will be drawn around the notebook pages when tabs are not shown. See `set_show_border()`.

gtk.Notebook.set_show_tabs

```
def set_show_tabs(show_tabs)
```

`show_tabs` :	if `TRUE` the tabs should be shown.

The `set_show_tabs()` method sets the "show−tabs" property to the value of *`show_tabs`*. If *`show_tabs`* is `TRUE` the notebook tabs will be displayed.

gtk.Notebook.get_show_tabs

```
def get_show_tabs()
```

Returns :	`TRUE` if the tabs are shown

The `get_show_tabs()` method returns the value of the "show−tabs" property. If "show−tabs" is `TRUE` the tabs of the notebook are shown. See `set_show_tabs()`.

gtk.Notebook.set_tab_pos

```
def set_tab_pos(pos)
```

`pos` :	the edge to draw the tabs at: `gtk.POS_LEFT`, `gtk.POS_RIGHT`, `gtk.POS_TOP` or `gtk.POS_BOTTOM`.

The `set_tab_pos()` method sets the edge at which the tabs for switching pages in the notebook are drawn as specified by *`pos`*. The value of *`pos`* can be one of: `gtk.POS_LEFT`, `gtk.POS_RIGHT`, `gtk.POS_TOP` or `gtk.POS_BOTTOM`.

gtk.Notebook.get_tab_pos

```
def get_tab_pos()
```

Returns :	the edge at which the tabs are drawn

The `get_tab_pos()` method returns the edge at which the tabs for switching pages in the notebook are drawn. The return value is one of: `gtk.POS_LEFT`, `gtk.POS_RIGHT`, `gtk.POS_TOP` or `gtk.POS_BOTTOM`.

gtk.Notebook.set_scrollable

```
def set_scrollable(scrollable)
```

`scrollable` :	if `TRUE` scroll arrows should be added

The `set_scrollable()` method sets the "scrollable" property to the value specified by *`scrollable`*. If *`scrollable`* is `TRUE` the tab label area will have arrows for scrolling if there are too many tabs to fit in the area.

gtk.Notebook.get_scrollable

```
def get_scrollable()
```

Returns :	`TRUE` if arrows for scrolling are enabled

The `get_scrollable()` method returns the value of the "scrollable" property. If "scrollable" is `TRUE` the tab label area has scrolling arrows enabled and displayed if there are too many tabs to fit in the display area. See `set_scrollable()`.

gtk.Notebook.popup_enable

```
def popup_enable()
```

The `popup_enable()` method enables the popup menu: if the user clicks with the right mouse button on the tabs, a menu with all the pages will be popped up.

gtk.Notebook.popup_disable

```
def popup_disable()
```

The `popup_disable()` method disables the popup menu.

gtk.Notebook.get_tab_label

```
def get_tab_label(child)
```

`child` :	a widget contained in a notebook page
Returns :	the tab label widget for the page containing *`child`*

The `get_tab_label()` method returns the tab label widget for the page containing the widget *`child`*. `None` is returned if *`child`* is not in the notebook.

gtk.Notebook.set_tab_label

```
def set_tab_label(child, tab_label=None)
```

`child` :	a widget contained in a notebook page
`tab_label` :	the tab label widget to use or `None`.

The `set_tab_label()` method replaces the tab label for the notebook page containing *`child`* with the widget specified by *`tab_label`*.

In PyGTK 2.4 and above *`tab_label`* can be `None` to use a default label. Also if *`tab_label`* is not specified it will default to `None`.

gtk.Notebook.set_tab_label_text

```
def set_tab_label_text(child, tab_text)
```

`child` :	a widget contained in a notebook page
`tab_text` :	the new label text

The `set_tab_label_text()` method creates a new label with the text specified by *`tab_text`* and sets it as the tab label for the page containing *`child`*.

gtk.Notebook.get_tab_label_text

```
def get_tab_label_text(child)
```

child :	a widget contained in a notebook page
Returns :	value: the text of the tab label, or `None` if the tab label widget is not a `gtk.Label` or *`child`* is not in a notebook page.

The `get_tab_label_text()` retrieves the text of the tab label for the page containing *`child`*. This method returns `None` if *`child`* is not in a notebook page or the page tab label is not a `gtk.Label` widget.

gtk.Notebook.get_menu_label

```
def get_menu_label(child)
```

child :	a widget contained in a notebook page
Returns :	the menu label, or `None` if the notebook page does not have a menu label other than the default (the tab label) or if child is not in a notebook page.

The `get_menu_label()` method retrieves the menu label widget of the page containing *`child`*. If *`child`* is not in a notebook page or the menu label has not been set (it defaults to the tab label), this method returns `None`.

gtk.Notebook.set_menu_label

```
def set_menu_label(child, menu_label=None)
```

child :	a widget contained in a notebook page
menu_label :	a widget to be used as the new menu label

The `set_menu_label()` replaces the menu label for the page containing *`child`* with the widget specified by *`menu_label`*.

In PyGTK 2.4 and above *`menu_label`* can be `None` to use a default label. Also if *`menu_label`* is not specified it will default to `None`. See the `append_page_menu()` method for more information.

gtk.Notebook.set_menu_label_text

```
def set_menu_label_text(child, menu_text)
```

child :	a widget contained in a notebook page
menu_text :	the new menu label text

The `set_menu_label_text()` method creates a new label widget and replaces the menu label of the page containing*`child`*.

gtk.Notebook.get_menu_label_text

```
def get_menu_label_text(child)
```

child :	a child widget of a notebook page.
Returns :	value: the text of the tab label, or `None` if the widget does not have a menu label other than the default menu label, or the menu label widget is not a `gtk.Label` or *`child`* is not contained in a notebook page.

The `get_menu_label_text()` method retrieves the text of the menu label for the page containing *`child`*. If *`child`* is not in a notebook page or the menu label is not a `gtk.Label` widget or the menu label has not been set (it default to the tab label), the return value is `None`.

gtk.Notebook.query_tab_label_packing

```
def query_tab_label_packing(child)
```

`child` :	the page
Returns :	a tuple containing: the expand value, the fill value and the pack type

The `query_tab_label_packing()` method returns a tuple containing the packing attributes (expand, fill, pack type) for the tab label of the page containing *`child`*. If the expand attribute is `TRUE` the tab can expand to take up the free space in the tab area. If fill is `TRUE` the label widget in the tab can use up all the space in the tab. The pack type can be one of `gtk.PACK_START` or `gtk.PACK_END` to specify whether the tab is packed to the left or right if tabs are on the top or bottom edge (top or bottom if the tabs are on the left or right edge).

gtk.Notebook.set_tab_label_packing

```
def set_tab_label_packing(child, expand, fill, pack_type)
```

`child` :	a widget contained in a notebook page
`expand` :	if `TRUE` the tab can expand to fill the free space in the tab area
`fill` :	if `TRUE` the label widget can fill the space in the tab
`pack_type` :	the position of the tab: `gtk.PACK_START` or `gtk.PACK_END`

The `set_tab_label_packing()` method sets the packing parameters for the tab label of the page containing *`child`*. If *`expand`* is `TRUE` the tab can expand to take up the free space in the tab area. If *`fill`* is `TRUE` the label widget in the tab can use up all the space in the tab. The *`pack_type`* can be one of `gtk.PACK_START` or `gtk.PACK_END` to specify whether the tab is packed to the left or right if tabs are on the top or bottom edge (top or bottom if the tabs are on the left or right edge).See `gtk.Box.pack_start()` for the exact meaning of the parameters.

gtk.Notebook.reorder_child

```
def reorder_child(child, position)
```

`child` :	the child widget to move
`position` :	the index of the page that *`child`* is to move to, or –1 to move to the end

The `reorder_child()` method reorders the notebook pages so that *`child`* appears in the page whose index is specified by *`position`*. If *`position`* is greater than or equal to the number of children in the list or negative, *`child`* will be moved to the end of the list.

Signals

The "change–current–page" gtk.Notebook Signal

```
def callback(notebook, offset, user_param1, ...)
```

`notebook` :	the notebook that received the signal
`offset` :	the count of pages to move (negative count is backward)
`user_param1` :	the first user parameter (if any) specified with the `connect()` method

`...` :	additional user parameters (if any)

The "change−current−page" signal is emitted when the page forward or page backward request is issued.

The "focus−tab" gtk.Notebook Signal

```
    def callback(notebook, type, user_param1, ...)
```

`notebook` :	the notebook that received the signal
`type` :	the type of tab: `gtk.NOTEBOOK_TAB_FIRST` or `gtk.NOTEBOOK_TAB_LAST`
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled

The "focus−tab" signal is emitted when the focus is changed by tabbing.

The "move−focus−out" gtk.Notebook Signal

```
    def callback(notebook, direction_type, user_param1, ...)
```

`notebook` :	the notebook that received the signal
`direction_type` :	the direction type of the focus move: `gtk.DIR_TAB_FORWARD`, `gtk.DIR_TAB_BACKWARD`, `gtk.DIR_UP`, `gtk.DIR_DOWN`, `gtk.DIR_LEFT` or `gtk.DIR_RIGHT`
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "move−focus−out" signal is emitted when the focus is moved out of the notebook with the specified *`direction_type`*.

The "select−page" gtk.Notebook Signal

```
    def callback(notebook, move_focus, user_param1, ...)
```

`notebook` :	the notebook that received the signal
`move_focus` :	if `TRUE` move the focus to a child widget
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled

The "select−page" signal is emitted when a new child page is selected.

The "switch−page" gtk.Notebook Signal

```
    def callback(notebook, page, page_num, user_param1, ...)
```

`notebook` :	the notebook that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "switch−page" signal is emitted when the notebook page is changed.

Prev	Up	Next
gtk.Misc	Home	gtk.Object

gtk.Object

gtk.Object the base class of the PyGTK type hierarchy.

Synopsis

```
class gtk.Object(gobject.GObject):
    def flags()
    def set_flags(flags)
    def unset_flags(flags)
    def destroy()
```

```
Functions

    def gtk.bindings_activate(object, keyval, modifiers)
    def gtk.bindings_activate_event(object, event)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
```

Signal Prototypes

"destroy"	`def callback(object, user_param1, ...)`

Description

`gtk.Object` is the base class for all widgets, and for a few non−widget objects such as `gtk.Adjustment`. `gtk.Object` predates `GObject`; non−widgets that derive from `gtk.Object` rather than `GObject` do so for backward compatibility reasons.

The "destroy" signal, emitted by the `destroy()` method asks all code owning a GTK reference to the object to release its GTK reference. So, for example, if you call `window.destroy()` where *`window`* is a `gtk.Window`, GTK will release the GTK reference count that it owns; if you call `button.destroy()` where button is a `gtk.Button`, *`button`* will be removed from its parent container and the parent container will release its GTK reference to *`button`*. Because these GTK references are released, calling `destroy()` should result in freeing all memory associated with an object (finalizing it) if the GTK reference count reaches zero. However, in PyGTK the GTK objects are wrapped in a Python object that has its own reference counting mechanism. The `destroy()` method does not affect the Python reference counts. The GTK object associated with a Python object will not be released until the Python object reference count reaches zero. Therefore, calling the `destroy()` method will not result in the finalization of the GTK object until the Python object is finalized. In the case mentioned above if a `gtk.Button` is destroyed using the `destroy()` method, it will be removed from its container and unmapped and unrealized but it will not be finalized because the Python wrapper object will still exist and hold a reference.

Methods

gtk.Object.flags

```
def flags()
```

Returns :	the flags set for the object

The `flags`() method returns the value of the flags for the object. The flags returned will include both the `gtk.Object` flags and the `gtk.Widget` flags.

The `gtk.Object` flags are:

`gtk.IN_DESTRUCTION`	the object is currently being destroyed.
`gtk.FLOATING`	the object is orphaned.
`gtk.RESERVED_1`	reserved for future use
`gtk.RESERVED_2`	reserved for future use

The `gtk.Widget` flags are:

`gtk.TOPLEVEL`	widgets without a real parent (e.g. `gtk.Window` and `gtk.Menu`) have this flag set throughout their lifetime. Toplevel widgets always contain their own `gtk.gdk.Window`.
`gtk.NO_WINDOW`	a widget that does not provide its own `gtk.gdk.Window`. Visible action (e.g. drawing) is performed on the parent's `gtk.gdk.Window`.
`gtk.REALIZED`	the widget has an associated `gtk.gdk.Window`.
`gtk.MAPPED`	the widget can be displayed on the screen.
`gtk.VISIBLE`	the widget will be mapped as soon as its parent is mapped.
`gtk.SENSITIVE`	The sensitivity of a widget determines whether it will receive certain events (e.g. button or key presses). One requirement for the widget's sensitivity is to have this flag set.
`gtk.PARENT_SENSITIVE`	This is the second requirement for the widget's sensitivity. Once a widget has `gtk.SENSITIVE` and `gtk.PARENT_SENSITIVE set`, its state is effectively sensitive.
`gtk.CAN_FOCUS`	the widget is able to handle focus grabs.
`gtk.HAS_FOCUS`	the widget has the focus – assumes that `gtk.CAN_FOCUS` is set
`gtk.CAN_DEFAULT`	the widget is allowed to receive the default action.
`gtk.HAS_DEFAULT`	the widget currently will receive the default action.
`gtk.HAS_GRAB`	the widget is in the grab_widgets stack, and will be the preferred one for receiving events.
`gtk.RC_STYLE`	the widgets style has been looked up through the RC mechanism. It does not imply that the widget actually had a style defined through the RC mechanism.
`gtk.COMPOSITE_CHILD`	the widget is a composite child of its parent.
`gtk.NO_REPARENT`	unused
`gtk.APP_PAINTABLE`	set on widgets whose window the application directly draws on, in order to keep GTK from overwriting the drawn stuff.
`gtk.RECEIVES_DEFAULT`	the widget when focused will receive the default action and have `gtk.HAS_DEFAULT` set even if there is a different widget set as default.
`gtk.DOUBLE_BUFFERED`	exposes done on the widget should be double–buffered.

gtk.Object.set_flags

```
    def set_flags(flags)
```

flags :	the gtk.Object and gtk.Widget flags to be set on this object

The `set_flags`() method sets the object flags according to the value of *`flags`*. See flags() for a description of the gtk.Object and gtk.Widget flags that can be set.

gtk.Object.unset_flags

```
    def unset_flags(flags)
```

flags :	the gtk.Object and gtk.Widget flags to be unset on this object

The `unset_flags`() method unsets the object flags according to the value of *`flags`*. See flags() for a description of the gtk.Object and gtk.Widget flags that can be unset.

gtk.Object.destroy

```
    def destroy()
```

The `destroy`() method emits the "destroy" signal notifying all reference holders that they should release the gtk.Object.

Functions

gtk.bindings_activate

```
    def gtk.bindings_activate(object, keyval, modifiers)
```

object :	the gtk.Object to activate the bindings on
keyval :	a key value
modifiers :	a modifier mask
:	
Returns :	`TRUE` if the binding could be activated

The `gtk.bindings_activate`() function activates the bindings associated with the gtk.Object specified by object with the key value specified by *`keyval`* and the modifier mask specified by *`modifiers`*.

gtk.bindings_activate_event

```
    def gtk.bindings_activate_event(object, event)
```

object :	the gtk.Object to activate the bindings on
event :	a gtk.gdk.Event
Returns :	`TRUE` if a matching key binding was found

The `gtk.bindings_activate_event`() function looks up key bindings for the gtk.Object specified by *`object`* to find one matching the key gtk.gdk.Event specified by *`event`*, and if one was found, activate it.

Signals

The "destroy" gtk.Object Signal

```
    def callback(object, user_param1, ...)
```

object :	the object that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "destroy" signal is emitted when the references for the object should be destroyed.

Prev Up Next
gtk.Notebook Home gtk.OptionMenu

gtk.OptionMenu
Prev **The gtk Class Reference** Next

gtk.OptionMenu

gtk.OptionMenu   a widget used to provide a list of valid choices.

Synopsis

```
class gtk.OptionMenu(gtk.Button):
    gtk.OptionMenu()
    def get_menu()
    def set_menu(menu)
    def remove_menu()
    def get_history()
    def set_history(index)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Button
            +-- gtk.OptionMenu
```

Properties

"menu"	Read–Write	The menu of options.

Style Properties

"indicator–spacing"	Read	The spacing around the indicator.
"indicator–size"	Read	The size of the dropdown indicator

Signal Prototypes

"changed"	`def callback(optionmenu, user_param1, ...)`

Description

Warning

`gtk.OptionMenu` is deprecated in PyGTK 2.4; use the `gtk.ComboBox` instead.

A `gtk.OptionMenu` is a widget allows the user to choose from a list of valid choices from an associated menu. The `gtk.OptionMenu` displays the last selected choice. When activated (clicked) the `gtk.OptionMenu` displays a popup `gtk.Menu` which allows the user to make a new choice. Using a `gtk.OptionMenu` is simple:

- build a `gtk.Menu` using `gtk.Menu()`
- then append menu items to it using `gtk.MenuShell.append()`
- associate the menu with the option menu using `set_menu()`
- set the selected menu item with `set_history()`
- connect to the "changed" signal on the option menu; in the "changed" signal
- check the new selected menu item with `get_history()`.

Constructor

```
gtk.OptionMenu()
```

Returns :	a new optionmenu widget

Creates a new `gtk.OptionMenu` widget.

Methods

gtk.OptionMenu.get_menu

```
def get_menu()
```

Returns :	a menu widget or `None` if no menu is associated

The `get_menu()` method returns the menu that is associated with the optionmenu or `None` if no menu is associated.

gtk.OptionMenu.set_menu

```
def set_menu(menu)
```

`menu` :	a menu to be associated with the optionmenu

The `set_menu()` method associates the `gtk.Menu` widget specified by *`menu`* with the optionmenu thus providing the way for a user to select a new choice. A simple menu, avoiding the use of tearoff menu items, submenus, and accelerators, should be used.

gtk.OptionMenu.remove_menu

```
    def remove_menu()
```

The `remove_menu()` method removes the currently associated menu from the optionmenu.

gtk.OptionMenu.get_history

```
    def get_history()
```

Returns :	the index of the selected menu item, or −1 if there are no menu items

The `get_history()` method returns the index of the currently selected menu item or −1 if there are no menu items. The menu items are numbered from top to bottom, starting with 0.

gtk.OptionMenu.set_history

```
    def set_history(index)
```

`index` :	the index of the menu item to display as the selected optionmenu choice

The `set_history()` method selects the menu item specified by *`index`* as the displayed optionmenu choice.

Signals

The "changed" gtk.OptionMenu Signal

```
    def callback(optionmenu, user_param1, ...)
```

`optionmenu` :	the optionmenu that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "changed" signal is emitted when a new optionmenu choice is made.

Prev | Up | Next
gtk.Object | Home | gtk.Paned

gtk.Paned
Prev | **The gtk Class Reference** | Next

gtk.Paned

gtk.Paned a base class for widgets with two adjustable panes

Synopsis

```
class gtk.Paned(gtk.Container):
    def add1(child)
    def add2(child)
    def pack1(child, resize=FALSE, shrink=TRUE)
    def pack2(child, resize=TRUE, shrink=TRUE)
    def get_position()
    def set_position(position)
    def compute_position(allocation, child1_req, child2_req)
    def get_child1()
    def get_child2()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Paned
```

Properties

"max–position"	Read–Write	The largest possible value for the position property. This property is derived from the size and shrinkability of the widget's children. Available in GTK+ 2.4 and above.
"min–position"	Read–Write	The smallest possible value for the position property. This property is derived from the size and shrinkability of the widget's children. Available in GTK+ 2.4 and above.
"position"	Read–Write	The position of the paned separator in pixels (0 means all the way to the left or top).
"position–set"	Read–Write	If `TRUE`, the "position" property is valid.

Child Properties

"resize"	Read–Write	If `TRUE`, the child expands and shrinks along with the paned widget. Available in GTK+ 2.4 and above.
"shrink"	Read–Write	If `TRUE`, the child can be made smaller than its requisition. Available in GTK+ 2.4 and above.

Style Properties

"handle–size"	Read	The width of the handle

Signal Prototypes

"accept–position"	`def callback(`*paned*`, `*user_param1*`, ...)`
"cancel–position"	`def callback(`*paned*`, `*user_param1*`, ...)`
"cycle–child–focus"	`def callback(`*paned*`, `*reversed*`, `*user_param1*`, ...)`
"cycle–handle–focus"	`def callback(`*paned*`, `*reversed*`, `*user_param1*`, ...)`
"move–handle"	`def callback(`*paned*`, `*scrolltype*`, `*user_param1*`, ...)`
"toggle–handle–focus"	`def callback(`*paned*`, `*user_param1*`, ...)`

Description

`gtk.Paned` is the base class for widgets with two panes, arranged either horizontally (`gtk.HPaned`) or vertically (`gtk.VPaned`). Child widgets are added to the panes of the widget with the `pack1()` and pack2() methods. The division between the two children is set by default from the size requests of the children, but it can be adjusted by the user.

A paned widget draws a separator between the two child widgets and a small handle that the user can drag to adjust the division. It does not draw any relief around the children or around the separator. (The space in which the separator located is called the gutter.) Often, it is useful to put each child inside a `gtk.Frame` with the shadow type set to `gtk.SHADOW_IN` so that the gutter appears as a ridge.

Each child has two options that can be set, resize and shrink. If resize is `TRUE`, when the `gtk.Paned` is resized, that child will expand or shrink along with the paned widget. If shrink is `TRUE`, the child can be made smaller than it's requisition. Setting shrink to `FALSE` allows the application to set a minimum size. If resize is `FALSE` for both children, the resizing behaves as if resize is `TRUE` for both children. The application can set the position of the slider by calling the `set_position()` method.

Methods

gtk.Paned.add1

```
    def add1(child)
```

`child` :	the widget to add

The `add1()` method adds the widget specified by *`child`* to the top or left pane with the default packing parameters (resize is `FALSE` and shrink is `TRUE`). See the `pack1()` method.

gtk.Paned.add2

```
    def add2(child)
```

`child` :	the widget to add

The `add2()` method adds the widget specified by *`child`* to the bottom or right pane with the default packing parameters (resize is `TRUE` and shrink is `TRUE`). See the `pack2()` method.

gtk.Paned.pack1

```
    def pack1(child, resize=FALSE, shrink=TRUE)
```

`child` :	the widget to add
`resize` :	if `TRUE` *`child`* should resize when the paned is resized
`shrink` :	if `TRUE` *`child`* can be made smaller than its minimum size request

The `pack1()` method adds the widget specified by *`child`* to the top or left pane with the parameters specified by *`resize`* and *`shrink`*. If *`resize`* is `TRUE` *`child`* should be resized when the paned widget is resized. If *`shrink`* is `TRUE` *`child`* can be made smaller than its minimum size request.

gtk.Paned.pack2

```
    def pack2(child, resize=TRUE, shrink=TRUE)
```

`child` :	the widget to add
`resize` :	if `TRUE` *`child`* should resize when the paned is resized
`shrink` :	if `TRUE` *`child`* can be made smaller than its minimum size request

The `pack2()` method adds the widget specified by *`child`* to the bottom or right pane with the parameters specified by *`resize`* and *`shrink`*. If *`resize`* is `TRUE` *`child`* should be resized when the paned widget is resized. If *`shrink`* is `TRUE` *`child`* can be made smaller than its minimum size request.

gtk.Paned.get_position

```
    def get_position()
```

Returns :	the position of the divider

The `get_position()` method returns the position of the divider in pixels between the two panes.

gtk.Paned.set_position

```
    def set_position(position)
```

`position` :	the new pixel position of divider, a negative value means that the position is unset.

The `set_position()` method sets the position of the divider between the two panes to the value specified by *`position`* and sets the "position–set" property to `TRUE` if *`position`* is non–negative. If *`position`* is negative the divider position will be recalculated by the paned widget using the child widget requisitions and the "position–set" property will be set to `FALSE`.

gtk.Paned.compute_position

```
    def compute_position(allocation, child1_req, child2_req)
```

`allocation` :	the total space allocation in pixels for the paned widget
`child1_req` :	the minimum space in pixels required for the left or top child widget
`child2_req` :	the minimum space in pixels required for the right or bottom child widget

The `compute_position()` method computes the position of the separator according to the specification of the parameters: *`allocation`*, *`child1_req`* and *`child2_req`*. The calculation is affected by the packing parameters of the child widgets depending on whether they can resize and shrink. This method is used by subclasses of `gtk.Paned` and is usually not needed by applications. the `gtk.Widget.queue_resize()` method must be called after this method to have the resizing displayed.

The minimum position is 0 if child1's shrink value is `TRUE` or the value of *`child1_req`*, if `FALSE`. The maximum position is the value of *`allocation`* if child2's shrink value is TRUE, or the value of (*`allocation`* – *`child2_req`*), if `FALSE`. The final calculated position will be between the minimum and maximum positions.

gtk.Paned.get_child1

```
    def get_child1()
```

Returns :	the first child, or`None`

Note

This method is available in PyGTK 2.4 and above.

The `get_child1()` method returns the first child of the paned widget.

gtk.Paned.get_child2

```
def get_child2()
```

Returns :	the second child, or `None`

Note

This method is available in PyGTK 2.4 and above.

The `get_child2()` method returns the second child of the paned widget.

Signals

The "accept−position" gtk.Paned Signal

```
def callback(paned, user_param1, ...)
```

`paned` :	the paned that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled

The "accept−position" signal is emitted when *`paned`* has the focus and any of the **Return**, **Enter**, **Space** keys are pressed. This will also cause the child widget with the focus to be activated.

The "cancel−position" gtk.Paned Signal

```
def callback(paned, user_param1, ...)
```

`paned` :	the paned that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled

The "cancel−position" signal is emitted when the **Esc** key is pressed while *`paned`* has the focus.

The "cycle−child−focus" gtk.Paned Signal

```
def callback(paned, reversed, user_param1, ...)
```

`paned` :	the paned that received the signal
`reversed` :	if `TRUE` the focus cycle direction should be reversed
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled

The "cycle−child−focus" signal is emitted when **F6** or **Shift+F6** is pressed while *`paned`* has the focus.

The "cycle−handle−focus" gtk.Paned Signal

```
def callback(paned, reversed, user_param1, ...)
```

paned :	the paned that received the signal
reversed :	if `TRUE` the focus cycle direction should be reversed
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled

The "cycle–handle–focus" signal is emitted when *paned* has the focus and any of the **Tab**, **Ctrl+Tab**, **Shift+Tab** or **Ctrl+Shift+Tab** keys combinations are pressed. **Tab** and **Ctrl+Tab** set *reversed* to `FALSE` while **Shift+Tab** and **Ctrl+Shift+Tab** set *reversed* to `TRUE`.

The "move–handle" gtk.Paned Signal

```
    def callback(paned, scrolltype, user_param1, ...)
```

paned :	the paned that received the signal
scrolltype :	the scroll type: `gtk.SCROLL_NONE`, `gtk.SCROLL_JUMP`, `gtk.SCROLL_STEP_BACKWARD`, `gtk.SCROLL_STEP_FORWARD`, `gtk.SCROLL_PAGE_BACKWARD`, `gtk.SCROLL_PAGE_FORWARD`, `gtk.SCROLL_STEP_UP`, `gtk.SCROLL_STEP_DOWN`, `gtk.SCROLL_PAGE_UP`, `gtk.SCROLL_PAGE_DOWN`, `gtk.SCROLL_STEP_LEFT`, `gtk.SCROLL_STEP_RIGHT`, `gtk.SCROLL_PAGE_LEFT`, `gtk.SCROLL_PAGE_RIGHT`, `gtk.SCROLL_START` or `gtk.SCROLL_END`
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled

The "move–handle" signal is emitted when *paned* has the focus and one of the following key combinations is pressed to move the separator:

- **Left Arrow**
- **Right Arrow**
- **Up Arrow**
- **Down Arrow**
- **Page Up**
- **Page Down**
- **Home**
- **End**

The value of *scrolltype* is one of: `gtk.SCROLL_NONE`, `gtk.SCROLL_JUMP`, `gtk.SCROLL_STEP_BACKWARD`, `gtk.SCROLL_STEP_FORWARD`, `gtk.SCROLL_PAGE_BACKWARD`, `gtk.SCROLL_PAGE_FORWARD`, `gtk.SCROLL_STEP_UP`, `gtk.SCROLL_STEP_DOWN`, `gtk.SCROLL_PAGE_UP`, `gtk.SCROLL_PAGE_DOWN`, `gtk.SCROLL_STEP_LEFT`, `gtk.SCROLL_STEP_RIGHT`, `gtk.SCROLL_PAGE_LEFT`, `gtk.SCROLL_PAGE_RIGHT`, `gtk.SCROLL_START` or `gtk.SCROLL_END`. The default handler for this signal moves the separator if the separator has the focus.

The "toggle–handle–focus" gtk.Paned Signal

```
    def callback(paned, user_param1, ...)
```

paned :	the paned that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method

`...` :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled

The "toggle−handle−focus" signal is emitted when *paned* has the focus and **F8** is pressed to give the focus to or take the focus from the separator handle.

Prev Up Next
gtk.OptionMenu Home gtk.Plug

gtk.Plug
Prev **The gtk Class Reference** Next

gtk.Plug

gtk.Plug A toplevel window for embedding into other processes.

Synopsis

```
class gtk.Plug(gtk.Window):
    gtk.Plug(socket_id)
    def construct(socket_id)
    def get_id()
Functions

    def gtk.plug_new_for_display(display, socket_id)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Window
            +-- gtk.Plug
```

Signal Prototypes

"embedded"	`def callback(`*plug*`, `*user_param1*`, ...)`

Description

Together with `gtk.Socket`, `gtk.Plug` provides the ability to embed widgets from one process into another process in a fashion that is transparent to the user. One process creates a `gtk.Socket` widget and, passes the ID of that widgets window to the other process, which then creates a `gtk.Plug` with that window ID. Any widgets contained in the `gtk.Plug` then will appear inside the first applications window.

Constructor

```
   gtk.Plug(socket_id)
```

`socket_id` :	the window ID of the socket, or 0.

Returns :	a gtk.Widget

Creates a new gtk.Plug widget inside the gtk.Socket identified by *socket_id*. If *socket_id* is 0, the plug is left "unplugged" and can later be plugged into a gtk.Socket by the gtk.Socket.add_id() method.

Methods

gtk.Plug.construct

```
def construct(socket_id)
```

socket_id :	the window ID of the socket

Warning

This method is *not* available in PyGTK 2.2 and above.

The construct() method finishes the initialization of plug for the gtk.Socket identified by *socket_id*. This method will generally only be used by subclasses of gtk.Plug.

gtk.Plug.get_id

```
def get_id()
```

Returns :	the window ID for the plug

The get_id() method returns the window ID of the gtk.Plug widget, which can be used to embed this window inside another window, for instance with gtk.Socket.add_id().

Functions

gtk.plug_new_for_display

```
def gtk.plug_new_for_display(display, socket_id)
```

display :	the gtk.gdk.Display associated with *socket_id's*.
socket_id :	the window ID of the socket's window.
Returns :	a gtk.Plug object

Note

This function is available in PyGTK 2.2 and above.

The gtk.plug_new_for_display() function creates a new plug widget inside the gtk.Socket specified by *socket_id* on the gtk.gdk.Display specified by display.

Signals

The "embedded" gtk.Plug Signal

```
  def callback(plug, user_param1, ...)
```

plug :	the plug that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "embedded" signal is emitted when the plug window is reparented to the socket window.

Prev Up Next
gtk.Paned Home gtk.ProgressBar

gtk.ProgressBar
Prev **The gtk Class Reference** Next

gtk.ProgressBar

gtk.ProgressBar a widget which indicates progress visually.

Synopsis

```
class gtk.ProgressBar(gtk.Progress):
    gtk.ProgressBar()
    def pulse()
    def set_text(text)
    def set_fraction(fraction)
    def set_pulse_step(fraction)
    def set_orientation(orientation)
    def get_text()
    def get_fraction()
    def get_pulse_step()
    def get_orientation()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Progress
        +-- gtk.ProgressBar
```

Properties

"fraction"	Read–Write	The fraction of total work that has been completed
"pulse–step"	Read–Write	The fraction of total progress to move the bouncing block when pulsed
"orientation"	Read–Write	The orientation and growth direction of the progress bar: `gtk.PROGRESS_LEFT_TO_RIGHT`, `gtk.PROGRESS_RIGHT_TO_LEFT`, `gtk.PROGRESS_BOTTOM_TO_TOP`, `gtk.PROGRESS_TOP_TO_BOTTOM`
"text"	Read–Write	The text to be displayed in the progress bar

Description

The gtk.ProgressBar is typically used to display the progress of a long running operation. It provides a visual clue that processing is underway. The gtk.ProgressBar can be used in two different modes: percentage mode and activity mode.

When an application can determine how much work needs to take place (e.g. read a fixed number of bytes from a file) and can monitor its progress, it can use the gtk.ProgressBar in percentage mode and the user sees a growing bar indicating the percentage of the work that has been completed. In this mode, the application is required to call the set_fraction() method periodically to update the progress bar.

When an application has no accurate way of knowing the amount of work to do, it can use the gtk.ProgressBar in activity mode, which shows activity by a block moving back and forth within the progress area. In this mode, the application is required to call the pulse() method periodically to update the progress bar.

There is quite a bit of flexibility provided to control the appearance of the gtk.ProgressBar. Methods are provided to control the orientation of the bar, optional text can be displayed along with the bar, and the step size used in activity mode can be set.

Constructor

```
    gtk.ProgressBar()
```

Returns :	a new gtk.ProgressBar widget

Creates a new gtk.Progressbar widget.

Methods

gtk.ProgressBar.pulse

```
    def pulse()
```

The `pulse`() method nudges the progressbar to indicate that some progress has been made, but you don't know how much. This method also changes progress bar mode to "activity mode," where a block bounces back and forth. Each call to the pulse() method causes the block to move by a little bit (the amount of movement per pulse is determined by the set_pulse_step() method).

gtk.ProgressBar.set_text

```
    def set_text(text)
```

`text` :	a UTF−8 string

The `set_text`() method superimposes the text specified by *`text`* on the progress bar.

gtk.ProgressBar.set_fraction

```
    def set_fraction(fraction)
```

`fraction` :	the fraction of the task that's been completed

The `set_fraction()` method causes the progress bar to "fill in" the portion of the bar specified by *`fraction`*. The value of *`fraction`* should be between 0.0 and 1.0.

gtk.ProgressBar.set_pulse_step

```
    def set_pulse_step(fraction)
```

`fraction` :	a value between 0.0 and 1.0

The `set_pulse_step()` method sets the portion (specified by *`fraction`*) of the total progress bar length to move the bouncing block for each call to the `pulse()` method.

gtk.ProgressBar.set_orientation

```
    def set_orientation(orientation)
```

`orientation` :	the orientation of the progress bar

The `set_orientation()` method switches the progress bar to a different orientation as specified by the value of *`orientation`*. The value of *`orientation`* must be one of:

`gtk.PROGRESS_LEFT_TO_RIGHT`	A horizontal progress bar growing from left to right.
`gtk.PROGRESS_RIGHT_TO_LEFT`	A horizontal progress bar growing from right to left.
`gtk.PROGRESS_BOTTOM_TO_TOP`	A vertical progress bar growing from bottom to top.
`gtk.PROGRESS_TOP_TO_BOTTOM`	A vertical progress bar growing from top to bottom.

gtk.ProgressBar.get_text

```
    def get_text()
```

Returns :	the text, or `None`

The `get_text()` method returns the text superimposed on the progress bar. If there is no superimposed text this method returns `None`.

gtk.ProgressBar.get_fraction

```
    def get_fraction()
```

Returns :	a fraction from 0.0 to 1.0

The `get_fraction()` method returns the current fraction of the task that's been set by the `set_fraction()` method .

gtk.ProgressBar.get_pulse_step

```
    def get_pulse_step()
```

Returns :	a fraction from 0.0 to 1.0

The `get_pulse_step()` method returns the pulse step set with the `set_pulse_step()`.

gtk.ProgressBar.get_orientation

```
    def get_orientation()
```

Returns :	the orientation of the progress bar

The `get_orientation()` method returns the current progress bar orientation. See the `set_orientation()` method for information about the orientation values: `gtk.PROGRESS_LEFT_TO_RIGHT`, `gtk.PROGRESS_RIGHT_TO_LEFT`, `gtk.PROGRESS_BOTTOM_TO_TOP`, `gtk.PROGRESS_TOP_TO_BOTTOM`.

gtk.RadioAction

gtk.RadioAction — an action that can be grouped so that only one can be active (new in PyGTK 2.4)

Synopsis

```
class gtk.RadioAction(gtk.ToggleAction):
    gtk.RadioAction(name, label, tooltip, stock_id, value)
    def set_group(group)
    def get_group()
    def get_current_value()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Action
    +-- gtk.ToggleAction
      +-- gtk.RadioAction
```

Properties

Note

These properties are available in GTK+ 2.4 and above.

"group"	Write	Sets a new group for a radio action.
"value"	Read–Write	The value is an arbitrary integer which can be used as a convenient way to determine which action in the group is currently active in an "activate" or "changed" signal handler. See the `get_current_value()` and the `gtk.ActionGroup.add_radio_actions()` methods for convenient ways to get and set this property.

Signal Prototypes

Note

This signal is available in GTK+ 2.4 and above.

"changed"	`def callback(`*`radioaction, current, user_param1, ...`*`)`

Description

Note

This object is available in PyGTK 2.4 and above.

A `gtk.RadioAction` is a subclass of `gtk.ToggleAction` and similar to `gtk.RadioMenuItem`. A number of radio actions can be linked together so that only one may be active at any one time.

Constructor

```
gtk.RadioAction(name, label, tooltip, stock_id, value)
```

`name` :	A unique name for the action
`label` :	The label displayed in menu items and on buttons
`tooltip` :	A tooltip for this action
`stock_id` :	The stock icon to display in widgets representing this action
`value` :	A unique integer value that `get_current_value()` should return if this action is selected.
Returns :	a new `gtk.RadioAction`

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new `gtk.RadioAction` object suing the properties specified by: *`name`*, *`label`*, *`tooltip`*, *`stock_id`* and *`value`*. To add the action to a `gtk.ActionGroup` and set the accelerator for the action, call the `gtk.ActionGroup.add_action_with_accel()`.

Methods

gtk.RadioAction.set_group

```
def set_group(group)
```

`group` :	another `gtk.RadioAction`

Note

This method is available in PyGTK 2.4 and above.

The `set_group()` method sets the radio group for the radio action to the same group as the `gtk.RadioAction` specified by *`group`* i.e. the radio action joins the group.

gtk.RadioAction.get_group

```
def get_group()
```

Returns :	a list containing the radio actions in the group

Note

This method is available in PyGTK 2.4 and above.

The `get_group()` method returns a list containing the group that the radio action belongs to.

gtk.RadioAction.get_current_value

```
def get_current_value()
```

Returns :	The value of the currently active group member

Note

This method is available in PyGTK 2.4 and above.

The `get_current_value()` method returns the "value" property of the the currently active member of the group that the radio action belongs to.

Signals

The "changed" gtk.RadioAction Signal

```
def callback(radioaction, current, user_param1, ...)
```

radioaction :	the radioaction that received the signal
current :	the currently active `gtk.RadioAction` in the group
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.4 and above.

The "changed" signal is emitted on every member of a radio group when the active member is changed. The signal gets emitted after the "activate" signals for the previous and current active members.

gtk.RadioButton

gtk.RadioButton — a choice of one of multiple check buttons.

Synopsis

```
class gtk.RadioButton(gtk.CheckButton):
    gtk.RadioButton(group=None, label=None, use_underline=TRUE)
    def get_group()
    def set_group(group)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Button
            +-- gtk.ToggleButton
              +-- gtk.CheckButton
                +-- gtk.RadioButton
```

Properties

"group"	Write	The group that the radiobutton is in.

Signal Prototypes

"group–changed"	`def callback(`*radiobutton*`,` *user_param1*`, ...)`

Description

A single `gtk.RadioButton` performs the same basic function as a `gtk.CheckButton`, as it's position in the object hierarchy reflects. It is only when multiple radio buttons are grouped together that they become a different user interface component in their own right. Every radio button is a member of some group of radio buttons. When one is selected, all other radio buttons in the same group are deselected. A `gtk.RadioButton` is used to give the user a choice of one of many options.

Radio button widgets are created with gtk.RadioButton() passing `None` as the *group* (first) argument if this is the first radio button in a group. In subsequent calls, pass a reference to a `gtk.RadioButton` as the first argument to specify the group. The second (optional) argument to gtk.RadioButton() is a label that is used to specify the text of the button. The label text is parsed for mnemonic characters (preceded by an underscore) to specify an accelerator for the radiobutton. See `gtk.AccelGroup` and `gtk.AccelLabel` for more information on mnemonic accelerators.

To retrieve the group a `gtk.RadioButton` is assigned to, use the `get_group()` method. To remove a `gtk.RadioButton` from one group and make it part of a new one, use the `set_group()` method.

Constructor

```
    gtk.RadioButton(group=None, label=None, use_underline=TRUE)
```

group :	an existing `gtk.RadioButton` or `None`
label :	a string to use as the button text or `None`
use_underline :	if `TRUE`, an underscore in the label text indicates the next character should be underlined and used for the mnemonic accelerator key if it is the first character so marked. Available in PyGTK 2.4 and above.
Returns :	a new `gtk.RadioButton` widget

Creates a new `gtk.RadioButton` widget with the label text specified by *`label`*, adding it to the same group as *`group`*. *`label`* is parsed for underscore characters that indicate mnemonic accelerators. If *`label`* is `None`, no label is created. If *`group`* is `None`, the new radiobutton becomes the first member of a new radiobutton group.

In PyGTK 2.4 and above the *`use_underline`* parameter is available and defaults to `TRUE`. If *`use_underline`* is set to `FALSE` the label text will not be parsed for mnemonic characters.

Methods

gtk.RadioButton.get_group

```
def get_group()
```

Returns :	the list of radiobuttons in the same group that contains the radiobutton

The `get_group()` method returns the list of radiobuttons that are in the same group as the radiobutton.

gtk.RadioButton.set_group

```
def set_group(group)
```

`group` :	a `gtk.RadioButton` whose group the radiobutton will be added to

The `set_group()` method adds the radiobutton to the group of the radiobutton specified by *`group`*.

Signals

The "group−changed" gtk.RadioButton Signal

```
def callback(radiobutton, user_param1, ...)
```

`radiobutton` :	the radiobutton that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.4 and above.

The "group−changed" signal is emitted when a `gtk.RadioButton` is added to or removed from the group.

Prev Up Next
gtk.RadioAction Home gtk.RadioMenuItem

gtk.RadioMenuItem

gtk.RadioMenuItem — a choice from multiple check menu items.

Synopsis

```
class gtk.RadioMenuItem(gtk.CheckMenuItem):
    gtk.RadioMenuItem(group=None, label=None, use_underline=TRUE)
    def get_group()
    def set_group(group)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Item
            +-- gtk.MenuItem
              +-- gtk.CheckMenuItem
                +-- gtk.RadioMenuItem
```

Signal Prototypes

"group–changed"	`def callback(radiomenuitem, user_param1, ...)`

Description

A `gtk.RadioMenuItem` widget is a check menu item that belongs to a group. Only one of the radio menu items in a group can be selected.

Constructor

```
    gtk.RadioMenuItem(group=None, label=None, use_underline=TRUE)
```

`group` :	a `gtk.RadioMenuItem` whose group the new radiomenuitem should be added to, or `None` if a new group should be created
`label` :	a string to be used as the label text or `None` if no label is needed
`use_underline` :	if `TRUE`, an underscore in the label text indicates the next character should be underlined and used for the mnemonic accelerator key if it is the first character so marked. Available in PyGTK 2.4 and above.
Returns :	a new `gtk.RadioMenuItem` widget

Creates a new `gtk.RadioMenuItem` containing a label with its text specified by *`label`*. *`label`* will be parsed for underscores that indicate the mnemonic accelerator for the radiomenuitem. The radiomenuitem will be added to the group containing the `gtk.RadioMenuItem` specified by *`group`*. If *`group`* is `None`, a new group will be created to hold the new radiomenuitem. If *`label`* is `None` the radiomenuitem is created without a label.

In PyGTK 2.4 and above the *`use_underline`* parameter is available and defaults to `TRUE`. If *`use_underline`* is set to `FALSE` the label text will not be parsed for mnemonic characters.

Methods

gtk.RadioMenuItem.get_group

```
def get_group()
```

Returns :	the list of radiomenuitems in the same group that contains the radiomenuitem

The `get_group()` method returns the list of `gtk.RadioMenuItem`s that are in the same group as the radiomenuitem.

gtk.RadioMenuItem.set_group

```
def set_group(group)
```

`group` :	a `gtk.RadioMenuItem` whose group the radiomenuitem will be added to

The `set_group()` method adds the radiomenuitem to the group of the `gtk.RadioMenuItem` specified by *`group`*.

Signals

The "group−changed" gtk.RadioMenuItem Signal

```
def callback(radiomenuitem, user_param1, ...)
```

`radiomenuitem` :	the radiomenuitem that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.4 and above.

The "group−changed" signal is emitted when a `gtk.RadioMenuItem` is added to or removed from the group.

gtk.RadioToolButton

gtk.RadioToolButton — a toolbar item that contains a radio button (new in PyGTK 2.4)

Synopsis

```
class gtk.RadioToolButton(gtk.ToggleToolButton):
    gtk.RadioToolButton(group=None, stock_id=None)
    def set_group(group)
    def get_group()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.ToolItem
            +-- gtk.ToolButton
              +-- gtk.ToggleToolButton
                +-- gtk.RadioToolButton
```

Properties

Note

This property is available in GTK+ 2.4 and above.

"group"	Write	Sets a new group for a radio tool button.

Description

Note

This widget is available in PyGTK 2.4 and above.

A `gtk.RadioToolButton` is a `gtk.ToolItem` that contains a radio button, that is, a button that is part of a group of toggle buttons where only one button can be active at a time. Use the gtk.RadioToolButton() constructor to create a new `gtk.RadioToolButton`.

Constructor

```
  gtk.RadioToolButton(group=None, stock_id=None)
```

`group` :	an existing `gtk.RadioToolButton`
`stock_id` :	the name of a stock item
Returns :	a new `gtk.RadioToolButton`

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new `gtk.RadioToolButton`, adding it to the same group as the `gtk.RadioToolButton` specified by *`group`* if it is not `None`. The new `gtk.RadioToolButton` will contain an icon and label from the stock item specified by *`stock_id`* if it is not `None`.

Methods

gtk.RadioToolButton.set_group

```
    def set_group(group)
```

group :	an existing gtk.RadioToolButton

Note

This method is available in PyGTK 2.4 and above.

The `set_group`() method adds the radio tool button to the same group as the gtk.RadioToolButton specified by *group*

gtk.RadioToolButton.get_group

```
    def get_group()
```

Returns :	a list containing the gtk.RadioToolButtons in the group

Note

This method is available in PyGTK 2.4 and above.

The `get_group`() method returns a list containing the gtk.RadioButton objects that are in the same group as the radio tool button.

Prev	Up	Next
gtk.RadioMenuItem	Home	gtk.Range

gtk.Range

Prev **The gtk Class Reference** Next

gtk.Range

gtk.Range — a base class for widgets that allow a user to set a value in a range.

Synopsis

```
class gtk.Range(gtk.Widget):
    def set_update_policy(policy)
    def get_update_policy()
    def set_adjustment(adjustment)
    def get_adjustment()
    def set_inverted(setting)
    def get_inverted()
    def set_increments(step, page)
    def set_range(min, max)
    def set_value(value)
    def get_value()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Range
```

Properties

"adjustment"	Read–Write	The `gtk.Adjustment` that contains the current value of this range object
"inverted"	Read–Write	If `TRUE`, the slider movement is reversed e.g right–to–left instead of left–to–right
"update–policy"	Read–Write	How the range should be updated on the screen: `gtk.UPDATE_CONTINUOUS`, `gtk.UPDATE_DISCONTINUOUS`, `gtk.UPDATE_DELAYED`.

Style Properties

"arrow–displacement–x"	Read	The distance horizontally to move the arrow when the button is depressed
"arrow–displacement–y"	Read	The distance vertically to move the arrow when the button is depressed
"slider–width"	Read	The width of scrollbar or scale thumb
"stepper–size"	Read	The length of step buttons at ends
"stepper–spacing"	Read	The spacing between step buttons and thumb
"trough–border"	Read	The spacing between thumb or steppers and outer trough bevel

Signal Prototypes

"adjust–bounds"	`def callback(`*`range`*`, `*`value`*`, `*`user_param1`*`, ...)`
"move–slider"	`def callback(`*`range`*`, `*`scrolltype`*`, `*`user_param1`*`, ...)`
"value–changed"	`def callback(`*`range`*`, `*`user_param1`*`, ...)`

Description

The `gtk.Range` is the base class for the `gtk.Scale` and `gtk.Scrollbar` widgets. The `gtk.Range` provides the common functionality for these widgets that allow a user to set a value in a range of values. The `gtk.Scale` works in conjunction with a `gtk.Adjustment` which provides the range information.

Methods

gtk.Range.set_update_policy

```
def set_update_policy(policy)
```

`policy` :	the update policy: `gtk.UPDATE_CONTINUOUS`, `gtk.UPDATE_DISCONTINUOUS`, `gtk.UPDATE_DELAYED`

The `set_update_policy()` method sets the "update–policy" property to the value specified by *`policy`*. The update policy has the following values and effects:

`gtk.UPDATE_CONTINUOUS`	anytime the range slider is moved, the range value will change and the "value_changed" signal will be emitted.
`gtk.UPDATE_DELAYED`	the value will be updated after a brief timeout where no slider motion occurs, so value changes are delayed slightly rather than continuously updated.
`gtk.UPDATE_DISCONTINUOUS`	

	the value will only be updated when the user releases the button and ends the slider drag operation.

gtk.Range.get_update_policy

```
    def get_update_policy()
```

Returns :	the current update policy

The `get_update_policy()` method gets the value of the "update–policy" property. The update policy is one of: `gtk.UPDATE_CONTINUOUS`, `gtk.UPDATE_DISCONTINUOUS` or `gtk.UPDATE_DELAYED`. See the `set_update_policy()` method for details.

gtk.Range.set_adjustment

```
    def set_adjustment(adjustment)
```

`adjustment` :	a `gtk.Adjustment`

The `set_adjustment()` method sets the "adjustment" property to the value specified by *`adjustment`*. The `gtk.Adjustment` is used as the "model" object for this range widget. *`adjustment`* indicates the current range value, the minimum and maximum range values, the step and page increments used for keybindings and scrolling, and the page size. The page size is normally 0 for `gtk.Scale` and nonzero for `gtk.Scrollbar`, and indicates the size of the visible area of the widget being scrolled. The page size affects the size of the scrollbar slider.

gtk.Range.get_adjustment

```
    def get_adjustment()
```

Returns :	a `gtk.Adjustment`

The `get_adjustment()` method returns the value of the "adjustment" property. See the `set_adjustment()` method for details.

gtk.Range.set_inverted

```
    def set_inverted(setting)
```

`setting` :	if `TRUE` invert the range

The `set_inverted()` method sets the "inverted" property to the value specified by *`setting`*. If *`setting`* is `TRUE` the normal motion of the range widget is reversed. Ranges normally move from lower to higher values as the slider moves from top to bottom or left to right. Inverted ranges have higher values at the top or left rather than on the bottom or right.

gtk.Range.get_inverted

```
    def get_inverted()
```

Returns :	`TRUE` if the range is inverted

The `get_inverted()` method returns the value of the "inverted" property that was set by the `set_inverted()` method.

gtk.Range.set_increments

```
def set_increments(step, page)
```

step :	the step size
page :	the page size

The `set_increments()` method sets the step and page sizes for the range to the values specified by *step* and *page* respectively. The step size is used when the user clicks the `gtk.Scrollbar` arrows or moves `gtk.Scale` via the arrow keys. The page size is used for example when moving via **Page Up** or **Page Down** keys.

gtk.Range.set_range

```
def set_range(min, max)
```

min :	the minimum range value
max :	the maximum range value

The `set_range()` method sets the minimum and maximum allowable values for the `gtk.Range` to that values specified by *min* and *max* respectively. If the range has a non−zero page size, it is also forced to be between *min* and *max*.

gtk.Range.set_value

```
def set_value(value)
```

value :	the new value of the range

The `set_value()` method sets the current value of the range to the value specified by *value*. *value* will be forced inside the minimum or maximum range values. The range emits the "value_changed" signal if the value changes.

gtk.Range.get_value

```
def get_value()
```

Returns :	the current value of the range.

The `get_value()` method gets the current value of the range.

Signals

The "adjust−bounds" gtk.Range Signal

```
def callback(range, value, user_param1, ...)
```

range :	the range that received the signal
value :	the value
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "adjust−bounds" signal is emitted when the range is adjusted by user action. Note the value can be more or less than the range since it depends on the mouse position.

The "move−slider" gtk.Range Signal

```
    def callback(range, scrolltype, user_param1, ...)
```

range :	the range that received the signal
scrolltype :	the scroll type issued because a key was pressed by the user; one of: `gtk.SCROLL_NONE`, `gtk.SCROLL_JUMP`, `gtk.SCROLL_STEP_BACKWARD`, `gtk.SCROLL_STEP_FORWARD`, `gtk.SCROLL_PAGE_BACKWARD`, `gtk.SCROLL_PAGE_FORWARD`, `gtk.SCROLL_STEP_UP`, `gtk.SCROLL_STEP_DOWN`, `gtk.SCROLL_PAGE_UP`, `gtk.SCROLL_PAGE_DOWN`, `gtk.SCROLL_STEP_LEFT`, `gtk.SCROLL_STEP_RIGHT`, `gtk.SCROLL_PAGE_LEFT`, `gtk.SCROLL_PAGE_RIGHT`, `gtk.SCROLL_START` or `gtk.SCROLL_END`
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "move−slider" signal is emitted when the user presses a key (e.g. **Page Up**, **Home**, **Right Arrow**) to move the slider.

The "value−changed" gtk.Range Signal

```
    def callback(range, user_param1, ...)
```

range :	the range that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "value−changed" signal is emitted when the range value is changed either programmatically or by user action.

Prev	Up	Next
gtk.RadioToolButton	Home	gtk.RcStyle

gtk.RcStyle

Prev **The gtk Class Reference** Next

gtk.RcStyle

gtk.RcStyle — an object holding resource styles

Synopsis

```
class gtk.RcStyle(gobject.GObject):
    def copy()
Functions

    def gtk.rc_add_default_file(filename)
    def gtk.rc_set_default_files(filenames)
    def gtk.rc_get_default_files()
    def gtk.rc_get_style_by_paths(settings, widget_path, class_path, type)
    def gtk.rc_reparse_all_for_settings(settings, force_load)
    def gtk.rc_reset_styles(settings)
    def gtk.rc_parse(filename)
    def gtk.rc_parse_string(rc_string)
    def gtk.rc_reparse_all()
    def gtk.rc_find_module_in_path(module_file)
    def gtk.rc_get_theme_dir()
    def gtk.rc_get_module_dir()
    def gtk.rc_get_im_module_path()
    def gtk.rc_get_im_module_file()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.RcStyle
```

Description

PyGTK via GTK+ provides resource file mechanism for configuring various aspects of the operation of a program at runtime.

Default files

An application can cause GTK+ to parse a specific RC file by calling the gtk.rc_parse() function. In addition to this, certain files will be read at the end of GTK+ initialization. Unless modified, the files looked for will be <SYSCONFDIR>/gtk−2.0/gtkrc and .gtkrc−2.0 in the users home directory. (<SYSCONFDIR> defaults to /usr/local/etc.) The set of these default files can be retrieved with the gtk.rc_get_default_files() function and modified with the gtk.rc_add_default_file() and gtk.rc_set_default_files() functions. Additionally, the GTK_RC_FILES environment variable can be set to a list of files in order to overwrite the set of default files at runtime.

For each RC file, in addition to the file itself, GTK+ will look for a locale−specific file that will be parsed after the main file. For instance, if LANG is set to ja_JP.ujis, when loading the default file ~/.gtkrc then GTK+ looks for ~/.gtkrc.ja_JP and ~/.gtkrc.ja, and parses the first of those that exists.

Pathnames and patterns

A resource file defines a number of styles and key bindings and attaches them to particular widgets. The attachment is done by the widget, widget_class, and class declarations. As an example of such a statement:

```
  widget "mywindow.*.GtkEntry" style "my-entry-class"
```

attaches the style "my−entry−class" to all widgets whose widget class matches the pattern "mywindow.*.GtkEntry". The patterns here are given in the standard shell glob syntax. The "?" wildcard matches any character, while "*" matches zero or more of any character. The three types of matching are against the widget path, the class path and the class hierarchy. Both the widget and the class paths consists of a "." separated list of all the parents of the widget and the widget itself from outermost to innermost. The difference is that in the widget path, the name assigned by the set_name() method is used if present, otherwise the class name of the widget, while for the widget path, the class name is always used. So, if you have a gtk.Entry named "myentry", inside of a of a window named "mywindow", then the widget path is: "mwindow.GtkHBox.myentry" while the class path is: "GtkWindow.GtkHBox.GtkEntry".

Matching against class is a little different. The pattern match is done against all class names in the widgets class hierarchy (not the layout hierarchy) in sequence, so the pattern:

```
  class "GtkButton" style "my-style"
```

will match not just gtk.Button widgets, but also gtk.ToggleButton and gtk.CheckButton widgets, since those classes derive from gtk.Button.

Toplevel declarations

An RC file is a text file which is composed of a sequence of declarations. '#' characters delimit comments and the portion of a line after a '#' is ignored when parsing an RC file. The possible toplevel declarations are:

binding *name* { ... }	Declares a binding set.
class *pattern* [style | binding [: priority]] *name*	Specifies a style or binding set for a particular branch of the inheritance hierarchy.
include *filename*	Parses another file at this point. If *filename* is not an absolute filename, it is searched in the directories of the currently open RC files. `GTK+` also tries to load a locale−specific variant of the included file.
module_path *path*	Sets a path (a list of directories separated by colons) that will be searched for theme engines referenced in RC files.
pixmap_path *path*	Sets a path (a list of directories separated by colons) that will be searched for pixmaps referenced in RC files.
style *name* [= *parent*] { ... }	Declares a style.
widget *pattern* [style | binding [: priority]] *name*	Specifies a style or binding set for a particular group of widgets by matching on the widget pathname.
widget_class *pattern* [style | binding [: priority]] *name*	Specifies a style or binding set for a particular group of widgets by matching on the class pathname.

Styles

A RC style is specified by a style declaration in a RC file, and then bound to widgets with a `widget`, `widget_class`, or `class` declaration. All styles applying to a particular widget are composited together with widget declarations overriding `widget_class` declarations which, in turn, override `class` declarations. Within each type of declaration, later declarations override earlier ones. Within a style declaration, the possible elements are:

bg[*state*] = *color*	Sets the color used for the background of most widgets.
fg[*state*] = *color*	Sets the color used for the foreground of most widgets.
base[*state*] = *color*	Sets the color used for the background of widgets displaying editable text. This color is used for the background of, among others, `gtk.TextView` and `gtk.Entry`.
text[*state*] = *color*	Sets the color used for foreground of widgets using *base* for the background color.
bg_pixmap[*state*] = *pixmap*	Sets a background pixmap to be used in place of the `bg` color (or for `gtk.TextView`, in place of the `base` color). The special value "<parent>" may be used to indicate that the widget should use the same background pixmap as its parent. The special value "<none>" may be used to indicate no background pixmap.
font = *font*	Sets the font for a widget. font must be a XLFD font description, e.g. "−*−helvetica−medium−r−normal−−10−*−*−*−*−*−*−*".
fontset = *font*	Sets the fontset for a widget. Overrides any font declarations. font must be a comma−separated list of XLFD font descriptions, e.g. "−JIS−Fixed−Medium−R−Normal−−26−180−100−100−C−240, −JIS−Fixed−Medium−R−Normal−−26−180−100−100−C−120, −GB−Fixed−Medium−R−Normal−−26−180−100−100−C−240, −Adobe−Courier−Bold−R−Normal−−25−180−100−100−M−150".
font_name = *font*	

	Sets the font for a widget. Overrides any `font` or `fontset` declarations. *`font`* must be a Pango font name, e.g. "Sans Italic 10".
stock["*`stock-id`*"] = { *`icon source specifications`* }	Defines the icon for a stock item.
engine "*`engine`*" { *`engine-specific settings`* }	Defines the engine to be used when drawing with this style.

The colors and background pixmaps are specified as a function of the state of the widget. The states are:

`NORMAL`	A color used for a widget in its normal state.
`ACTIVE`	A variant of the `NORMAL` color used when the widget is in the `gtk.STATE_ACTIVE` state, and also for the trough of a `gtk.Scrollbar`, tabs of a `gtk.Notebook` other than the current tab and similar areas. Frequently, this should be a darker variant of the `NORMAL` color.
`PRELIGHT`	A color used for widgets in the `gtk.STATE_PRELIGHT` state. This state is the used for `gtk.Button` and `gtk.MenuItem` widgets that have the mouse cursor over them, and for their children.
`SELECTED`	A color used to highlight data selected by the user. for instance, the selected items in a list widget, and the selection in an editable widget.
`INSENSITIVE`	A color used for the background of widgets that have been set insensitive with the `set_sensitive()` method.

Colors can be specified as a string containing a color name (from the X color database `/usr/lib/X11/rgb.txt`), in one of the hexadecimal forms `#rrrrggggbbbb`, `#rrrgggbbb`, `#rrggbb`, or `#rgb`, where `r`, `g` and `b` are hex digits, or they can be specified as a triplet `{ r, g, b}`, where `r`, `g` and `b` are either integers in the range 0–65635 or floats in the range 0.0–1.0.

In a stock definition, icon sources are specified as a 4–tuple of image filename, text direction, widget state, and size, in that order. Each icon source specifies an image filename to use with a given direction, state, and size. The * character can be used as a wildcard, and if direction–state–size are omitted they default to *. So for example, the following specifies different icons to use for left–to–right and right–to–left languages:

```
  stock["my-stock-item"] =
  {
    { "itemltr.png", LTR, *, * },
    { "itemrtl.png", RTL, *, * }
  }
```

This could be abbreviated as follows:

```
  stock["my-stock-item"] =
  {
    { "itemltr.png", LTR },
    { "itemrtl.png", RTL }
  }
```

You can specify custom icons for specific sizes, as follows:

```
  stock["my-stock-item"] =
  {
    { "itemmenusize.png", *, *, "gtk-menu" },
    { "itemtoolbarsize.png", *, *, "gtk-large-toolbar" }
    { "itemgeneric.png" } /* implicit *, *, * as a fallback */
  }
```

The sizes that come with `GTK+` itself are "gtk–menu", "gtk–small–toolbar", "gtk–large–toolbar", "gtk–button", "gtk–dialog". Applications can define other sizes. It's also possible to use custom icons for a given state, for example:

```
  stock["my-stock-item"] =
  {
    { "itemprelight.png", *, PRELIGHT },
    { "iteminsensitive.png", *, INSENSITIVE },
    { "itemgeneric.png" } /* implicit *, *, * as a fallback */
  }
```

When selecting an icon source to use, `GTK+` will consider text direction most important, state second, and size third. It will select the best match based on those criteria. If an attribute matches exactly (e.g. you specified `PRELIGHT` or specified the size), `GTK+` won't modify the image; if the attribute matches with a wildcard, `GTK+` will scale or modify the image to match the state and size the user requested.

Key bindings

Key bindings allow the user to specify actions to be taken on particular key presses. The form of a binding set declaration is:

```
  binding name {
    bind key {
      signalname (param, ...)
      ...
    }
    ...
  }
```

`key` is a string consisting of a series of modifiers followed by the name of a key. The modifiers can be:

- <alt>
- <control>
- <mod1>
- <mod2>
- <mod3>
- <mod4>
- <mod5>
- <release>
- <shft>
- <shift>

<shft> is an alias for <shift> and <alt> is an alias for <mod1>.

The action that is bound to the key is a sequence of signal names (strings) followed by parameters for each signal. The signals must be action signals. Each parameter can be a float, integer, string, or unquoted string representing an enumeration value. The types of the parameters specified must match the types of the parameters of the signal. Binding sets are connected to widgets in the same manner as styles, with one addition. A priority can be specified for each pattern, and within each type of pattern, binding sets override other binding sets first by priority, and only then by order of specification. (Later overrides earlier). The priorities that can be specified are (highest to lowest):

- highest
- rc
- theme
- application
- gtk
- lowest

`rc` is the default for bindings read from an RC file, `theme` is the default for bindings read from theme RC files, `application` should be used for bindings an application sets up, and `gtk` is used for bindings that `GTK+` creates internally.

Methods

gtk.RcStyle.copy

```
def copy()
```

Returns :	a new gtk.RcStyle that is a copy of the rcstyle

The copy() method returns a new gtk.RcStyle that is a copy of the RC style. This method will correctly copy an RC style that is a member of a class derived from gtk.RcStyle.

Functions

gtk.rc_add_default_file

```
def gtk.rc_add_default_file(filename)
```

filename :	the name of a file containing resource data

The gtk.rc_add_default_file() function adds the file specified by *filename* to the list of files to be parsed for resource data.

gtk.rc_set_default_files

```
def gtk.rc_set_default_files(filenames)
```

filenames :	a list of filenames

The gtk.rc_set_default_files() function sets the list of files (specified by *filenames*) that will be parsed for resource information.

gtk.rc_get_default_files

```
def gtk.rc_get_default_files()
```

Returns :	the current list of resource files

The gtk.rc_get_default_files() function returns a list of filenames (as set by the gtk.rc_set_default_files() function) that will be parsed for resource data.

gtk.rc_get_style_by_paths

```
def gtk.rc_get_style_by_paths(settings, widget_path, class_path, type)
```

settings :	a gtk.Settings object
widget_path :	the widget path to use when looking up the style
class_path :	the class path to use when looking up the style
type :	a type that will be used along with parent types of this type when matching against class styles, or gobject.TYPE_NONE
Returns :	a gtk.Style created by matching with the supplied paths, or None if nothing matching was specified and the default style should be used.

The gtk.rc_get_style_by_paths() function returns a gtk.Style created from styles defined in a RC file by providing the raw components used in matching. This function may be useful when creating pseudo–widgets that should be themed like widgets but don't actually have corresponding `PyGTK` widgets. An example of this would be items inside a `GNOME` canvas widget.

gtk.rc_reparse_all_for_settings

```
def gtk.rc_reparse_all_for_settings(settings, force_load)
```

`settings` :	a `gtk.Settings` object
`force_load` :	if `TRUE` reparse the RC files even if they haven't changed
Returns :	`TRUE` if the files were reparsed

The `gtk.rc_reparse_all_for_settings`() function reparses the files associated with the `gtk.Settings` object specified by *`settings`* if any of the files have changed and *`force_load`* is `FALSE` and . If *`force_load`* is `TRUE` the files are always reparsed.

gtk.rc_reset_styles

```
def gtk.rc_reset_styles(settings)
```

`settings` :	a `gtk.Settings` object
Returns :	a `gtk.Style`

Note

This function is available in PyGTK 2.4 and above.

The `gtk.rc_reset_styles`() function returns a `gtk.Style`. This function computes the styles for all widgets that use the `gtk.Settings` object specified by *`settings`*. (There is one `gtk.Settings` object per `gtk.gdk.Screen`, see the `gtk.settings_get_for_screen()` function). It is useful when some global parameter has changed that affects the appearance of all widgets, because when a widget gets a new style, it will both redraw and recompute any cached information about its appearance. As an example, it is used when the default font size set by the operating system changes. Note that this function doesn't affect widgets that have a style set explicitly on them with the `gtk.Widget.set_style()` method.

gtk.rc_parse

```
def gtk.rc_parse(filename)
```

`filename` :	the name of a file to parse for resource data

The `gtk.rc_parse`() function parses the file specified by *`filename`* for resource data.

gtk.rc_parse_string

```
def gtk.rc_parse_string(rc_string)
```

`rc_string` :	a string to parse for resource data

The `gtk.rc_parse_string`() function parses the string specified by *`rc_string`* for resource data.

gtk.rc_reparse_all

```
def gtk.rc_reparse_all()
```

Returns :	`TRUE` if the files were reparsed.

The `gtk.rc_reparse_all()` function discards all style data and reparses all the RC files for resource data if any of them have changed.

gtk.rc_find_module_in_path

```
def gtk.rc_find_module_in_path(module_file)
```

`module_file` :	the name of a theme engine
Returns :	the filename of the theme engine or `None`

The `gtk.rc_find_module_in_path()` function searches for a theme engine named by *`module_file`*. This function is not useful for applications and should not be used.

gtk.rc_get_theme_dir

```
def gtk.rc_get_theme_dir()
```

Returns :	the name of the themes directory

The `gtk.rc_get_theme_dir()` function returns the name of the directory where themes should be installed.

gtk.rc_get_module_dir

```
def gtk.rc_get_module_dir()
```

Returns :	the theme engines directory name

The `gtk.rc_get_module_dir()` function returns the name of the directory where `PyGTK` searches for theme engines.

gtk.rc_get_im_module_path

```
def gtk.rc_get_im_module_path()
```

Returns :	the IM modules path

The `gtk.rc_get_im_module_path()` function returns the path where `PyGTK` searches for IM modules.

gtk.rc_get_im_module_file

```
def gtk.rc_get_im_module_file()
```

Returns :	the name of the IM modules file

The `gtk.rc_get_im_module_file()` function returns the name of the `PyGTK` IM modules file.

Prev	Up	Next
gtk.Range	Home	gtk.Requisition

gtk.Requisition

Prev **The gtk Class Reference** Next

gtk.Requisition

gtk.Requisition — an object containing information about the desired space requirements of a widget.

Synopsis

```
class gtk.Requisition(gobject.GBoxed):
    def copy()
    def free()
```

Attributes

"width"	Read−Write	the desired width of the widget
"height"	Read−Write	the desired height of the widget

Description

A `gtk.Requisition` holds the information about the desired space requirements (width and height) of a widget. A `gtk.Requisition` object has `width` and `height` attributes that can be read and written.

Note

There appears to be no way to create or use a `gtk.Requisition` in PyGTK other than as an argument in the handler for the `gtk.Widget` "size−request" signal.

Methods

gtk.Requisition.copy

```
def copy()
```

Returns :	a copy of the `gtk.Requisition`

The `copy()` method returns a copy of the `gtk.Requisition`.

gtk.Requisition.free

```
def free()
```

The `free()` method frees the resources allocated to the `gtk.Requisition`.

gtk.Ruler

gtk.Ruler — a base class for horizontal or vertical rulers

Synopsis

```
class gtk.Ruler(gtk.Widget):
    def set_metric(metric)
    def set_range(lower, upper, position, max_size)
    def draw_ticks()
    def draw_pos()
    def get_metric()
    def get_range()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Ruler
```

Properties

"lower"	Read−Write	the lower limit of the ruler
"upper"	Read−Write	the upper limit of the ruler
"position"	Read−Write	the position of the mark on the ruler
"max−size"	Read−Write	the maximum size of the ruler

Description

Note

This widget is considered too specialized or little−used for PyGTK and GTK+, and will in the future be moved to some other package. If your application needs this widget, feel free to use it, as the widget does work and is useful in some applications; it's just not of general interest. However, it will eventually move out of the PyGTK and GTK+ distribution.

The `gtk.Ruler` widget is a base class for horizontal and vertical rulers. Rulers are used to show the mouse pointer's location in a window. Within the ruler a small triangle indicates the location of the mouse relative to the horizontal or vertical ruler. See `gtk.HRuler` to learn about horizontal rulers. See `gtk.VRuler` to learn about vertical rulers.

Methods

gtk.Ruler.set_metric

```
    def set_metric(metric)
```

metric :	the measurement units

The `set_metric()` method sets the measurement units of the ruler to the value specified by *`metric`*. Available units are `gtk.PIXELS`, `gtk.INCHES`, or `gtk.CENTIMETERS`. The default unit of measurement is `gtk.PIXELS`.

gtk.Ruler.set_range

```
    def set_range(lower, upper, position, max_size)
```

`lower` :	the upper limit of the ruler
`upper` :	the lower limit of the ruler
`position` :	the position of the mark on the ruler
`max_size` :	the maximum size of the rule

The `set_range()` method sets the range of the ruler to the values specified by *`lower`*, *`upper`*, *`position`* and *`max_size`* (all values are floats).

gtk.Ruler.draw_ticks

```
    def draw_ticks()
```

The `draw_ticks()` method is overridden by the `gtk.Ruler` subclasses (`gtk.HRuler` and `gtk.VRuler`) to draw the tick marks on the ruler. This method is not used by applications.

gtk.Ruler.draw_pos

```
    def draw_pos()
```

The `draw_pos()` method is overridden by the `gtk.Ruler` subclasses (`gtk.HRuler` and `gtk.VRuler`) to draw the position mark on the ruler. This method is not used by applications.

gtk.Ruler.get_metric

```
    def get_metric()
```

Returns :	the measurement units currently used for the ruler

The `get_metric()` method returns the units used for a `gtk.Ruler`. See the `set_metric()` method.

gtk.Ruler.get_range

```
    def get_range()
```

Returns>	a tuple containing: the lower limit of the ruler, the upper limit of the ruler, the current position of the mark on the ruler and the maximum size of the ruler used when calculating the space to leave for the text.

The `get_range()` method returns a tuple containing the values indicating the range and current position of a `gtk.Ruler`. See the `set_range()` method.

gtk.Scale

gtk.Scale — a base class for the scale widgets.

Synopsis

```
class gtk.Scale(gtk.Range):
    def set_digits(digits)
    def get_digits()
    def set_draw_value(draw_value)
    def get_draw_value()
    def set_value_pos(pos)
    def get_value_pos()
    def get_layout()
    def get_layout_offsets()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Range
        +-- gtk.Scale
```

Properties

"digits"	Read–Write	The number of decimal places that are displayed in the value
"draw–value"	Read–Write	If `TRUE` the current value is displayed as a string next to the slider
"value–pos"	Read–Write	The position in which the current value is displayed: `gtk.POS_LEFT`, `gtk.POS_RIGHT`, `gtk.POS_TOP` or `gtk.POS_BOTTOM`

Style Properties

"slider–length"	Read	The length of scale's slider
"value–spacing"	Read	The space between value text and the slider or trough area

Signal Prototypes

"format–value"	`def callback(scale, user_param1, ...)`

Description

The `gtk.Scale` widget is an abstract base class, used only for deriving the subclasses `gtk.HScale` and `gtk.VScale`. A `gtk.Scale` is a slider control used to select a numeric value. Since `gtk.Scale` is a subclass of `gtk.Range`, see the `gtk.Range` methods for additional methods. To set the value of a scale, you would normally use `set_value()`. To detect changes to the value, you would normally use the "value_changed" signal.

Methods

gtk.Scale.set_digits

```
    def set_digits(digits)
```

`digits` :	the number of decimal places to display

The `set_digits()` method sets the "digits" property to the value specified by *`digits`*. The value of *`digits`* specifies the number of decimal places that are displayed in the value. The value of the adjustment is also rounded off to this number of digits, so the retrieved value matches the value the user sees.

gtk.Scale.get_digits

```
    def get_digits()
```

Returns :	the number of decimal places that are displayed.

The `get_digits()` method returns the value of the "digits" property that indicates the number of decimal places that are displayed in the value.

gtk.Scale.set_draw_value

```
    def set_draw_value(draw_value)
```

`draw_value` :	If `TRUE` draw the current value next to the slider

The `set_draw_value()` method sets the "draw−value" property to the value specified by *`draw_value`*. If *`draw_value`* is *`TRUE`* the current value is displayed next to the slider.

gtk.Scale.get_draw_value

```
    def get_draw_value()
```

Returns :	`TRUE` if the current value is to be drawn next to the slider

The `get_draw_value()` method returns the value of the "draw−value" property. If "draw−value" is `TRUE` the current scale value is drawn next to the slider.

gtk.Scale.set_value_pos

```
    def set_value_pos(pos)
```

`pos` :	the position where the current value is displayed.

The `set_value_pos()` method sets the value of the "value−pos" property to the value specified by *`pos`*. The value of *`pos`* must be one of: `gtk.POS_LEFT`, `gtk.POS_RIGHT`, `gtk.POS_TOP` or `gtk.POS_BOTTOM`.

gtk.Scale.get_value_pos

```
    def get_value_pos()
```

Returns :	the position where the current value is displayed.

The `get_value_pos()` method returns the value of the "value−pos" property. See the `set_value_pos()` method.

gtk.Scale.get_layout

```
    def get_layout()
```

Returns :	the `pango.Layout` for this scale or `None` if the draw_value property is `FALSE`.

Note

This method is available in PyGTK 2.4 and above.

The `get_layout()` method returns the `pango.Layout` used to display the scale.

gtk.Scale.get_layout_offsets

```
    def get_layout_offsets()
```

Returns :	a 2–tuple containing the coordinates where the scale will draw the `pango.Layout` representing the text in the scale

Note

This method is available in PyGTK 2.4 and above.

The `get_layout_offsets()` method returns a 2–tuple containing the coordinates where the scale will draw the `pango.Layout` representing the text in the scale. Remember when using the `pango.Layout` function you need to convert to and from pixels using the `pango.PIXELS()` function or `pango.SCALE`. If the "draw–value" property is `FALSE`, the return values are undefined.

Signals

The "format–value" gtk.Scale Signal

```
    def callback(scale, value, user_param1, ...)
```

`scale` :	the scale that received the signal
`value` :	the value to be formatted
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)
Returns :	a string representing *`value`* for display

The "format–value" signal is emitted when the scale is being redrawn with a value displayed. The "format–value" signal handler should return a formatted string containing *`value`*.

Prev Up Next
gtk.Ruler Home gtk.Scrollbar

gtk.Scrollbar
Prev **The gtk Class Reference** Next

gtk.Scrollbar

gtk.Scrollbar a base class for scrollbar widgets.

Synopsis

```
class gtk.Scrollbar(gtk.Range):
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Range
        +-- gtk.Scrollbar
```

Style Properties

"fixed–slider–length"	Read	If `TRUE` don't change the slider size, just lock it to the minimum length
"has–backward–stepper"	Read	If `TRUE` display the standard backward arrow button
"has–forward–stepper"	Read	If `TRUE` display the standard forward arrow button
"has–secondary–backward–stepper"	Read	If `TRUE` display a second backward arrow button on the opposite end of the scrollbar
"has–secondary–forward–stepper"	Read	If `TRUE` display a secondary forward arrow button on the opposite end of the scrollbar
min–slider–length""	Read	The minimum length of scrollbar slider

Description

The `gtk.Scrollbar` widget is an abstract base class for `gtk.HScrollbar` and `gtk.VScrollbar`. The position of the thumb in a scrollbar is controlled by the scroll adjustments. The `gtk.Scrollbar` uses the attributes in an adjustment (see `gtk.Adjustment`) as follows:

- the `adjustment.lower` attribute is the minimum value of the scroll region
- the `adjustment.upper` attribute is the maximum value of the scroll region
- the `adjustment.value` attribute represents the position of the scrollbar, which must be between `adjustment.lower` and `adjustment.upper` – `adjustment.page_size`
- the `adjustment.page_size` attribute represents the size of the visible scrollable area
- the `adjustment.step_increment` attribute is the distance to scroll when the small stepper arrows are clicked
- the `adjustment.page_increment` attribute is the distance to scroll when the **Page Up** or **Page Down** keys are pressed

gtk.ScrolledWindow

gtk.ScrolledWindow adds scrollbars to its child widget.

Synopsis

```
class gtk.ScrolledWindow(gtk.Bin):
    gtk.ScrolledWindow(hadjustment=None, vadjustment=None)
    def set_hadjustment(hadjustment)
    def set_vadjustment(hadjustment)
    def get_hadjustment()
    def get_vadjustment()
    def set_policy(hscrollbar_policy, vscrollbar_policy)
    def get_policy(hscrollbar_policy, vscrollbar_policy)
    def set_placement(window_placement)
    def get_placement()
    def set_shadow_type(type)
    def get_shadow_type()
    def add_with_viewport(child)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.ScrolledWindow
```

Properties

"hadjustment"	Read–Write–Construct	The `gtk.Adjustment` for the horizontal position.
hscrollbar–policy""	Read–Write	The horizontal scrollbar display policy; one of: `gtk.POLICY_ALWAYS`, `gtk.POLICY_AUTOMATIC` or `gtk.POLICY_NEVER`.
"shadow–type"	Read–Write	The style of bevel around the contents; one of: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`.
"vadjustment"	Read–Write–Construct	The `gtk.Adjustment` for the vertical position.
"vscrollbar–policy"	Read–Write	The vertical scrollbar display policy; one of: `gtk.POLICY_ALWAYS`, `gtk.POLICY_AUTOMATIC` or `gtk.POLICY_NEVER`.
"window–placement"	Read–Write	Where the contents are located with respect to the scrollbars; one of: `gtk.CORNER_TOP_LEFT`, `gtk.CORNER_BOTTOM_LEFT`, `gtk.CORNER_TOP_RIGHT`, `gtk.CORNER_BOTTOM_RIGHT`.

Style Properties

"scrollbar–spacing"	Read	The number of pixels between the scrollbars and the scrolled window. Available in `GTK+` 2.2 and above.

Signal Prototypes

"move–focus–out"	`def callback(`*`scrolledwindow, direction, user_param1, ...`*`)`

"scroll–child"	def callback(*scrolledwindow*, *scrolltype*, *horizontal*, *user_param1*, *...*)

Description

A `gtk.ScrolledWindow` is a subclass of `gtk.Bin` that adds scrollbars to a single child widget and optionally draws a beveled frame around the child widget. The scrolled window can work in two ways:

- Some widgets have native scrolling support using "slots" to hold `gtk.Adjustment` objects. Widgets with native scroll support include `gtk.TreeView`, `gtk.TextView`, and `gtk.Layout`.
- Widgets that lack native scrolling support use the `gtk.Viewport` widget that acts as an adapter class, implementing scrollability for child widgets that lack their own scrolling capabilities. Use `gtk.Viewport` to scroll child widgets such as `gtk.Table`, `gtk.Box`, and so on.

If a widget has native scrolling abilities, it can be added to the `gtk.ScrolledWindow` with the `gtk.Container.add()` method. If a widget does not, you must first add the widget to a `gtk.Viewport`, then add the `gtk.Viewport` to the scrolled window. The convenience method `add_with_viewport()` does exactly this, so you can ignore the presence of the viewport.

The position of the scrollbars is controlled by the scroll adjustments. The `gtk.ScrolledWindow` uses the attributes in an adjustment (see `gtk.Adjustment`) as follows:

- the `adjustment.lower` attribute is the minimum value of the scroll region
- the `adjustment.upper` attribute is the maximum value of the scroll region
- the `adjustment.value` attribute represents the position of the scrollbar, which must be between `adjustment.lower` and `adjustment.upper – adjustment.page_size`
- the `adjustment.page_size` attribute represents the size of the visible scrollable area
- the `adjustment.step_increment` attribute is the distance to scroll when the small stepper arrows are clicked
- the `adjustment.page_increment` attribute is the distance to scroll when the **Page Up** or **Page Down** keys are pressed

If a `gtk.ScrolledWindow` doesn't behave quite as you would like, or doesn't have exactly the right layout, it's very possible to set up your own scrolling with `gtk.Scrollbar` and for example a `gtk.Table`.

Constructor

```
    gtk.ScrolledWindow(hadjustment=None, vadjustment=None)
```

hadjustment :	the horizontal `gtk.Adjustment` or None
vadjustment :	a vertical `gtk.Adjustment` or None
Returns :	a new `gtk.ScrolledWindow` widget

Creates a new scrolled window with the horizontal and vertical `gtk.Adjustment` specified by *hadjustment* and *vadjustment* respectively. These will be shared with the scrollbars and the child widget to keep the bars in sync with the child. If *hadjustment* and *vadjustment* are `None` or not specified the scrolled window will create them for you.

Methods

gtk.ScrolledWindow.set_hadjustment

```
    def set_hadjustment(hadjustment)
```

hadjustment :	the horizontal gtk.Adjustment

The `set_hadjustment()` method sets the horizontal adjustment (and the "hadjustment" property) to the value of *hadjustment*. *hadjustment* must be a gtk.Adjustment.

gtk.ScrolledWindow.set_vadjustment

```
    def set_vadjustment(hadjustment)
```

hadjustment :	the vertical gtk.Adjustment

The `set_vadjustment()` method sets the vertical adjustment (and the "vadjustment" property) to the value of *vadjustment*. *vadjustment* must be a gtk.Adjustment.

gtk.ScrolledWindow.get_hadjustment

```
    def get_hadjustment()
```

Returns :	the horizontal gtk.Adjustment

The `get_hadjustment()` method returns the value of the "hadjustment" property which is a reference to the horizontal adjustment.

gtk.ScrolledWindow.get_vadjustment

```
    def get_vadjustment()
```

Returns :	the vertical gtk.Adjustment

The `get_vadjustment()` method returns the value of the "vadjustment" property which is a reference to the vertical adjustment.

gtk.ScrolledWindow.set_policy

```
    def set_policy(hscrollbar_policy, vscrollbar_policy)
```

hscrollbar_policy :	the policy for the horizontal scrollbar
vscrollbar_policy :	the policy for the vertical scrollbar

The `set_policy()` method sets the "hscrollbar_policy" and "vscrollbar_policy" properties to the value of *hscrollbar_policy* and *vscrollbar_policy* respectively. The policy determines when the scrollbar should be displayed. The policy value is one of:

gtk.POLICY_ALWAYS	the scrollbar is always present
gtk.POLICY_AUTOMATIC	the scrollbar is present only if needed i.e. the contents are larget than the window
gtk.POLICY_NEVER	the scrollbar is never present

gtk.ScrolledWindow.get_policy

```
    def get_policy()
```

Returns> :	a tuple containing the horizontal and vertical scrollbar policies

The `get_policy()` method returns a tuple containing the horizontal and vertical scrollbar policies. See the `set_policy()` method for more detail.

gtk.ScrolledWindow.set_placement

```
    def set_placement(window_placement)
```

`window_placement` :	the placement of the contents with respect to the scrollbars

The `set_placement()` method sets the "window−placement" property to the value specified by *`window_placement`*. The window placement determines the location of the child widget with respect to the scrollbars. *`window_placement`* must be one of:

gtk.CORNER_TOP_LEFT	Place the scrollbars on the right and bottom of the widget (default behavior).
gtk.CORNER_BOTTOM_LEFT	Place the scrollbars on the top and right of the widget.
gtk.CORNER_TOP_RIGHT	Place the scrollbars on the left and bottom of the widget.
gtk.CORNER_BOTTOM_RIGHT	Place the scrollbars on the top and left of the widget.

gtk.ScrolledWindow.get_placement

```
    def get_placement()
```

Returns :	the current placement value.

The `get_placement()` method returns the value of the "window−placement" property that determines the placement of the scrollbars with respect to the scrolled window. See the `set_placement()` method for more detail.

gtk.ScrolledWindow.set_shadow_type

```
    def set_shadow_type(type)
```

`type` :	the kind of bevel shadow to draw around the scrolled window contents

The `set_shadow_type()` method sets the value of the "shadow−type" property to the value of *`shadow_type`*. *`shadow_type`* determines the type of bevel shadow drawn around the contents of the scrolled window. The shadow type must be one of:

gtk.SHADOW_NONE	No outline.
gtk.SHADOW_IN	The outline is beveled inward.
gtk.SHADOW_OUT	The outline is beveled outward.
gtk.SHADOW_ETCHED_IN	The outline is an inward etched bevel.
gtk.SHADOW_ETCHED_OUT	The outline is an outward etched bevel.

gtk.ScrolledWindow.get_shadow_type

```
    def get_shadow_type()
```

Returns :	the current shadow type

The `get_shadow_type()` method returns the value of the "shadow−type" property that determines the shadow type of the scrolled window. See the `set_shadow_type()` method for more detail.

gtk.ScrolledWindow.add_with_viewport

```
    def add_with_viewport(child)
```

child :	the widget to be scrolled

The `add_with_viewport`() method is used to add a widget (specified by *`child`*) without native scrolling capabilities to the scrolled window. This is a convenience function that is equivalent to adding *`child`* to a `gtk.Viewport`, then adding the viewport to the scrolled window. If a child has native scrolling (e.g. `gtk.TextView`, `gtk.TreeView`, `gtk.Layout`), use `gtk.Container.add()` instead of this method.

The viewport scrolls the child by moving its `gtk.gdk.Window`, and takes the size of the child to be the size of its toplevel `gtk.gdk.Window`. This will be wrong for most widgets that support native scrolling. For example, if you add a widget such as `gtk.TreeView` with a viewport, the whole widget will scroll, including the column headings.

Signals

The "move−focus−out" gtk.ScrolledWindow Signal

```
    def callback(scrolledwindow, direction, user_param1, ...)
```

scrolledwindow :	the scrolledwindow that received the signal
direction :	the direction that the focus is moving either `gtk.DIR_TAB_FORWARD` or `gtk.DIR_TAB_BACKWARD`.
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "move−focus−out" signal is emitted when the user presses **Control+Tab** or **Control+Shift+Tab** to move the focus out of the scrolled window. The *`direction`* is either `gtk.DIR_TAB_FORWARD` or `gtk.DIR_TAB_BACKWARD`

The "scroll−child" gtk.ScrolledWindow Signal

```
    def callback(scrolledwindow, scrolltype, horizontal, user_param1, ...)
```

scrolledwindow :	the scrolledwindow that received the signal
scrolltype :	the scroll type; one of: `gtk.SCROLL_STEP_BACKWARD`, `gtk.SCROLL_STEP_FORWARD`, `gtk.SCROLL_PAGE_BACKWARD`, `gtk.SCROLL_PAGE_FORWARD`, `gtk.SCROLL_PAGE_UP`, `gtk.SCROLL_PAGE_DOWN`, `gtk.SCROLL_START` or `gtk.SCROLL_END`.
horizontal :	if TRUE scroll in the horizontal direction
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "scroll−child" signal is emitted when the child widget is being scrolled by a keyboard action. The default key bindings with resulting *`scrolltype`* and *`horizontal`* arguments are:

Control+Left Arrow	`gtk.SCROLL_STEP_BACKWARD` – horizontal
Control+Right Arrow	`gtk.SCROLL_STEP_FORWARD` – horizontal
Control+Up Arrow	`gtk.SCROLL_STEP_BACKWARD` – vertical
Control+Down Arrow	`gtk.SCROLL_STEP_FORWARD` – vertical
Control+Page Up	`gtk.SCROLL_PAGE_BACKWARD` – horizontal

Control+Page Down	`gtk.SCROLL_PAGE_FORWARD` – horizontal
Page Up	`gtk.SCROLL_PAGE_BACKWARD` – vertical
Page Down	`gtk.SCROLL_PAGE_FORWARD` – vertical
Control+Home	`gtk.SCROLL_START` – horizontal
Control+End	`gtk.SCROLL_END` – horizontal
Home	`gtk.SCROLL_START` – vertical
End	`gtk.SCROLL_END` – vertical

gtk.SelectionData

gtk.SelectionData — an object that stores information about a selection

Synopsis

```
class gtk.SelectionData(gobject.GBoxed):
    def set(type, format, data)
    def set_text(str, len)
    def get_text()
    def get_targets()
    def targets_include_text()
    def tree_set_row_drag_data(tree_model, path)
    def tree_get_row_drag_data()
    def set_pixbuf(pixbuf)
    def get_pixbuf()
    def set_uris(uris)
    def get_uris()
    def targets_include_image(writable)
Functions

    def gtk.selection_owner_set_for_display(display, widget, selection, time=0)
    def gtk.target_list_add_image_targets(list=None, info=0, writable=FALSE)
    def gtk.target_list_add_text_targets(list=None, info=0)
    def gtk.target_list_add_uri_targets(list=None, info=0)
```

Attributes

"selection"	Read	A `gtk.gdk.Atom` indicating the selection type (e.g. "PRIMARY").
"target"	Read	A `gtk.gdk.Atom` indicating the selection target type (e.g. "TARGETS").
"type"	Read	A `gtk.gdk.Atom` indicating the selection data type (e.g. "STRING").
"format"	Read	The unit length of the data in bits (e.g. 8 for a string or 32 of an integer).
"data"	Read	The data as a string.

Description

A gtk.SelectionData object is used to store information about a chunk of data associated with a selection. In PyGTK the selection data is always a string so the application will have to provide functions to convert the data to and from a string to support data types other than strings and targets. The string and targets types are directly supported using the set_text(), get_text() and get_targets() methods.

Methods

gtk.SelectionData.set

```
    def set(type, format, data)
```

type :	a gtk.gdk.Atom or string that specifies a gtk.gdk.Atom
format :	the number of bits in a unit
data :	a string containing the data

The `set`() method sets the data for a selection in the gtk.SelectionData object. *data* is a string containing the data to be set; *format* is the number of bits in a unit of the data (e.g. integer data has a format of 32 on most systems; string data format is 8); and, *type* is a gtk.gdk.Atom or a string that specifies a gtk.gdk.Atom.

gtk.SelectionData.set_text

```
    def set_text(str, len)
```

str :	a string
len :	the length of *str*, or −1 if *str* for the full length.
Returns :	`TRUE`, if the selection was successfully set; otherwise, `FALSE`.

The `set_text`() method sets the contents of the selection from the string specified by *str*. The string is converted to the form specified by the `selection_data.target` attribute. This method returns TRUE if the selection data was successfully set.

gtk.SelectionData.get_text

```
    def get_text()
```

Returns :	a string containing the converted text, or `None`.

The `get_text`() method returns the contents of the selection data as a string.

gtk.SelectionData.get_targets

```
    def get_targets()
```

Returns :	a tuple containing a list of targets (gtk.gdk.Atoms) or `None` if no valid targets are available.

The `get_targets`() method returns a tuple containing a list of valid targets for the selection as a list of gtk.gdk.Atoms or None if there are no valid targets.

gtk.SelectionData.targets_include_text

```
    def targets_include_text()
```

Returns :	`TRUE` if the selection data holds a list of targets, and a suitable target for text is included.

The `targets_include_text()` method returns `TRUE` if any of the selection data targets can be used to provide text.

gtk.SelectionData.tree_set_row_drag_data

```
    def tree_set_row_drag_data(tree_model, path)
```

`tree_model` :	a `gtk.TreeModel`
`path` :	a row in *`tree_model`*
Returns :	`TRUE` if the `gtk.SelectionData` had the proper target type to allow us to set a tree row

The `tree_set_row_drag_data()` method sets the selection data of target type `GTK_TREE_MODEL_ROW` for the row (specified by *`path`*) in the `gtk.TreeModel` (specified by tree_model). Normally used in a "drag–data–get" signal handler.

gtk.SelectionData.tree_get_row_drag_data

```
    def tree_get_row_drag_data()
```

Returns :	a tuple containing a `gtk.TreeModel` and one of its rows.

The `tree_get_row_drag_data()` method returns a tuple containing a `gtk.TreeModel` and a row from that `gtk.TreeModel` from selection data of target type `GTK_TREE_MODEL_ROW`. Normally called from a "drag–data–received" signal handler. This method can only be used if the selection data originates from the same process that's calling this method, because a pointer to the tree model is being passed around. In the "drag–data–received" signal handler, you can assume that selection data of type `"GTK_TREE_MODEL_ROW"` is from the current process.

gtk.SelectionData.set_pixbuf

```
    def set_pixbuf(pixbuf)
```

`pixbuf` :	a `gtk.gdk.Pixbuf`
Returns :	`TRUE`, if the selection was successfully set; otherwise, `FALSE`.

Note

This method is available in PyGTK 2.6 and above.

The `set_pixbuf()` method sets the contents of the selection from the `gtk.gdk.Pixbuf` specified by *`pixbuf`*. This method returns `TRUE` if the selection data was successfully set.

gtk.SelectionData.get_pixbuf

```
    def get_pixbuf()
```

Returns :	if the selection data contained a recognized image type and it could be converted to a `gtk.gdk.Pixbuf`, a newly allocated pixbuf is returned, or `None`.

Note

This method is available in PyGTK 2.6 and above.

The `get_pixbuf()` method returns the contents of the selection data as a `gtk.gdk.Pixbuf` if possible.

gtk.SelectionData.set_uris

```
    def set_uris(uris)
```

`uris` :	a list of strings holding URIs
Returns :	`TRUE`, if the selection was successfully set; otherwise, `FALSE`.

Note

This method is available in PyGTK 2.6 and above.

The `set_uris()` method sets the contents of the selection from the list of URIs specified by *`uris`*. This method returns `TRUE` if the selection data was successfully set.

gtk.SelectionData.get_uris

```
    def get_uris()
```

Returns :	a list of URIs, or `None`.

Note

This method is available in PyGTK 2.6 and above.

The `get_uris()` method returns the contents of the selection data as a list or URIs.

gtk.SelectionData.targets_include_image

```
    def targets_include_image(writable)
```

`writable` :	If `TRUE` only accept targets that GTK+ can convert a `gtk.gdk.Pixbuf` to.
Returns :	`TRUE`, if the selection has a list of targets that includes an image target.

Note

This method is available in PyGTK 2.6 and above.

The `targets_include_image()` method returns `TRUE` if the selection targets include an image target.

Functions

gtk.selection_owner_set_for_display

```
    def gtk.selection_owner_set_for_display(display, widget, selection, time=0)
```

`display` :	the `gtk.gdk.Display` where the selection is set
`widget` :	the new selection owner (a `gtk.Widget`), or `None`.

`selection` :	a `gtk.gdk.Atom` or string representing a selection target
`time` :	the timestamp used to claim the selection
Returns :	`TRUE` if the operation succeeded

The `gtk.selection_owner_set_for_display()` function claims ownership of the selection specified by *`selection`* for the widget specified by *`widget`* on the `gtk.gdk.Display` specified by *`display`*. If *`widget`* is `None`, the ownership of the selection is released.

gtk.target_list_add_image_targets

```
    def gtk.target_list_add_image_targets(list=None, info=0, writable=FALSE)
```

`list` :	A sequence of target entry tuples or `None`
`info` :	an application specified ID that will be passed back to the application
`writable` :	If `TRUE`, only add targets for image formats that a pixbuf can be converted to.
Returns :	a new list concatenating *`list`* and the built–in image targets supported by `gtk.SelectionData`.

Note

This function is available in PyGTK 2.6 and above.

The `gtk.target_list_add_image_targets()` function adds the image target tuples supported by `gtk.SelectionData` to the list of target entry tuples specified by *`list`*. `info` is used as the info field of the target entry tuples.

gtk.target_list_add_text_targets

```
    def gtk.target_list_add_text_targets(list=None, info=0)
```

`list` :	A sequence of target entry tuples or `None`
`info` :	an application specified ID that will be passed back to the application
Returns :	a new list concatenating *`list`* and the built–in text targets supported by `gtk.SelectionData`.

Note

This function is available in PyGTK 2.6 and above.

The `gtk.target_list_add_text_targets()` function adds the text target tuples supported by `gtk.SelectionData` to the list of target entry tuples specified by *`list`*. `info` is used as the info field of the target entry tuples.

gtk.target_list_add_uri_targets

```
    def gtk.target_list_add_uri_targets(list=None, info=0)
```

`list` :	A sequence of target entry tuples or `None`
`info` :	an application specified ID that will be passed back to the application
Returns :	a new list concatenating *`list`* and the built–in URI targets supported by `gtk.SelectionData`.

Note

This function is available in PyGTK 2.6 and above.

The `gtk.target_list_add_uri_targets()` function adds the URI target tuples supported by `gtk.SelectionData` to the list of target entry tuple specified by *`list`*. `info` is used as the info field of the target entry tuples.

Prev	Up	Next
gtk.ScrolledWindow	Home	gtk.Separator

gtk.Separator

Prev	The gtk Class Reference	Next

gtk.Separator

gtk.Separator — a base class for visual separator widgets.

Synopsis

```
class gtk.Separator(gtk.Widget):
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Separator
```

Description

The `gtk.Separator` widget is an abstract base class, used only for deriving the subclasses `gtk.HSeparator` and `gtk.VSeparator`. A separator is a visual delimiter between widgets.

Prev	Up	Next
gtk.SelectionData	Home	gtk.SeparatorMenuItem

gtk.SeparatorMenuItem

Prev	The gtk Class Reference	Next

gtk.SeparatorMenuItem

gtk.SeparatorMenuItem — a separator used in menus.

Synopsis

```
class gtk.SeparatorMenuItem(gtk.MenuItem):
    gtk.SeparatorMenuItem()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Item
            +-- gtk.MenuItem
              +-- gtk.SeparatorMenuItem
```

Description

The `gtk.SeparatorMenuItem` is a separator used to group items within a menu. It displays a horizontal line with a shadow to make it appear sunken into the interface.

Constructor

```
   gtk.SeparatorMenuItem()
```

Returns :	a new `gtk.SeparatorMenuItem` widget

Creates a new `gtk.SeparatorMenuItem` widget.

gtk.SeparatorToolItem

gtk.SeparatorToolItem a toolbar item that separates groups of other toolbar items (new in PyGTK 2.4)

Synopsis

```
class gtk.SeparatorToolItem(gtk.ToolItem):
    gtk.SeparatorToolItem()
    def get_draw()
    def set_draw(draw)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.ToolItem
            +-- gtk.SeparatorToolItem
```

Properties

"draw"	Read−Write	If `TRUE`, the separator is drawn. Available in GTK+ 2.4.

Description

Note

This widget is available in PyGTK 2.4 and above.

A `gtk.SeparatorToolItem` is a `gtk.ToolItem` that separates groups of other `gtk.ToolItem` objects. Depending on the theme, a `gtk.SeparatorToolItem` will often look like a vertical line on horizontally docked toolbars.

If the `gtk.SeparatorToolItem` is set to expand (using the `gtk.ToolItem.set_expand()` method) and the "draw" property is `FALSE`, a `gtk.SeparatorToolItem` will act as a "spring" that forces other items to the ends of the toolbar.

Constructor

```
gtk.SeparatorToolItem()
```

Returns :	the new `gtk.SeparatorToolItem` Since: 2.4

Note

This constructor is available in PyGTK 2.4 and above.

Create a new `gtk.SeparatorToolItem`

Methods

gtk.SeparatorToolItem.get_draw

```
def get_draw()
```

Returns :	`TRUE` if *`separator_tool_item`* is drawn as a line, or just blank.

Note

This method is available in PyGTK 2.4 and above.

The `get_draw`() method returns `TRUE` if the separator tool item is drawn as a line or `FALSE`, if it's just blank. See the `set_draw()` method.

gtk.SeparatorToolItem.set_draw

```
def set_draw(draw)
```

`draw` :	if `TRUE` the `gtk.SeparatorToolItem` is drawn as a vertical line

Note

This method is available in PyGTK 2.4 and above.

The `set_draw()` method sets the "draw" property to the value of *`draw`*. If *`draw`* is `TRUE` the `gtk.SeparatorToolItem` is drawn as a vertical line; if `FALSE`, just blank. Setting the "draw" property to `FALSE` along with passing TRUE to the `gtk.ToolItem.set_expand()` is useful to create an item that forces following items to the end of the toolbar.

gtk.Settings

gtk.Settings — an object that contains the global settings for the widgets on a `gtk.gdk.Screen`

Synopsis

```
class gtk.Settings(gobject.GObject):
    def set_string_property(name, v_string, origin)
    def set_long_property(name, v_long, origin)
    def set_double_property(name, v_double, origin)
Functions

    def gtk.settings_get_default()
    def gtk.settings_get_for_screen(screen)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Settings
```

Properties

"gtk–alternative–button–order"	Read–Write	If `TRUE`, buttons in dialogs should use the alternative button order. Available in GTK+ 2.6 and above. Default value: `FALSE`
"gtk–button–images"	Read–Write	If `TRUE`, stock icons should be shown in buttons. Available in GTK+ 2.4 and above. Default value: `TRUE`
"gtk–can–change–accels"	Read–Write	If `TRUE`, the menu accelerators can be changed by pressing a key over the menu item. Default value: `FALSE`
"gtk–color–palette"	Read–Write	The palette to use in the color selector. Default value: "black:white:gray50:red:purple:blue:light blue:green:yellow:orange:lavender:brown:goldenrod4:dodger blue:pink:light green:gray10:gray30:gray75:gray90"
"gtk–cursor–blink"	Read–Write	If `TRUE`, the cursor should blink. Default value: `TRUE`
"gtk–cursor–blink–time"	Read–Write	The length of the cursor blink cycle, in milliseconds. Allowed values: >= 100. Default value: 1200

"gtk–dnd–drag–threshold"	Read–Write	The number of pixels the cursor can move before dragging starts. Allowed values: >= 1. Default value: 8
"gtk–double–click–distance"	Read–Write	The maximum distance allowed between two clicks for them to be considered a double click (in pixels). Available in GTK+ 2.4 and above. Allowed values: >= 0. Default value: 5
"gtk–double–click–time"	Read–Write	The maximum time allowed between two clicks for them to be considered a double click (in milliseconds). Allowed values: >= 0. Default value: 250
"gtk–entry–select–on–focus"	Read–Write	If `TRUE`, select the contents of an entry when it is focused. Default value: `TRUE`
"gtk–font–name"	Read–Write	The name of the default font to use. Default value: "Sans 10"
"gtk–icon–sizes"	Read–Write	The list of icon sizes (gtk–menu=16,16;gtk–button=20,20...). Default value: `None`
"gtk–icon–theme–name"	Read–Write	The name of the icon theme to use. Available in GTK+ 2.4 and above. Default value: "hicolor"
"gtk–key–theme–name"	Read–Write	The name of the key theme RC file to load. Default value: `None`
"gtk–menu–bar–accel"	Read–Write	The keybinding to activate the menu bar. Default value: "F10"
"gtk–menu–bar–popup–delay"	Read–Write	The delay before the submenus of a menu bar appear. Allowed values: >= 0. Default value: 0 Available in GTK+ 2.2 and above.
"gtk–menu–images"	Read–Write	If `TRUE` images should be shown in menus. Default value: `TRUE`. Available in GTK+ 2.4 and above.
"gtk–menu–popdown–delay"	Read–Write	The time before hiding a submenu when the pointer is moving toward the submenu. Allowed values: >= 0. Default value: 1000. Available in GTK+ 2.2 and above.
"gtk–menu–popup–delay"	Read–Write	Minimum time the pointer must stay over a menu item before the submenu appears. Allowed values: >= 0. Default value: 225. Available in GTK+ 2.2 and above.
"gtk–modules"	Read–Write	The list of currently active GTK modules. Default value: `None`. Available in GTK+ 2.6 and above.
"gtk–split–cursor"	Read–Write	If `TRUE`, two cursors should be displayed for mixed left–to–right and right–to–left text. Default value: `TRUE`
"gtk–theme–name"	Read–Write	The name of the theme RC file to load. Default value: "Default"
"gtk–toolbar–icon–size"	Read–Write	the toolbar icon size – one of: `gtk.ICON_SIZE_MENU`, `gtk.ICON_SIZE_SMALL_TOOLBAR`, `gtk.ICON_SIZE_LARGE_TOOLBAR`, `gtk.ICON_SIZE_BUTTON`, `gtk.ICON_SIZE_DND` or `gtk.ICON_SIZE_DIALOG`. Default value: `gtk.ICON_SIZE_LARGE_TOOLBAR`
"gtk–toolbar–style"	Read–Write	The toolbar display style – one of: `gtk.TOOLBAR_ICONS`, `gtk.TOOLBAR_TEXT`, `gtk.TOOLBAR_BOTH` or `gtk.TOOLBAR_BOTH_HORIZ`. Default value: `gtk.TOOLBAR_BOTH`
"gtk–xft–antialias"	Read–Write	Whether to antialias Xft fonts; 0=no, 1=yes, –1=default. Default value: –1. Available in GTK+ 2.4 and above.
"gtk–xft–dpi"	Read–Write	Resolution for Xft, in 1024 * dots/inch. –1 to use default value. Allowed values: [–1,1048576]. Default value: –1. Available in GTK+ 2.4 and above.

"gtk−xft−hinting"	Read−Write	Whether to hint Xft fonts; 0=no, 1=yes, −1=default. Default value: −1. Available in GTK+ 2.4 and above.
"gtk−xft−hintstyle"	Read−Write	What degree of hinting to use; none, slight, medium, or full. Default value: `None`. Available in GTK+ 2.4 and above.
"gtk−xft−rgba"	Read−Write	Type of subpixel antialiasing; none, rgb, bgr, vrgb, vbgr. Default value: `None`. Available in GTK+ 2.4 and above.

Description

The `gtk.Settings` object stores the values of the global settings associated with a `gtk.gdk.Screen`. The `gtk.Settings` object for the default `gtk.gdk.Screen` can be retrieved using the `gtk.settings_get_default()` function. The `gtk.Widget.get_settings()` method returns the `gtk.Settings` object of the `gtk.gdk.Screen` that the widget is displayed on.

Methods

gtk.Settings.set_string_property

```
    def set_string_property(name, v_string, origin)
```

`name` :	the name of the property to set
`v_string` :	the string value
`origin` :	the string value of the origin

The `set_string_property()` method sets the property named *`name`* to the string value specified by *`v_string`* at the string origin specified by *`origin`*.

gtk.Settings.set_long_property

```
    def set_long_property(name, v_long, origin)
```

`name` :	the name of the property to set
`v_long` :	the long value
`origin` :	the string value of the origin

The `set_long_property()` method sets the property named *`name`* to the long value specified by *`v_long`* at the string origin specified by *`origin`*.

gtk.Settings.set_double_property

```
    def set_double_property(name, v_double, origin)
```

`name` :	the name of the property to set
`v_double` :	the double value
`origin` :	the string value of the origin

The `set_double_property()` method sets the property named *`name`* to the double value specified by *`v_long`* at the string origin specified by *`origin`*.

Functions

gtk.settings_get_default

```
    def gtk.settings_get_default()
```

Returns :	the singleton `gtk.Settings` object

The `gtk.settings_get_default` function returns the singleton `gtk.Settings` object.

gtk.settings_get_for_screen

```
    def gtk.settings_get_for_screen(screen)
```

`screen` :	a `gtk.gdk.Screen` object
Returns :	a `gtk.Settings` object

Note

This function is available in PyGTK 2.2 and above.

The `gtk.settings_get_for_screen()` function returns the singleton `gtk.Settings` object for the `gtk.gdk.Screen` specified by *`screen`*.

Prev	Up	Next
gtk.SeparatorToolItem	Home	gtk.SizeGroup

gtk.SizeGroup

Prev **The gtk Class Reference** Next

gtk.SizeGroup

gtk.SizeGroup — an object that groups widgets so they request the same size

Synopsis

```
class gtk.SizeGroup(gobject.GObject):
    gtk.SizeGroup(mode)
    def set_mode(mode)
    def get_mode()
    def add_widget(widget)
    def remove_widget(widget)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.SizeGroup
```

Properties

"mode"	Read–Write	The directions in which the size group effects the requested sizes of its component widgets – one of: `gtk.SIZE_GROUP_NONE`, `gtk.SIZE_GROUP_HORIZONTAL`,

gtk.SIZE_GROUP_VERTICAL, gtk.SIZE_GROUP_BOTH.

Description

gtk.SizeGroup provides a mechanism for grouping a number of widgets together so they all request the same amount of space. This is typically useful when you want a column of widgets to have the same size, but you can't use a gtk.Table widget. The size requested for each widget in a gtk.SizeGroup is the maximum of the sizes that would have been requested for each widget in the size group if they were not in the size group. The mode of the size group (see set_mode()) determines whether this applies to the horizontal size, the vertical size, or both sizes:

`gtk.SIZE_GROUP_NONE`	the group has no effect
`gtk.SIZE_GROUP_HORIZONTAL`	the group affects the horizontal requisition
`gtk.SIZE_GROUP_VERTICAL`	the group affects the vertical requisition
`gtk.SIZE_GROUP_BOTH`	the group affects both the horizontal and vertical requisition

Note that size groups only affect the amount of space requested, not the size that the widgets finally receive. If you want the widgets in a gtk.SizeGroup to actually be the same size, you need to pack them in such a way that they get the size they request and not more. For example, if you are packing your widgets into a table, you would not include the `gtk.FILL` flag. gtk.SizeGroup objects are referenced by each widget in the size group, so once you have added all widgets to a gtk.SizeGroup. If the widgets in the size group are subsequently destroyed, then they will be removed from the size group and drop their references on the size group; when all widgets have been removed, the size group will be freed.

Widgets can be part of multiple size groups; PyGTK will compute the horizontal size of a widget from the horizontal requisition of all widgets that can be reached from the widget by a chain of size groups of type `gtk.SIZE_GROUP_HORIZONTAL` or `gtk.SIZE_GROUP_BOTH`, and the vertical size from the vertical requisition of all widgets that can be reached from the widget by a chain of size groups of type `gtk.SIZE_GROUP_VERTICAL` or `gtk.SIZE_GROUP_BOTH`.

Constructor

```
gtk.SizeGroup(mode)
```

mode :	the mode for the new size group.
Returns :	a new gtk.SizeGroup

Creates a new gtk.SizeGroup with the mode specified by the value of *mode*:

`gtk.SIZE_GROUP_NONE`	the group has no effect
`gtk.SIZE_GROUP_HORIZONTAL`	the group affects the horizontal requisition
`gtk.SIZE_GROUP_VERTICAL`	the group affects the vertical requisition
`gtk.SIZE_GROUP_BOTH`	the group affects both the horizontal and vertical requisition

Methods

gtk.SizeGroup.set_mode

```
def set_mode(mode)
```

mode :	the mode to set for the size group.

The `set_mode()` method sets the "mode" property of the size group to the value specified by *`mode`*. The "mode" of the size group determines whether the widgets in the size group should all have the same horizontal requisition (`gtk.SIZE_GROUP_MODE_HORIZONTAL`) all have the same vertical requisition (`gtk.SIZE_GROUP_MODE_VERTICAL`), or should all have the same requisition in both directions (`gtk.SIZE_GROUP_MODE_BOTH`).

gtk.SizeGroup.get_mode

```
def get_mode()
```

Returns :	the current mode of the size group.

The `get_mode()` method returns the value of the "mode" property of the size group. See the `set_mode()` method.

gtk.SizeGroup.add_widget

```
def add_widget(widget)
```

`widget` :	the `gtk.Widget` to add

The `add_widget()` method adds the widget specified by *`widget`* to the `gtk.SizeGroup`. The requisition of the widget will then be determined as the maximum of its requisition and the requisition of the other widgets in the size group. Whether this applies horizontally, vertically, or in both directions depends on the mode of the size group. See the `set_mode()` method for more detail.

gtk.SizeGroup.remove_widget

```
def remove_widget(widget)
```

`widget` :	the `gtk.Widget` to remove

The remove_widget() method removes the widget specified by widget from the `gtk.SizeGroup`.

gtk.Socket

gtk.Socket — a container for widgets from other processes.

Synopsis

```
class gtk.Socket(gtk.Container):
    gtk.Socket()
    def add_id(window_id)
    def get_id()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Socket
```

Signal Prototypes

"plug–added"	`def callback(socket, user_param1, ...)`
"plug–removed"	`def callback(socket, user_param1, ...)`

Description

Together with a `gtk.Plug`, a `gtk.Socket` provides the ability to embed widgets from one process into another process in a fashion that is transparent to the user. One process creates a `gtk.Socket` widget and, passes the that widget's window ID to the other process, which then creates a `gtk.Plug` with that window ID. Any widgets contained in the `gtk.Plug` then will appear inside the first applications window. The socket's window ID is obtained by using the `get_id()` method. Before using this function, the socket must have been realized, and added to its parent.

When PyGTK is notified that the embedded window has been destroyed, then it will destroy the socket as well. You should always, therefore, be prepared for your sockets to be destroyed at any time when the main event loop is running. The communication between a `gtk.Socket` and a `gtk.Plug` follows the XEmbed protocol.

Constructor

```
  gtk.Socket()
```

Returns :	the new `gtk.Socket`.

Creates a new `gtk.Socket`.

Methods

gtk.Socket.add_id

```
  def add_id(window_id)
```

`window_id` :	the window ID of a client participating in the XEMBED protocol.

The `add_id()` method adds an XEMBED client specified by *`window_id`*, such as a `gtk.Plug`, to the `gtk.Socket`. The client may be in the same process or in a different process.

To embed a `gtk.Plug` in a `gtk.Socket`, you can either:

- create the `gtk.Plug` by calling gtk.Plug(), then call `gtk.Plug.get_id()` to get the window ID of the plug, and finally pass that to the `gtk.Socket.add_id()`; or,
- call the `gtk.Socket.get_id()` method to get the window ID for the socket, then create the `gtk.Plug` by calling gtk.Plug() passing in that ID. The `gtk.Socket` must have already be added into a toplevel window before you can make this call.

gtk.Socket.get_id

```
    def get_id()
```

Returns :	the window ID for the socket

The `get_id()` method gets the window ID of a `gtk.Socket` widget, which can then be used to create a client embedded inside the socket, for instance with gtk.Plug(). The `gtk.Socket` must have already be added into a toplevel window before you can make this call.

Signals

The "plug−added" gtk.Socket Signal

```
    def callback(socket, user_param1, ...)
```

`socket` :	the socket that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "plug−added" signal is emitted when a plug is added to the socket.

The "plug−removed" gtk.Socket Signal

```
    def callback(socket, user_param1, ...)
```

`socket` :	the socket that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)
Returns :	`TRUE` if the `gtk.Socket` should not be destroyed.

The "plug−removed" signal is emitted when a plug is removed from the socket. The default action is to destroy the `gtk.Socket` widget, so if you want to reuse it you must add a signal handler that returns `TRUE`.

Prev Up Next
gtk.SizeGroup Home gtk.SpinButton

gtk.SpinButton
Prev **The gtk Class Reference** Next

gtk.SpinButton

gtk.SpinButton — retrieve an integer or floating−point number from the user.

Synopsis

```
class gtk.SpinButton(gtk.Entry):
    gtk.SpinButton(adjustment=None, climb_rate=0.0, digits=0)
    def configure(adjustment, climb_rate, digits)
    def set_adjustment(adjustment)
    def get_adjustment()
    def set_digits(digits)
    def get_digits()
    def set_increments(step, page)
    def get_increments()
    def set_range(min, max)
    def get_range()
    def get_value()
    def get_value_as_int()
    def set_value(value)
    def set_update_policy(policy)
    def get_update_policy()
    def set_numeric(numeric)
    def get_numeric()
    def spin(direction, increment)
    def set_wrap(wrap)
    def get_wrap()
    def set_snap_to_ticks(snap_to_ticks)
    def get_snap_to_ticks()
    def update()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Entry (implements gtk.Editable, gtk.CellEditable)
        +-- gtk.SpinButton
```

Properties

"adjustment"	Read–Write	The adjustment that holds the value of the spinbutton
"climb–rate"	Read–Write	The acceleration rate when you hold down a button
"digits"	Read–Write	The number of decimal places to display
"numeric"	Read–Write	If `TRUE`, non–numeric characters should be ignored
"snap–to–ticks"	Read–Write	If `TRUE`, erroneous values are automatically changed to a spin button's nearest step increment
"update–policy"	Read–Write	either `gtk.UPDATE_ALWAYS` (the spin button should update always), or `gtk.UPDATE_IF_VALID` the spin button should update only when the value is legal)
"value"	Read–Write	the current value
"wrap"	Read–Write	If `TRUE`, a spin button should wrap upon reaching its limits

Style Properties

"shadow–type" Read	the shadow type of the spinbutton – one of: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN` or `gtk.SHADOW_ETCHED_OUT`

Signal Prototypes

"change–value"	`def callback(`*`spinbutton, scrolltype, user_param1, ...`*`)`
"input"	`def callback(`*`spinbutton, value_ptr, user_param1, ...`*`)`
"output"	`def callback(`*`spinbutton, user_param1, ...`*`)`
"value–changed"	`def callback(`*`spinbutton, user_param1, ...`*`)`

Description

A gtk.SpinButton is a subclass of gtk.Entry that provides a way for a user to set the value of some numeric attribute within a predefined range. Rather than having to directly type a number into a gtk.Entry, a gtk.SpinButton allows the user to click on one of two arrows to increment or decrement the displayed value. A value can still be typed in and validated. A gtk.SpinButton uses a gtk.Adjustment to specify the range and value of interest. See the gtk.Adjustment section for more details about an adjustment's properties.

Constructor

```
gtk.SpinButton(adjustment=None, climb_rate=0.0, digits=0)
```

`adjustment` :	a gtk.Adjustment or None to create a new adjustment
`climb_rate` :	the acceleration factor
`digits` :	the number of decimal places to display
Returns :	a new gtk.SpinButton widget

Creates a new gtk.SpinButton widget using the gtk.Adjustment specified by *`adjustment`*, the acceleration factor specified by *`climb_rate`* and displaying the number of decimals specified by *`digits`*. If *`adjustment`* is `None` or is not specified a new gtk.Adjustment will be created. If *`climb_rate`* is not specified it defaults to 0.0. If *`digits`* is not specified it defaults to 0.

Methods

gtk.SpinButton.configure

```
def configure(adjustment, climb_rate, digits)
```

`adjustment` :	a gtk.Adjustment or None to create a new adjustment
`climb_rate` :	the acceleration factor
`digits` :	the number of decimal places to display

The `configure`() method changes the properties of an existing spin button by associating the gtk.Adjustment specified by *`adjustment`*, setting the acceleration rate to the c value specified by *`climb_rate`* and setting the number of decimal places to display to the value specified by *`digits`*. If *`adjustment`* is `None` a new gtk.Adjustment will be created.

gtk.SpinButton.set_adjustment

```
def set_adjustment(adjustment)
```

`adjustment` :	a gtk.Adjustment to replace the existing adjustment

The `set_adjustment`() method sets the "adjustment" property to the value specified by *`adjustment`* replacing the current adjustment object associated with the spinbutton.

gtk.SpinButton.get_adjustment

```
def get_adjustment()
```

Returns :	the gtk.Adjustment object associated with the spinbutton

The `get_adjustment()` method returns the value of the "adjustment" property.

gtk.SpinButton.set_digits

```
def set_digits(digits)
```

`digits` :	the number of decimal places to be displayed for the spin button's value

The `set_digits()` method sets the "digits" property to the value specified by *`digits`*. The value of "digits" determines the number of decimal places (up to 20 digits) to be displayed by the spinbutton.

gtk.SpinButton.get_digits

```
def get_digits()
```

Returns :	the current number of decimal places to be displayed

The `get_digits()` method returns the value of the "digits" property. The value of "digits" determines the number of decimal places the spinbutton displays. See the `set_digits()` method for more detail.

gtk.SpinButton.set_increments

```
def set_increments(step, page)
```

`step` :	increment applied for each left mousebutton press.
`page` :	increment applied for each middle mousebutton press.

The `set_increments()` method sets the `step_increment` and `page_increment` attributes of the `gtk.Adjustment` associated with the spinbutton to the values specified by *`step`* and *`page`* respectively. These affect how quickly the value changes when the spin button's arrows are activated.

gtk.SpinButton.get_increments

```
def get_increments()
```

Returns :	a tuple containing the step and page increments

The `get_increments()` method returns a tuple containing the values of the `step_increment` and `page_increment` attributes of the `gtk.Adjustment` associated with the spinbutton. See the `set_increments()` method for more detail.

gtk.SpinButton.set_range

```
def set_range(min, max)
```

`min` :	the minimum allowable value
`max` :	the maximum allowable value

The `set_range()` method sets the minimum and maximum allowable values for spinbutton by setting the `lower` and `upper` attributes of the associated `gtk.Adjustment` to the values of *`min`* and *`max`* respectively.

gtk.SpinButton.get_range

```
def get_range()
```

Returns :	a tuple containing the minimum and maximum allowed values

The `get_range()` method returns a tuple containing the range allowed for the spinbutton. See the `set_range()` method for more detail.

gtk.SpinButton.get_value

```
    def get_value()
```

Returns :	the value of the spin_button

The `get_value()` method returns the value of the "value" property of the spinbutton (really the `value` attribute of the associated `gtk.Adjustment`).

gtk.SpinButton.get_value_as_int

```
    def get_value_as_int()
```

Returns :	the value of the spinbutton as an integer

The `get_value_as_int()` method returns the value of the spinbutton represented as an integer.

gtk.SpinButton.set_value

```
    def set_value(value)
```

`value` :	the new value

The `set_value()` method sets the value of the "value" property to the value specified by *`value`* (sets the `value` attribute of the associated `gtk.Adjustment`.

gtk.SpinButton.set_update_policy

```
    def set_update_policy(policy)
```

`policy` :	the new update policy

The `set_update_policy()` method sets the "update−policy" property to the value of *`policy`*. The value of *`policy`* is either of:

`gtk.UPDATE_ALWAYS`	the value is always displayed.
`gtk.UPDATE_IF_VALID`	the value is only displayed if it is valid within the bounds of the spinbutton's `gtk.Adjustment`.

gtk.SpinButton.get_update_policy

```
    def get_update_policy()
```

Returns :	the current update policy

The `get_update_policy()` method returns the value of the "update−policy" property that determines the update behavior of a spin button. See the `set_update_policy()` method for more detail.

gtk.SpinButton.set_numeric

```
    def set_numeric(numeric)
```

`numeric` :	a flag indicating if only numeric entry is allowed.

The `set_numeric`() method sets the value of the "numeric" property to the value of *`numeric`*. If *`numeric`* is `TRUE` only numeric text can be typed into the spin button.

gtk.SpinButton.get_numeric

```
def get_numeric()
```

Returns :	`TRUE` if only numeric text can be entered

The `get_numeric`() method returns the value of the "numeric" preoperty. See the `set_numeric()` method for more detail.

gtk.SpinButton.spin

```
def spin(direction, increment)
```

`direction` :	the direction to spin.
`increment` :	the step increment to apply in the specified direction.

The `spin`() method increments or decrements a spin button's value in the direction specified by *`direction`* with a step size specified by *`increment`*. The value of *`increment`* is only used if direction is `gtk.SPIN_USER_DEFINED`. The value of *`direction`* must be one of:

`gtk.SPIN_STEP_FORWARD`	forward by step_increment
`gtk.SPIN_STEP_BACKWARD`	backward by step_increment
`gtk.SPIN_PAGE_FORWARD`	forward by step_increment
`gtk.SPIN_PAGE_BACKWARD`	backward by step_increment
`gtk.SPIN_HOME`	move to minimum value
`gtk.SPIN_END`	move to maximum value
`gtk.SPIN_USER_DEFINED`	add *`increment`* to the value

gtk.SpinButton.set_wrap

```
def set_wrap(wrap)
```

`wrap` :	if `TRUE` wrapping is performed.

The `set_wrap`() method sets the "wrap" property to the value of *`wrap`*. If *`wrap`* is `TRUE` the spin button value wraps around to the opposite limit when the upper or lower limit of the range is exceeded.

gtk.SpinButton.get_wrap

```
def get_wrap()
```

Returns :	`TRUE` if the spin button wraps

The `get_wrap`() method returns the value of the "wrap" property. If the value of "wrap" is `TRUE` the spinbutton's value wraps around to the opposite limit when the upper or lower limit of the range is exceeded. See the `set_wrap()` method.

gtk.SpinButton.set_snap_to_ticks

```
def set_snap_to_ticks(snap_to_ticks)
```

`snap_to_ticks` :	if `TRUE` invalid values should be corrected.

The `set_snap_to_ticks()` method sets the "snap–to–ticks" property to the value of *`snap_to_ticks`*. If *`snap_to_ticks`* is `TRUE` values are corrected to the nearest step increment when a spin button is activated after providing an invalid value.

gtk.SpinButton.get_snap_to_ticks

```
def get_snap_to_ticks()
```

Returns :	`TRUE` if values are snapped to the nearest step.

The `get_snap_to_ticks()` method returns the value of the "snap–to–ticks" property. If the value of "snap–to–ticks" is `TRUE` the input values are corrected to the nearest step. See the `set_snap_to_ticks()` method.

gtk.SpinButton.update

```
def update()
```

The `update()` method manually forces an update of the spin button.

Signals

The "change–value" gtk.SpinButton Signal

```
def callback(spinbutton, scrolltype, user_param1, ...)
```

`spinbutton` :	the spinbutton that received the signal
`scrolltype` :	the scrolltype: `gtk.SCROLL_STEP_UP`, `gtk.SCROLL_STEP_DOWN`, `gtk.SCROLL_PAGE_UP`, `gtk.SCROLL_PAGE_DOWN, gtk.SCROLL_START or gtk.SCROLL_END`
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "change–value" signal is emitted when the spinbutton value is changed by keyboard action using the **Up Arrow**, **Down Arrow**, **Page Up**, **Page Down**, **Control+Page Up** or **Control+Page Down** keys.

The "input" gtk.SpinButton Signal

```
def callback(spinbutton, value_ptr, user_param1, ...)
```

`spinbutton` :	the spinbutton that received the signal
`value_ptr` :	a pointer to the value
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)
Returns :	`TRUE` if the input value was retrieved and handled; `FALSE` if not handled and −1 if an error occurred during processing.

The "input" signal is emitted when the value changes. The value_ptr is a GPointer to the value that cannot be accessed from PyGTK. This signal cannot be handled in PyGTK.

The "output" gtk.SpinButton Signal

```
def callback(spinbutton, user_param1, ...)
```

spinbutton :	the spinbutton that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	TRUE if no further processing is required.

The "output" signal is emitted when the spinbutton display value is changed either by setting a new value or changing the digits and on realizing the widget. Returns TRUE if the handler successfully set the text and no further processing is required.

The "value−changed" gtk.SpinButton Signal

```
def callback(spinbutton, user_param1, ...)
```

spinbutton :	the spinbutton that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "value−changed" signal is emitted when any of the settings (i.e. value, digits) that change the display of the spinbutton are changed.

Prev Up Next
gtk.Socket Home gtk.Statusbar

gtk.Statusbar
Prev The gtk Class Reference Next

gtk.Statusbar

gtk.Statusbar — report messages of minor importance to the user.

Synopsis

```
class gtk.Statusbar(gtk.HBox):
    gtk.Statusbar()
    def get_context_id(context_description)
    def push(context_id, text)
    def pop(context_id)
    def remove(context_id, message_id)
    def set_has_resize_grip(setting)
    def get_has_resize_grip()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Box
```

```
          +-- gtk.HBox
            +-- gtk.Statusbar
```

Properties

"has−resize−grip"	Read−Write	If TRUE, the statusbar has a grip for resizing the toplevel window. Available in GTK+ 2.4 and above.

Style Properties

"shadow−type"	Read	The style of bevel around the statusbar text

Signal Prototypes

"text−popped"	`def callback(statusbar, context_id, text, user_param1, ...)`
"text−pushed"	`def callback(statusbar, context_id, text, user_param1, ...)`

Description

A `gtk.Statusbar` is usually placed along the bottom of an application's main `gtk.Window`. It may provide a regular commentary of the application's status (as is usually the case in a web browser, for example), or may be used to simply output a message when the status changes, (when an upload is complete in an `FTP` client, for example). It may also have a resize grip (a triangular area in the lower right corner) which can be clicked on to resize the window containing the statusbar. Status bars in PyGTK maintain a stack of messages. The message at the top of the each bar's stack is the one that will currently be displayed.

Any messages added to a statusbar's stack must specify a `context_id` that is used to uniquely identify the source of a message. The `context_id` can be generated by the `get_context_id()` method, and associated with a context message. An existing context_id can be retrieved using the context message using the `get_context_id()` method. Note that messages are stored in a stack, and when choosing which message to display, the stack structure is adhered to, regardless of the context identifier of a message.

Constructor

```
  gtk.Statusbar()
```

Returns :	a new `gtk.Statusbar` widget

Creates a new `gtk.Statusbar` widget.

Methods

gtk.Statusbar.get_context_id

```
  def get_context_id(context_description)
```

`context_description` :	a string identifying the context for the message
Returns :	an integer context identifier

The `get_context_id()` method returns a new or existing context identifier, given a description of the actual context specified by *`context_description`*. In effect, `get_context_id()` both registers and retrieves a context identifier.

gtk.Statusbar.push

```
def push(context_id, text)
```

`context_id` :	a context identifier
`text` :	the message text
Returns :	an integer message identifier

The `push()` method pushes a new message specified by *`text`* with the specified *`context_id`* onto a statusbar's stack and returns a message id that that can be used with the `remove()` method.

gtk.Statusbar.pop

```
def pop(context_id)
```

`context_id` :	a context identifier

The `pop()` method removes the top message with the specified *`context_id`* from the statusbar's stack.

gtk.Statusbar.remove

```
def remove(context_id, message_id)
```

`context_id` :	the context identifier
`message_id` :	the message identifier

The `remove()` method removes the message with the specified *`message_id`* and *`context_id`* from the statusbar's message stack.

gtk.Statusbar.set_has_resize_grip

```
def set_has_resize_grip(setting)
```

`setting` :	if `TRUE` a resize grip is displayed

The `set_has_resize_grip()` method sets the internal "has_resize_grip" property to the value specified by *`setting`*. If *`setting`* is `TRUE` a resize grip is displayed on the statusbar

gtk.Statusbar.get_has_resize_grip

```
def get_has_resize_grip()
```

Returns :	`TRUE` if a resize grip is displayed

The `get_has_resize_grip()` method returns the value of the internal "has_resize_grip" property that determines if a resize grip is displayed on the statusbar.

Signals

The "text−popped" gtk.Statusbar Signal

```
    def callback(statusbar, context_id, text, user_param1, ...)
```

statusbar :	the statusbar that received the signal
context_id :	the context identifier of the top message
text :	the string containing the top message text
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "text−popped" signal is emitted when a message is removed from the statusbar message stack. Note the text and context_id are for the top message on the statusbar stack not the message that was actually removed.

The "text−pushed" gtk.Statusbar Signal

```
    def callback(statusbar, context_id, text, user_param1, ...)
```

statusbar :	the statusbar that received the signal
context_id :	the context identifier of the message added
text :	the string containing the message text
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "text−pushed" signal is emitted when a message is added to the statusbar message stack.

Prev	Up	Next
gtk.SpinButton	Home	gtk.Style

gtk.Style

Prev **The gtk Class Reference** Next

gtk.Style

gtk.Style — an object that hold style information for widgets

Synopsis

```
class gtk.Style(gobject.GObject):
    gtk.Style()
    def copy()
    def attach(window)
    def detach()
    def set_background(window, state_type)
    def apply_default_background(window, set_bg, state_type, area, x, y, width, height)
    def lookup_icon_set(stock_id)
    def render_icon(source, direction, state, size, widget, detail)
    def paint_hline(window, state_type, area, widget, detail, x1, x2, y)
    def paint_vline(window, state_type, area, widget, detail, y1, y2, x)
    def paint_shadow(window, state_type, shadow_type, area, widget, detail, x, y, width, height
    def paint_polygon(window, state_type, shadow_type, area, widget, detail, points, fill)
    def paint_arrow(window, state_type, shadow_type, area, widget, detail, arrow_type, fill, x,
    def paint_diamond(window, state_type, shadow_type, area, widget, detail, x, y, width, heigh
    def paint_box(window, state_type, shadow_type, area, widget, detail, x, y, width, height)
    def paint_flat_box(window, state_type, shadow_type, area, widget, detail, x, y, width, heig
    def paint_check(window, state_type, shadow_type, area, widget, detail, x, y, width, height)
    def paint_option(window, state_type, shadow_type, area, widget, detail, x, y, width, height
    def paint_tab(window, state_type, shadow_type, area, widget, detail, x, y, width, height)
    def paint_shadow_gap(window, state_type, shadow_type, area, widget, detail, x, y, width, he
    def paint_box_gap(window, state_type, shadow_type, area, widget, detail, x, y, width, heigh
    def paint_extension(window, state_type, shadow_type, area, widget, detail, x, y, width, hei
    def paint_focus(window, state_type, area, widget, detail, x, y, width, height)
    def paint_slider(window, state_type, shadow_type, area, widget, detail, x, y, width, height
    def paint_handle(window, state_type, shadow_type, area, widget, detail, x, y, width, height
    def paint_expander(window, state_type, area, widget, detail, x, y, expander_style)
    def paint_layout(window, state_type, use_text, area, widget, detail, x, y, layout)
    def paint_resize_grip(window, state_type, area, widget, detail, edge, x, y, width, height)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Style
```

Attributes

"fg"	Read	An array of gtk.gdk.Colors to be used for the foreground colors in each widget state.
"bg"	Read	An array of gtk.gdk.Colors to be used for the background colors in each widget state.
"light"	Read	An array of gtk.gdk.Colors to be used for the light colors in each widget state. The light colors are slightly lighter than the bg colors and used for creating shadows.
"dark"	Read	An array of gtk.gdk.Colors to be used for the dark colors in each widget state. The dark colors are slightly darker than the bg colors and used for creating shadows.
"mid"	Read	An array of gtk.gdk.Colors to be used for the mid colors (between light and dark) in each widget state
"text"	Read	An array of gtk.gdk.Colors to be used for the text colors in each widget state.
"base"	Read	An array of gtk.gdk.Colors to be used for the base colors in each widget state.
"text_aa"	Read	An array of gtk.gdk.Colors to be used for the anti−aliased text colors in each widget state.
"black"	Read	A gtk.gdk.Color that is used for the black color.
"white"	Read	A gtk.gdk.Color that is used for the white color.
"font_desc"	Read	A pango.FontDescription used as the default text font.
"xthickness"	Read	The thickness of lines drawn vertically.
"ythickness"	Read	The thickness of lines drawn horizontally.
"fg_gc"	Read	An array of graphics contexts (gtk.gdk.GC) for drawing using the fg colors.
"bg_gc"	Read	An array of graphics contexts (gtk.gdk.GC) for drawing using the bg colors.

"light_gc"	Read	An array of graphics contexts (`gtk.gdk.GC`) for drawing using the light colors.
"dark_gc"	Read	An array of graphics contexts (`gtk.gdk.GC`) for drawing using the dark colors.
"mid_gc"	Read	An array of graphics contexts (`gtk.gdk.GC`) for drawing using the mid colors.
"text_gc"	Read	An array of graphics contexts (`gtk.gdk.GC`) for drawing using the text colors.
"base_gc"	Read	An array of graphics contexts (`gtk.gdk.GC`) for drawing using the base colors.
"text_aa_gc"	Read	An array of graphics contexts (`gtk.gdk.GC`) for drawing using the anti−aliased text colors.
"black_gc"	Read	A graphics context (`gtk.gdk.GC`) for drawing using the black color.
"white_gc"	Read	A graphics context (`gtk.gdk.GC`) for drawing using the white color.
"bg_pixmap"	Read	An array of `gtk.gdk.Pixmap` to be used for the background stippling in each widget state.

Description

A `gtk.Style` object encapsulates the information that provides the look and feel for a widget. Each `gtk.Widget` has an associated `gtk.Style` object that is used when rendering that widget. Usually the `gtk.Style` for a widget is the same as the default style that is set by GTK and modified the theme engine. A `gtk.Style` holds information for the five possible widget states though not every widget supports all five states:

`gtk.STATE_NORMAL`	The state of a sensitive widget that is not active and does not have the focus
`gtk.STATE_ACTIVE`	The state of a sensitive widget when it is active e.g. a button that is pressed but not yet released
`gtk.STATE_PRELIGHT`	The state of a sensitive widget that has the focus e.g. a button that has the mouse pointer over it.
`gtk.STATE_SELECTED`	The state of a widget that is selected e.g. selected text in a `gtk.Entry` widget
`gtk.STATE_INSENSITIVE`	The state of a widget that is insensitive and will not respond to any events e.g. cannot be activated, selected or prelit.

A `gtk.Style` contains the read−only attributes described in the above section.

Usually applications should not need to use or modify the `gtk.Style` of their widgets.

Constructor

```
    gtk.Style()
```

Returns :	a new `gtk.Style` object

Creates a new `gtk.Style` object.

Methods

gtk.Style.copy

```
def copy()
```

Returns :	a copy of the gtk.Style object

The `copy`() method returns a copy of the style object.

gtk.Style.attach

```
def attach(window)
```

window :	a gtk.Window.
Returns :	a gtk.Style object

The `attach`() method adapts a style to the specified *window*. This process allocates the colors and creates the graphics contexts for the style specializing them to a particular visual and colormap. A new gtk.Style may be created if the style has already been adapted to a window with a different style and colormap.

gtk.Style.detach

```
def detach()
```

The `detach`() method detaches the style and frees its resources if it is no longer attached.

gtk.Style.set_background

```
def set_background(window, state_type)
```

window :	a gtk.gdk.Window
state_type :	a widget state

The `set_background`() method sets the background of *window* to the background color or pixmap of the style for the widget state specified by *state_type*.

gtk.Style.apply_default_background

```
def apply_default_background(window, set_bg, state_type, area, x, y, width, height)
```

window :	a gtk.gdk.Window
set_bg :	if `TRUE` use the bg_pixmap for the widget state
state_type :	the widget state
area :	the clipping area
x :	the x location
y :	the y location
width :	the width
height :	the height

The `apply_default_background`() method sets the background of the specified *window* in the area specified by *x*, *y*, *width* and *height*. The background color is derived from the bg_gc of the style with the state specified by *state_type*. If *area* is not `None` it specifies a clipping area for the application of the background.

gtk.Style.lookup_icon_set

```
    def lookup_icon_set(stock_id)
```

stock_id :	a stock ID
Returns :	a gtk.IconSet

The lookup_icon_set() method looks in the gtk.IconFactory list associated with the style and the default icon factory for the stock icon specified by *stock_id*. If the stock icon iconset is found it is returned.

gtk.Style.render_icon

```
    def render_icon(source, direction, state, size, widget=None, detail=None)
```

source :	the gtk.IconSource specifying the icon to render
direction :	a text direction
state :	a widget state
size :	the size to render the icon at. A size of –1 means render at the size of the source and don't scale.
widget :	a widget
detail :	a style detail
Returns :	a gtk.gdk.Pixbuf

The render_icon() method renders the icon specified by *source* at the specified *size* for the specified widget *state* according to the given parameters and returns the result in a gtk.gdk.Pixbuf.

gtk.Style.paint_hline

```
    def paint_hline(window, state_type, area, widget, detail, x1, x2, y)
```

window :	a gtk.gdk.Window
state_type :	a widget state
area :	the rectangle that clips the output
widget :	a widget
detail :	a detail string
x1 :	the starting x coordinate
x2 :	the ending x coordinate
y :	the y coordinate

The paint_hline() method draws a horizontal line from (*x1*, *y*) to (*x2*, *y*) in *window* using the specified *state_type* of the style. If *area* is not None the line is clipped by the rectangle specified by *area*.

gtk.Style.paint_vline

```
    def paint_vline(window, state_type, area, widget, detail, y1, y2, x)
```

window :	a gtk.gdk.Window
state_type :	a widget state
area :	the rectangle to which the output is clipped
widget :	a widget
detail :	a detail string
y1 :	the starting y coordinate

y2 :	the ending y coordinate
x :	the x coordinate

The `paint_vline()` method draws a vertical line from (*x*, *y1*) to (*x*, *y2*) in *window* using the specified *state_type* of the style. If *area* is not `None` the line is clipped by the rectangle specified by *area*.

gtk.Style.paint_shadow

```
    def paint_shadow(window, state_type, shadow_type, area, widget, detail, x, y, width, height)
```

window :	a `gtk.gdk.Window`*
state_type :	a widget state
shadow_type :	a type of shadow – one of: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`
area :	the rectangle to which the output is clipped
widget :	a widget
detail :	a detail string
x :	the x location
y :	the y location
width :	the width
height :	the height

The `paint_shadow()` method draws a shadow of the specified *shadow_type* on the specified *window* with the size specified by *x*, *y*, *width* and *height*. If *area* is not `None` the shadow is clipped to the rectangle' area. *state_type* specifies the style state to use for drawing.

gtk.Style.paint_polygon

```
    def paint_polygon(window, state_type, shadow_type, area, widget, detail, points, fill)
```

window :	a `gtk.gdk.Window`
state_type :	a widget state
shadow_type :	a shadow type – one of: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`
area :	a clipping rectangle
widget :	a widget
detail :	a detail string
points :	a list or tuple containing point (x, y) tuples
fill :	if `TRUE` the polygon should be filled

The `paint_polygon()` method draws a polygon on the specified *window* with the shadow type specified by *shadow_type* connecting the points specified by *points*. If *area* is not `None` it specifies a clipping rectangle. The style state specified by *state_type* determines the graphics context to use while drawing.

gtk.Style.paint_arrow

```
    def paint_arrow(window, state_type, shadow_type, area, widget, detail, arrow_type, fill, x,
```

window :	a `gtk.gdk.Window`
state_type :	the widget state
shadow_type :	

	a shadow type – one of: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`
area :	a clipping rectangle
widget :	a widget
detail :	a detail string
arrow_type :	an arrow type – one of: `gtk.ARROW_UP`, `gtk.ARROW_DOWN`, `gtk.ARROW_LEFT`, `gtk.ARROW_RIGHT`
fill :	if `TRUE` the arrow should be filled
x :	the x location
y :	the y location
width :	the width
height :	the height

The `paint_arrow()` method draws an arrow of the type specified by *arrow_type* on the specified *window* with the shadow type specified by *shadow_type* with the location and size specified by *x*, *y*, *width* and *height*. If *area* is not `None` it specifies a clipping rectangle. The widget state specified by *state_type* determines the graphics context to use while drawing.

gtk.Style.paint_diamond

```
    def paint_diamond(window, state_type, shadow_type, area, widget, detail, x, y, width, height
```

window :	a `gtk.gdk.Window`
state_type :	a widget state
shadow_type :	a shadow type – one of: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`
area :	a clipping rectangle
widget :	a widget
detail :	a detail string
x :	the x location
y :	the y location
width :	the width
height :	the height

The `paint_diamond()` method draws a diamond on the specified *window* with the shadow type specified by *shadow_type* and the location and size specified by *x*, *y*, *width* and *height*. If *area* is not `None` it specifies a clipping rectangle. The widget state specified by *state_type* determines the graphics context to use while drawing.

gtk.Style.paint_box

```
    def paint_box(window, state_type, shadow_type, area, widget, detail, x, y, width, height)
```

window :	a `gtk.gdk.Window`
state_type :	a widget state
shadow_type :	a shadow type – one of: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`
area :	a clipping rectangle
widget :	a widget
detail :	a detail string

x :	the x location
y :	the y location
width :	the width
height :	the height

The `paint_box()` method draws a box on the specified *window* with the shadow type specified by *shadow_type* and the location and size specified by *x*, *y*, *width* and *height*. If *area* is not `None` it specifies a clipping rectangle. The widget state specified by *state_type* determines the graphics context to use while drawing.

gtk.Style.paint_flat_box

```
    def paint_flat_box(window, state_type, shadow_type, area, widget, detail, x, y, width, heigh
```

window :	a gtk.gdk.Window
state_type :	a widget state
shadow_type :	a shadow type – one of: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`
area :	a clipping rectangle
widget :	a widget
detail :	a detail string
x :	the x location
y :	the y location
width :	the width
height :	the height

The `paint_flat_box()` method draws a flat box (no shadow) on the specified *window* with the location and size specified by *x*, *y*, *width* and *height*. If *area* is not `None` it specifies a clipping rectangle. The style state specified by *state_type* determines the graphics context to use while drawing.

gtk.Style.paint_check

```
    def paint_check(window, state_type, shadow_type, area, widget, detail, x, y, width, height)
```

window :	a gtk.gdk.Window
state_type :	a widget state
shadow_type :	a shadow type – one of: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`
area :	a clipping rectangle
widget :	a widget
detail :	a detail string
x :	the x location
y :	the y location
width :	the width
height :	the height

The `paint_check()` method draws a check on the specified *window* with the shadow type specified by *shadow_type* and the location and size specified by *x*, *y*, *width* and *height*. If *area* is not `None` it specifies a clipping rectangle. The widget state specified by *state_type* determines the graphics context to use while drawing.

gtk.Style.paint_option

```
    def paint_option(window, state_type, shadow_type, area, widget, detail, x, y, width, height)
```

window :	a gtk.gdk.Window
state_type :	a widget state
shadow_type :	a shadow type – one of: gtk.SHADOW_NONE, gtk.SHADOW_IN, gtk.SHADOW_OUT, gtk.SHADOW_ETCHED_IN, gtk.SHADOW_ETCHED_OUT
area :	a clipping rectangle
widget :	a widget
detail :	a detail string
x :	the x location
y :	the y location
width :	the width
height :	the height

The paint_option() method draws an option menu item on the specified *window* with the shadow type specified by *shadow_type* and the location and size specified by *x*, *y*, *width* and *height*. If *area* is not None it specifies a clipping rectangle. The widget state specified by *state_type* determines the graphics context to use while drawing.

gtk.Style.paint_tab

```
    def paint_tab(window, state_type, shadow_type, area, widget, detail, x, y, width, height)
```

window :	a gtk.gdk.Window
state_type :	a widget state
shadow_type :	a shadow type – one of: gtk.SHADOW_NONE, gtk.SHADOW_IN, gtk.SHADOW_OUT, gtk.SHADOW_ETCHED_IN, gtk.SHADOW_ETCHED_OUT
area :	a clipping rectangle
widget :	a widget
detail :	a detail string
x :	the x location
y :	the y location
width :	the width
height :	the height

The paint_tab() method draws a tab on the specified *window* with the shadow type specified by *shadow_type* and the location and size specified by *x*, *y*, *width* and *height*. If *area* is not None it specifies a clipping rectangle. The widget state specified by *state_type* determines the graphics context to use while drawing.

gtk.Style.paint_shadow_gap

```
    def paint_shadow_gap(window, state_type, shadow_type, area, widget, detail, x, y, width, hei
```

window :	a gtk.gdk.Window
state_type :	a widget state
shadow_type :	a shadow type – one of: gtk.SHADOW_NONE, gtk.SHADOW_IN, gtk.SHADOW_OUT, gtk.SHADOW_ETCHED_IN, gtk.SHADOW_ETCHED_OUT
area :	a clipping rectangle

widget :	a widget
detail :	a detail string
x :	the x location
y :	the y location
width :	the width
height :	the height
gap_side :	a position – one of: `gtk.POS_LEFT`, `gtk.POS_RIGHT`, `gtk.POS_TOP`, `gtk.POS_BOTTOM`
gap_x :	the x position of the gap
gap_width :	the gap width

The `paint_shadow_gap()` method draws a shadow with a gap on the specified *window* with the shadow type specified by *shadow_type* and the location and size specified by *x*, *y*, *width* and *height*. The gap position and width is specified by *gap_side*, *gap_x* and *gap_width*. If *area* is not `None` it specifies a clipping rectangle. The widget state specified by *state_type* determines the graphics context to use while drawing.

gtk.Style.paint_box_gap

```
    def paint_box_gap(window, state_type, shadow_type, area, widget, detail, x, y, width, height
```

window :	a gtk.gdk.Window
state_type :	a widget state
shadow_type :	a shadow type – one of: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`
area :	a clipping rectangle
widget :	a widget
detail :	a detail string
x :	the x location
y :	the y location
width :	the width
height :	the height
gap_side :	a position – one of: `gtk.POS_LEFT`, `gtk.POS_RIGHT`, `gtk.POS_TOP`, `gtk.POS_BOTTOM`
gap_x :	the x position of the gap
gap_width :	the gap width

The `paint_box_gap()` method draws a box with a gap on the specified *window* with the shadow type specified by *shadow_type* and the location and size specified by *x*, *y*, *width* and *height*. The gap position and width is specified by *gap_side*, *gap_x* and *gap_width*. If *area* is not `None` it specifies a clipping rectangle. The widget state specified by *state_type* determines the graphics context to use while drawing.

gtk.Style.paint_extension

```
    def paint_extension(window, state_type, shadow_type, area, widget, detail, x, y, width, heig
```

window :	a gtk.gdk.Window
state_type :	a widget state
shadow_type :	a shadow type – one of: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`

area :	a clipping rectangle
widget :	a widget
detail :	a detail string
x :	the x location
y :	the y location
width :	the width
height :	the height
gap_side :	a position – one of: `gtk.POS_LEFT`, `gtk.POS_RIGHT`, `gtk.POS_TOP`, `gtk.POS_BOTTOM`

The `paint_extension()` method draws an extension on the specified *`window`* with the shadow type specified by *`shadow_type`* and the location and size specified by *`x`*, *`y`*, *`width`* and *`height`*. The extension position is specified by *`gap_side`*. If *`area`* is not `None` it specifies a clipping rectangle. The widget state specified by *`state_type`* determines the graphics context to use while drawing.

gtk.Style.paint_focus

```
    def paint_focus(window, state_type, area, widget, detail, x, y, width, height)
```

window :	a gtk.gdk.Window
state_type :	a widget state
area :	a clipping rectangle
widget :	a widget
detail :	a detail string
x :	the x location
y :	the y location
width :	the width
height :	the height

The `paint_focus()` method draws a focus indicator on the specified *`window`* with the location and size specified by *`x`*, *`y`*, *`width`* and *`height`*. If *`area`* is not `None` it specifies a clipping rectangle. The widget state specified by *`state_type`* determines the graphics context to use while drawing.

gtk.Style.paint_slider

```
    def paint_slider(window, state_type, shadow_type, area, widget, detail, x, y, width, height,
```

window :	a gtk.gdk.Window
state_type :	a widget state
shadow_type :	a shadow type – one of: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`
area :	a clipping rectangle
widget :	a widget
detail :	a detail string
x :	the x location
y :	the y location
width :	the width
height :	the height
orientation :	a position – one of: `gtk.ORIENTATION_HORIZONTAL` or `gtk.ORIENTATION_VERTICAL`

The `paint_slider()` method draws a slider with the specified *`orientation`* on the specified *`window`* with the shadow type specified by *`shadow_type`* and the location and size specified by *`x`*, *`y`*, *`width`* and *`height`*. If *`area`* is not `None` it specifies a clipping rectangle. The widget state specified by *`state_type`* determines the graphics context to use while drawing.

gtk.Style.paint_handle

```
def paint_handle(window, state_type, shadow_type, area, widget, detail, x, y, width, height,
```

`window` :	a `gtk.gdk.Window`
`state_type` :	a widget state
`shadow_type` :	a shadow type – one of: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN`, `gtk.SHADOW_ETCHED_OUT`
`area` :	a clipping rectangle
`widget` :	a widget
`detail` :	a detail string
`x` :	the x location
`y` :	the y location
`width` :	the width
`height` :	the height
`orientation` :	a position – one of: `gtk.ORIENTATION_HORIZONTAL` or `gtk.ORIENTATION_VERTICAL`

The `paint_handle()` method draws a handle with the specified *`orientation`* on the specified *`window`* with the shadow type specified by *`shadow_type`* and the location and size specified by *`x`*, *`y`*, *`width`* and *`height`*. If *`area`* is not `None` it specifies a clipping rectangle. The widget state specified by *`state_type`* determines the graphics context to use while drawing.

gtk.Style.paint_expander

```
def paint_expander(window, state_type, area, widget, detail, x, y, expander_style)
```

`window` :	a `gtk.gdk.Window`
`state_type` :	a widget state
`area` :	a clipping rectangle
`widget` :	a widget
`detail` :	a detail string
`x` :	the x location
`y` :	the y location
`expander_style` :	an expander style – one of: `gtk.EXPANDER_COLLAPSED`, `gtk.EXPANDER_SEMI_COLLAPSED`, `gtk.EXPANDER_SEMI_EXPANDED` or `gtk.EXPANDER_EXPANDED`

The `paint_expander()` method draws an expander with the specified *`expander_style`* on the specified *`window`* at the location specified by *`x`*, *`y`*. If *`area`* is not `None` it specifies a clipping rectangle. The widget state specified by *`state_type`* determines the graphics context to use while drawing.

gtk.Style.paint_layout

```
def paint_layout(window, state_type, use_text, area, widget, detail, x, y, layout)
```

window :	a gtk.gdk.Window
state_type :	a widget state
use_text :	if TRUE use the text graphics context for drawing
area :	a clipping rectangle
widget :	a widget
detail :	a detail string
x :	the x location
y :	the y location
layout :	a Pango.Layout object containing the text to display

The paint_layout() method draws the text in a pango.Layout specified by *layout* on the specified *window* at the location specified by *x* and *y*. If text is TRUE use the text graphics context of the style for drawing, otherwise use the fg graphics context. If *area* is not None it specifies a clipping rectangle. The widget state specified by *state_type* determines the graphics context to use while drawing.

gtk.Style.paint_resize_grip

```
    def paint_resize_grip(window, state_type, area, widget, detail, edge, x, y, width, height)
```

window :	a gtk.gdk.Window
state_type :	a widget state
area :	a clipping rectangle
widget :	a widget
detail :	a detail string
edge :	the edge in which to draw the resize grip, currently only gtk.WINDOW_EDGE_SOUTH_EAST is implemented
x :	the x location
y :	the y location
width :	the width
height :	the height

The paint_resize_grip() method draws a resize grip at the specified *edge* on the specified *window* with the location and size specified by *x*, *y*, *width* and *height*. If *area* is not None it specifies a clipping rectangle. The widget state specified by *state_type* determines the graphics context to use while drawing. Currently the grip can only be drawn at the gtk.WINDOW_EDGE_SOUTH_EAST (lower right) edge.

Prev Up Next
gtk.Statusbar Home gtk.Table

gtk.Table
Prev The gtk Class Reference Next

gtk.Table

gtk.Table — layout widgets in a two−dimensional array

Synopsis

```
class gtk.Table(gtk.Container):
    gtk.Table(rows=1, columns=1, homogeneous=FALSE)
    def resize(rows, columns)
    def attach(child, left_attach, right_attach, top_attach, bottom_attach, xoptions=gtk.EXPAND
    def set_row_spacing(row, spacing)
    def get_row_spacing(row)
    def set_col_spacing(column, spacing)
    def get_col_spacing(column)
    def set_row_spacings(spacing)
    def get_default_row_spacing()
    def set_col_spacings(spacing)
    def get_default_col_spacing()
    def set_homogeneous(homogeneous)
    def get_homogeneous()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Table
```

Properties

"column−spacing"	Read−Write	The amount of space between two adjacent columns
"homogeneous"	Read−Write	If `TRUE`, the table cells are all the same width or height
"n−columns"	Read−Write	The number of columns in the table
"n−rows"	Read−Write	The number of rows in the table
"row−spacing"	Read−Write	The amount of space between two adjacent rows

Child Properties

"bottom−attach"	Read−Write	The lowest row of the child
"left−attach"	Read−Write	The leftmost column of the child
"right−attach"	Read−Write	The rightmost column of the child
"top−attach"	Read−Write	The uppermost row of the child
"x−options"	Read−Write	the horizontal behavior of the child − a combination of: `gtk.EXPAND`, `gtk.SHRINK` and `gtk.FILL`
"x−padding"	Read−Write	Extra space added between the child widget and its left and right neighbors, in pixels
"y−options"	Read−Write	the vertical behavior of the child − a combination of: `gtk.EXPAND`, `gtk.SHRINK` and `gtk.FILL`
"y−padding"	Read−Write	Extra space added between the child widget and its top and bottom neighbors, in pixels

Description

The `gtk.Table` manages a group of widgets that are arranged in rows and columns, making it easy to align many widgets next to each other, horizontally and vertically. Tables are created with a call to gtk.Table(). The size of a table can be changed using the `resize()` method.

Widgets can be added to a table using the `attach()` method. To alter the space of the row next to a specific row, use the `set_row_spacing()` method, and for a column, the `set_col_spacing()` method. The gaps between all rows or columns can be changed by calling the set_row_spacings() or set_col_spacings() methods respectively. The set_homogeneous() method changes the setting that determines whether all cells in the table will resize themselves to the size of the largest widget in the table.

Constructor

```
    gtk.Table(rows=1, columns=1, homogeneous=FALSE)
```

rows :	the number of rows
columns :	the number of columns
homogeneous :	if `TRUE` all table cells will be the same size as the largest cell
Returns :	a new `gtk.Table` widget

Creates a new `gtk.Table` widget with the number of rows and columns specified by the value of *rows* and *columns* respectively. The value of *rows* and *columns* must be in the range 0 .. 65535. If *homogeneous* is `TRUE` the table cells will all be the same size as the largest cell. If *rows* or *columns* are not specified they default to 1.

Methods

gtk.Table.resize

```
    def resize(rows, columns)
```

rows :	The new number of rows.
columns :	The new number of columns.

The `resize()` method changes the size of the table as specified by the parameters, *rows* and *columns*.

gtk.Table.attach

```
    def attach(child, left_attach, right_attach, top_attach, bottom_attach, xoptions=gtk.EXPAND
```

child :	the widget to add.
left_attach :	the column number to attach the left side of a child widget to.
right_attach :	the column number to attach the right side of a child widget to.
top_attach :	the row number to attach the top side of a child widget to.
bottom_attach :	the row number to attach the bottom side of a child widget to.
xoptions :	used to specify the properties of the child widget when the table is resized horizontally.
yoptions :	used to specify the properties of the child widget when the table is resized vertically.
xpadding :	the amount of padding to add on the left and right of the widget
ypadding :	the amount of padding to add above and below the widget

The `attach()` method adds the widget specified by *child* to the table. The number of 'cells' that a widget will occupy is specified by:

- *left_attach* – the column to the left of the widget
- *right_attach* – the column to the right of the widget
- *top_attach* – the row above the widget and
- *bottom_attach* – the row below the widget

The *xoptions* and *yoptions* determine the expansion properties of the widget in the horizontal and vertical directions respectively (the default value is `gtk.FILL|gtk.EXPAND`). The value of the options is a combination of:

`gtk.EXPAND`	the table cell should expand to take up any extra space that has been allocated to the table.
`gtk.SHRINK`	the widget should shrink when the table cell shrinks.
`gtk.FILL`	the widget should fill the space allocated to it in the table cell.

The *xpadding* and *ypadding* parameters determine the extra padding added around the widget. By default these are 0.

gtk.Table.set_row_spacing

```
    def set_row_spacing(row, spacing)
```

row :	the row number whose spacing will be changed.
spacing :	the number of pixels of added spacing

The `set_row_spacing()` method sets the spacing in pixels (specified by *spacing*) between the specified *row* and the following row.

gtk.Table.get_row_spacing

```
    def get_row_spacing(row)
```

row :	a row in the table, 0 indicates the first row
Returns :	the row spacing

The `get_row_spacing()` method returns the amount of space between the specified *row*, and the following row. See the `set_row_spacing()` method.

gtk.Table.set_col_spacing

```
    def set_col_spacing(column, spacing)
```

column :	the column number whose spacing will be changed.
spacing :	the number of pixels of added spacing

The `set_col_spacing()` method sets the spacing in pixels (specified by *spacing*) between the specified *column* and the following column.

gtk.Table.get_col_spacing

```
    def get_col_spacing(column)
```

column :	a column in the table, 0 indicates the first column
Returns :	the column spacing

The `get_col_spacing()` returns the amount of space between the specified *column*, and the following column. See the `set_col_spacing()` method.

gtk.Table.set_row_spacings

```
    def set_row_spacings(spacing)
```

spacing :	the number of pixels of space to place between every row in the table.

The `set_row_spacings()` method sets the "row−spacing" property, that determines the space between every row in table, to the value of *`spacing`*.

gtk.Table.get_default_row_spacing

```
    def get_default_row_spacing()
```

Returns :	the default row spacing

The `get_default_row_spacing()` method returns the value of the "row−spacing" property that specifies the default row spacing for the table i.e. the spacing that will be used for newly added rows. (See the `set_row_spacings()`)

gtk.Table.set_col_spacings

```
    def set_col_spacings(spacing)
```

`spacing` :	the number of pixels of space to place between every column in the table.

The `set_col_spacings()` method sets the "column−spacing" property, that determines the space between every column in table, to the value of *`spacing`*.

gtk.Table.get_default_col_spacing

```
    def get_default_col_spacing()
```

Returns :	the default column spacing

The `get_default_col_spacing()` method returns the value of the "column−spacing" property to the default column spacing for the table i.e. the spacing that will be used for newly added columns. (See the `set_col_spacings()`)

gtk.Table.set_homogeneous

```
    def set_homogeneous(homogeneous)
```

`homogeneous` :	if `TRUE` all cells will be the same size as the largest cell

The `set_homogeneous()` method sets the "homogeneous" property to the value of *`homogeneous`*. If homogeneous is `TRUE` all cells will be the same size as the largest cell.

gtk.Table.get_homogeneous

```
    def get_homogeneous()
```

Returns :	`TRUE` if the cells are all set to the same size

The `get_homogeneous()` method returns the value of the "homogeneous" property. If the value of "homogeneous" is `TRUE` all cells are set to the same width and height. (See the `set_homogeneous()` method)

Prev	Up	Next
gtk.Style	Home	gtk.TearoffMenuItem

	gtk.TearoffMenuItem	
Prev	**The gtk Class Reference**	Next

gtk.TearoffMenuItem

gtk.TearoffMenuItem a menu item used to tear off and reattach its menu.

Synopsis

```
class gtk.TearoffMenuItem(gtk.MenuItem):
    gtk.TearoffMenuItem()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Item
            +-- gtk.MenuItem
              +-- gtk.TearoffMenuItem
```

Description

A `gtk.TearoffMenuItem` is a special `gtk.MenuItem` which is used to tear off and reattach its menu. When its menu is shown normally, the `gtk.TearoffMenuItem` is drawn as a dotted line indicating that the menu can be torn off. Activating it causes its menu to be torn off and displayed in its own window as a tearoff menu. When its menu is shown as a tearoff menu, the `gtk.TearoffMenuItem` is drawn as a dotted line which has a left pointing arrow graphic indicating that the tearoff menu can be reattached. Activating it will remove the tearoff menu window.

Constructor

```
   gtk.TearoffMenuItem()
```

Returns :	a new `gtk.TearoffMenuItem` widget

Creates a new `gtk.TearoffMenuItem` widget.

gtk.TextAttributes

gtk.TextAttributes an object containing the attributes set on some text

Synopsis

```
class gtk.TextAttributes(gobject.GBoxed):
    gtk.TextAttributes()
    def copy()
    def copy_values(dest)
```

Attributes

"bg_color"	Read	The background color
"fg_color"	Read	The foreground color
"bg_stipple"	Read	The background stipple bitmap
"fg_stipple"	Read	The foreground stipple bitmap
"rise"	Read	The subscript or superscript rise
"underline"	Read	The style of underline – one of: `pango.UNDERLINE_NONE`, `pango.UNDERLINE_SINGLE`, `pango.UNDERLINE_DOUBLE`, `pango.UNDERLINE_LOW`
"strikethrough"	Read	If `TRUE` strikethrough the text
"draw_bg"	Read	If `TRUE` some background attributes are set
"justification"	Read	The type of justification – one of: `gtk.JUSTIFY_LEFT`, `gtk.JUSTIFY_RIGHT`, `gtk.JUSTIFY_CENTER`, `gtk.JUSTIFY_FILL`
"direction"	Read	The text direction – one of: `gtk.TEXT_DIR_NONE`, `gtk.TEXT_DIR_LTR`, `gtk.TEXT_DIR_RTL`
"font"	Read	A `pango.FontDescription`
"font_scale"	Read	The scale of the font e.g. 2.5
"left_margin"	Read	The width of the left margin in pixels
"indent"	Read	The width of the paragraph indent in pixels
"right_margin"	Read	The width of the right margin
"pixels_above_lines"	Read	The number of pixels space above a paragraph
"pixels_below_lines"	Read	The number of pixels space below a paragraph
"pixels_inside_wrap"	Read	The number of pixels of space between wrapped lines in a paragraph
"tabs"	Read	A set of tabs contained in a `pango.TabArray`
"wrap_mode"	Read	The wrap mode – one of: `gtk.WRAP_NONE`, `gtk.WRAP_CHAR`, `gtk.WRAP_WORD`
"language"	Read	The `pango.Language` object describing the text language
"invisible"	Read	If `TRUE` the text is hidden (Not implemented in PyGTK2)
"bg_full_height"	Read	If `TRUE` the background is fit to the full line height
"editable"	Read	If `TRUE` the text is editable
"realized"	Read	If `TRUE` the text has been realized
"pad1"	Read	

"pad2"	Read
"pad3"	Read
"pad4"	Read

Description

A `gtk.TextAttributes` object holds a set of attributes that describe the properties of a section of text. A `gtk.TextAttributes` object is usually obtained by calling either of the `gtk.TextIter.get_attributes()` or `gtk.TextView.get_default_attributes()` methods to retrieve the attributes in effect.

A `gtk.TextAttributes` object created with gtk.TextAttributes() cannot be applied within PyGTK because there is no way to set the attributes. Likewise, the `copy()` and `copy_values()` methods can create a new copy or copy the attributes but there are no methods in PyGTK that take a `gtk.TextAttributes` object as an argument. The most effective way to use a `gtk.TextAttributes` object is to read its attributes and use them to set the properties of a `gtk.TextTag`.

Constructor

```
gtk.TextAttributes()
```

Returns :	a new `gtk.TextAttributes`

Creates a `gtk.TextAttributes` object, that contains a set of attributes of some text.

Methods

gtk.TextAttributes.copy

```
def copy()
```

Returns :	a copy of the `gtk.TextAttributes` object

The `copy()` method copies the text attributes and returns a new `gtk.TextAttributes` object.

gtk.TextAttributes.copy_values

```
def copy_values(dest)
```

`dest` :	the `gtk.TextAttributes` whose attributes will be set

The `copy_values()` method copies the values from the `gtk.TextAttributes` object to *`dest`* so that *`dest`* has the same values. Frees existing values in *`dest`*.

Prev	Up	Next
gtk.TearoffMenuItem	Home	gtk.TextBuffer

gtk.TextBuffer

Prev **The gtk Class Reference** Next

gtk.TextBuffer

gtk.TextBuffer — stores attributed text for display in a gtk.TextView

Synopsis

```
class gtk.TextBuffer(gobject.GObject):
    gtk.TextBuffer(table=None)
    def get_line_count()
    def get_char_count()
    def get_tag_table()
    def set_text(text)
    def insert(iter, text)
    def insert_at_cursor(text)
    def insert_interactive(iter, text, default_editable)
    def insert_interactive_at_cursor(text, default_editable)
    def insert_range(iter, start, end)
    def insert_range_interactive(iter, start, end, default_editable)
    def insert_with_tags(iter, text, ...)
    def insert_with_tags_by_name(iter, text, ...)
    def delete(start, end)
    def delete_interactive(start_iter, end_iter, default_editable)
    def get_text(start, end, include_hidden_chars=TRUE)
    def get_slice(start, end, include_hidden_chars=TRUE)
    def insert_pixbuf(iter, pixbuf)
    def insert_child_anchor(iter, anchor)
    def create_child_anchor(iter)
    def create_mark(mark_name, where, left_gravity=FALSE)
    def move_mark(mark, where)
    def delete_mark(mark)
    def get_mark(name)
    def move_mark_by_name(name, where)
    def delete_mark_by_name(name)
    def get_insert()
    def get_selection_bound()
    def place_cursor(where)
    def select_range(ins, bound)
    def apply_tag(tag, start, end)
    def remove_tag(tag, start, end)
    def apply_tag_by_name(name, start, end)
    def remove_tag_by_name(name, start, end)
    def remove_all_tags(start, end)
    def create_tag(tag_name=None, ...)
    def get_iter_at_line_offset(line_number, char_offset)
    def get_iter_at_line_index(line_number, byte_index)
    def get_iter_at_offset(char_offset)
    def get_iter_at_line(line_number)
    def get_start_iter()
    def get_end_iter()
    def get_bounds()
    def get_iter_at_mark(mark)
    def get_iter_at_child_anchor(anchor)
    def get_modified()
    def set_modified(setting)
    def add_selection_clipboard(clipboard)
    def remove_selection_clipboard(clipboard)
    def cut_clipboard(clipboard, default_editable)
    def copy_clipboard(clipboard)
    def paste_clipboard(clipboard, override_location, default_editable)
    def get_selection_bounds()
    def delete_selection(interactive, default_editable)
    def begin_user_action()
    def end_user_action()
    def backspace(iter, interactive, default_editable)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.TextBuffer
```

Properties

"tag–table"	Read–Write–Construct Only	The gtk.TextTagTable associated with the textbuffer. Available in GTK+ 2.2 and above.

Attributes

"tag_table"	Read	The gtk.TextTagTable associated with the textbuffer.

Signal Prototypes

"apply–tag"	def callback(*textbuffer*, *texttag*, *start*, *end*, *user_param1*, *...*)
"begin–user–action"	def callback(*textbuffer*, *user_param1*, *...*)
"changed"	def callback(*textbuffer*, *user_param1*, *...*)
"delete–range"	def callback(*textbuffer*, *start*, *end*, *user_param1*, *...*)
"end–user–action"	def callback(*textbuffer*, *user_param1*, *...*)
"insert–child–anchor"	def callback(*textbuffer*, *iter*, *anchor*, *user_param1*, *...*)
"insert–pixbuf"	def callback(*textbuffer*, *iter*, *pixbuf*, *user_param1*, *...*)
"insert–text"	def callback(*textbuffer*, *iter*, *text*, *length*, *user_param1*, *...*)
"mark–deleted"	def callback(*textbuffer*, *textmark*, *user_param1*, *...*)
"mark–set"	def callback(*textbuffer*, *iter*, *textmark*, *user_param1*, *...*)
"modified–changed"	def callback(*textbuffer*, *user_param1*, *...*)
"remove–tag"	def callback(*textbuffer*, *texttag*, *start*, *end*, *user_param1*, *...*)

Description

A `gtk.TextBuffer` is the core component of the PyGTK text editing system. A `gtk.TextBuffer` contains the text, pixbufs and child widget anchors that are displayed in one or more `gtk.TextView` widgets. A `gtk.TextBuffer` has an associated `gtk.TextTagTable` that contains the `gtk.TextTag` objects that can be used to set attributes on the text in the textbuffer.

A `gtk.TextBuffer` can be automatically created when creating a `gtk.TextView` or it can be created with the gtk.TextBuffer() constructor and associated with a `gtk.TextView` using the `set_buffer()` method or the gtk.TextView() constructor.

Constructor

```
gtk.TextBuffer(table=None)
```

`table` :	a tag table, or None to create a new one
Returns :	a new text buffer object

Creates a new `gtk.TextBuffer` object.

Methods

gtk.TextBuffer.get_line_count

```
def get_line_count()
```

Returns :	the number of lines in the buffer

The `get_line_count()` method returns the number of lines in the buffer. This value is cached, so the function is very fast.

gtk.TextBuffer.get_char_count

```
def get_char_count()
```

Returns :	the number of characters in the buffer

The `get_char_count()` method returns the number of characters in the buffer; note that characters and bytes are not the same, you can't e.g. expect the contents of the buffer in string form to be this many bytes long. The character count is cached, so this function is very fast.

gtk.TextBuffer.get_tag_table

```
def get_tag_table()
```

Returns :	the buffer's tag table

The `get_tag_table()` method returns the `gtk.TextTagTable` object associated with the textbuffer.

gtk.TextBuffer.set_text

```
def set_text(text)
```

`text` :	UTF−8 text to insert

The `set_text()` method replaces the current contents of the textbuffer with the contents of *`text`*. *`text`* must be valid UTF−8.

gtk.TextBuffer.insert

```
def insert(iter, text)
```

`iter` :	a position in the buffer
`text` :	UTF−8 format text to insert

The `insert()` method inserts the contents of *`text`* into the textbuffer at the position specified by *`iter`*. The "insert_text" signal is emitted and the text insertion actually occurs in the default handler for the signal. *`iter`* is invalidated when insertion occurs (because the buffer contents change), but the default signal handler revalidates it to point to the end of the inserted text.

gtk.TextBuffer.insert_at_cursor

```
    def insert_at_cursor(text)
```

text :	some text in UTF−8 format

The insert_at_cursor() method is a convenience method that calls the insert() method, using the current cursor position as the insertion point.

gtk.TextBuffer.insert_interactive

```
    def insert_interactive(iter, text, default_editable)
```

iter :	a position in *buffer*
text :	some UTF−8 text
default_editable :	default editability of buffer
Returns :	TRUE if the text was actually inserted

The insert_interactive() method is similar to the insert() method, except the insertion of *text* at *iter* will not occur if *iter* is at a non−editable location in the buffer. A location is non−editable if a gtk.TextTag applied at that location has its "editable" attribute set to FALSE or the gtk.TextView used by the user is set non−editable. Usually you want to prevent insertions at locations if the insertion results from a user action (is interactive).

default_editable indicates the editability of text that doesn't have a tag affecting editability applied to it. Typically the result of the gtk.TextView.get_editable() method is appropriate here.

gtk.TextBuffer.insert_interactive_at_cursor

```
    def insert_interactive_at_cursor(text, default_editable)
```

text :	text in UTF−8 format
default_editable :	default editability of buffer
Returns :	TRUE if the text was actually inserted

The insert_interactive_at_cursor() method calls the insert_interactive() method to insert *text* at the cursor (insert) position. *default_editable* indicates the editability of text that doesn't have a tag affecting editability applied to it. Typically the result of the gtk.TextView.get_editable() method is appropriate here.

gtk.TextBuffer.insert_range

```
    def insert_range(iter, start, end)
```

iter :	a position in the textbuffer
start :	a position in a (possibly different) gtk.TextBuffer
end :	another position in the same buffer as *start*

The insert_range() method copies text, tags, and pixbufs (but not child anchors) between *start* and *end* (the order of *start* and *end* doesn't matter) form a gtk.TextBuffer and inserts the copy at *iter*. Used instead of simply getting/inserting text because it preserves images and tags. If *start* and *end* are in a different buffer from *buffer*, the two buffers must share the same tag table. This method is implemented via emissions of the "insert_text" and "apply_tag" signals.

gtk.TextBuffer.insert_range_interactive

```
    def insert_range_interactive(iter, start, end, default_editable)
```

`iter` :	a position in the textbuffer
`start` :	a position in a (possibly different) `gtk.TextBuffer`
`end` :	another position in the same buffer as *`start`*
`default_editable` :	default editability of the buffer
Returns :	`TRUE` if an insertion was possible at *`iter`*

The `insert_range_interactive()` method is similar to the `insert_range()` method, except the insertion of *`text`* at *`iter`* will not occur if the insertion position is non–editable. A location is non–editable if a `gtk.TextTag` applied at that location has its "editable" attribute set to `FALSE` or the `gtk.TextView` used by the user is set non–editable. The *`default_editable`* parameter indicates whether the text is editable at *`iter`* if no tags enclosing *`iter`* affect editability. Typically the result of the `gtk.TextView.get_editable()` method is appropriate here.

gtk.TextBuffer.insert_with_tags

```
    def insert_with_tags(iter, text, ...)
```

`iter` :	an iterator in *`buffer`*
`text` :	UTF–8 text
`...` :	one or more optional `gtk.TextTag` objects to apply to *`text`*

The `insert_with_tags()` method inserts the specified *`text`* into the textbuffer at the location specified by *`iter`*, applying any optional tags following the first two parameters to the newly–inserted text. This method is a convenience method that is equivalent to calling the `insert()` method, then the `apply_tag()` method on the inserted text.

gtk.TextBuffer.insert_with_tags_by_name

```
    def insert_with_tags_by_name(iter, text, ...)
```

`iter` :	position in *`buffer`*
`text` :	UTF–8 text
`...` :	one or more optional `gtk.TextTag` names to apply to *`text`*

The `insert_with_tags_by_name()` method is similar to the `insert_with_tags()` method, but uses tag names instead of tag objects.

gtk.TextBuffer.delete

```
    def delete(start, end)
```

`start` :	a position in the textbuffer
`end` :	another position in the textbuffer

The `delete()` method deletes the text between *`start`* and *`end`*. The order of *`start`* and *`end`* is not actually relevant as the `delete()` method will reorder them. This method emits the "delete_range" signal, and the default handler of that signal deletes the text. Because the textbuffer is modified, all outstanding iterators become invalid after calling this function; however, *`start`* and *`end`* will be re–initialized to point to the location where text was deleted.

gtk.TextBuffer.delete_interactive

```
    def delete_interactive(start_iter, end_iter, default_editable)
```

start_iter :	the start of the text to delete
end_iter :	the end of the text to delete
default_editable :	whether the buffer is editable by default
Returns :	TRUE if some text was actually deleted

The delete_interactive() method deletes all *editable* text in the given range. This method calls the delete() method for each editable sub–range of [*start*,*end*). *start* and *end* are revalidated to point to the location of the last deleted range, or left untouched if no text was deleted. A range of *text* is non–editable if a gtk.TextTag applied to that range has its "editable" attribute set to FALSE or the gtk.TextView used by the user is set non–editable. The *default_editable* parameter indicates whether *text* is editable if no tags enclosing any part of *text* affect editability. Typically the result of the gtk.TextView.get_editable() method is appropriate here.

gtk.TextBuffer.get_text

```
    def get_text(start, end, include_hidden_chars=TRUE)
```

start :	the start of a range
end :	the end of a range
include_hidden_chars :	if TRUE include invisible text
Returns :	the text in the range

The get_text() method returns the text in the specified range [*start*,*end*). Undisplayed text (text marked with tags that set the invisibility attribute) are excluded if *include_hidden_chars* is FALSE. get_text() does not return characters representing embedded images, so byte and character indexes into the returned string do *not* correspond to byte and character indexes into the buffer. Contrast this behavior with the get_slice() method.

gtk.TextBuffer.get_slice

```
    def get_slice(start, end, include_hidden_chars)
```

start :	the start of a range
end :	the end of a range
include_hidden_chars :	if TRUE include invisible text
Returns :	the contents of the range including text and indicators for pixbufs and child anchors

The get_slice() method returns the text in the range [*start*,*end*). Undisplayed text (text marked with tags that set the invisibility attribute) is excluded if *include_hidden_chars* is FALSE. The returned string includes a 0xFFFC character whenever the textbuffer contains embedded images or child anchors, so byte and character indexes into the returned string *do* correspond to byte and character indexes into the buffer. Contrast this behavior with the get_text() method. Note that 0xFFFC can occur in normal text as well, so it is not a reliable indicator that a pixbuf or widget is in the buffer.

gtk.TextBuffer.insert_pixbuf

```
    def insert_pixbuf(iter, pixbuf)
```

iter :	the location to insert the pixbuf

pixbuf :	a gtk.gdk.Pixbuf

The insert_pixbuf() method inserts an image specified by *pixbuf* into the text buffer at the location specified by *iter*. The image will be counted as one character in character counts, and when obtaining the buffer contents as a string, will be represented by the Unicode "object replacement character" 0xFFFC. Note that the "slice" variants for obtaining portions of the buffer as a string include this character for pixbufs, but the "text" variants do not. e.g. see the get_slice() and get_text() methods.

gtk.TextBuffer.insert_child_anchor

```
    def insert_child_anchor(iter, anchor)
```

iter :	the location to insert the anchor
anchor :	a gtk.TextChildAnchor

The insert_child_anchor() method inserts a child widget anchor specified by *anchor* into the textbuffer at the location specified by *iter*. The anchor will be counted as one character in character counts, and when obtaining the buffer contents as a string, will be represented by the Unicode "object replacement character" 0xFFFC. Note that the "slice" variants for obtaining portions of the buffer as a string include this character for child anchors, but the "text" variants do not. e.g. see the get_slice() and get_text() methods. The create_child_anchor() is a more convenient alternative to this function.

gtk.TextBuffer.create_child_anchor

```
    def create_child_anchor(iter)
```

iter :	a location in the buffer
Returns :	the new child anchor

The create_child_anchor() method is a convenience function that creates a child anchor with the gtk.TextChildAnchor() constructor and inserts it into the textbuffer at the location specified by *iter* with the insert_child_anchor() method.

gtk.TextBuffer.create_mark

```
    def create_mark(mark_name, where, left_gravity)
```

mark_name :	the name for the new mark, or None
where :	the location to place the mark
left_gravity :	if TRUE the mark has left gravity
Returns :	the new gtk.TextMark object

The create_mark() method creates a mark with the name specified by *mark_name* at the position specified by *where* and left gravity specified by *left_gravity*. If *mark_name* is None, the mark is anonymous; otherwise, the mark can be retrieved by name using the get_mark() method. If a mark has left gravity, and text is inserted at the mark's current location, the mark will be moved to the left of the newly–inserted text. If the mark has right gravity (*left_gravity* = FALSE), the mark will end up on the right of newly–inserted text. The standard left–to–right cursor is a mark with right gravity (when you type, the cursor stays on the right side of the text you're typing). Marks are owned by the buffer and go away when the buffer does. This method emits the "mark_set" signal as notification of the mark's initial placement.

gtk.TextBuffer.move_mark

```
    def move_mark(mark, where)
```

`mark` :	a `gtk.TextMark`
`where` :	a new location for *`mark`*

The `move_mark()` method moves the `gtk.TextMark` specified by *`mark`* to the new location specified by *`where`*.This method emits the "mark_set" signal as notification of the move.

gtk.TextBuffer.delete_mark

```
    def delete_mark(mark)
```

`mark` :	a `gtk.TextMark` in the textbuffer

The `delete_mark()` method deletes the `gtk.TextMark` specified by *`mark`*, so that it's no longer located anywhere in the textbuffer. Most operations on *`mark`* become invalid and there is no way to undelete a mark. The `get_deleted()` method will return `TRUE` after this method has been called on a mark to indicate that a mark no longer belongs to a textbuffer. The "mark_deleted" signal will be emitted as notification after the mark is deleted.

gtk.TextBuffer.get_mark

```
    def get_mark(name)
```

`name` :	a mark name
Returns :	a `gtk.TextMark`, or None

The get_mark() method returns the mark named *`name`* in the textbuffer, or `None` if no such mark exists in the buffer.

gtk.TextBuffer.move_mark_by_name

```
    def move_mark_by_name(name, where)
```

`name` :	the name of a mark
`where` :	the new location for mark

The `move_mark_by_name()` method moves the mark named *`name`* (which must exist) to the textbuffer location specified by *`where`*. See the `move_mark()` method for details.

gtk.TextBuffer.delete_mark_by_name

```
    def delete_mark_by_name(name)
```

`name` :	the name of a mark in *`buffer`*

The `delete_mark_by_name()` method deletes the mark (which must exist) named *`name`*. See the `delete_mark()` for details.

gtk.TextBuffer.get_insert

```
    def get_insert()
```

Returns :	the insertion point mark

The `get_insert()` method returns the mark that represents the cursor (insertion point). Equivalent to calling the `get_mark()` method to get the mark named "insert", but very slightly more efficient, and involving less typing.

gtk.TextBuffer.get_selection_bound

```
    def get_selection_bound()
```

Returns :	the selection bound mark

The `get_selection_bound`() method returns the mark that represents the selection bound. Equivalent to calling the `get_mark()` method to get the mark named "selection_bound", but very slightly more efficient, and involving less typing. The currently−selected text in a textbuffer is the region between the "selection_bound" and "insert" marks. If "selection_bound" and "insert" are in the same place, then there is no current selection. The `get_selection_bounds()` method is a convenience method for handling the selection, if you just want to know whether there's a selection and what its bounds are.

gtk.TextBuffer.place_cursor

```
    def place_cursor(where)
```

where :	where to put the cursor

The `place_cursor`() method moves the "insert" and "selection_bound" marks simultaneously to the location specified by *where*. If you move them to the same place in two steps with the `move_mark()` method, you will temporarily select a region in between their old and new locations, which is inefficient. This method moves them as a unit, which can be optimized.

gtk.TextBuffer.select_range

```
    def select_range(ins, bound)
```

ins :	where to put the "insert" mark
bound :	where to put the "selection_bound" mark

Note

This method is available in PyGTK 2.4 and above.

The `select_range`() method moves the "insert" and "selection_bound" marks simultaneously to the locations specified by *ins* and *bound* respectively. If you move them to the same place in two steps with the `move_mark()` method, you will temporarily select a region in between their old and new locations, which is inefficient. This method moves them as a unit, which can be optimized.

gtk.TextBuffer.apply_tag

```
    def apply_tag(tag, start, end)
```

tag :	a `gtk.TextTag`
start :	the start of the range
end :	the end of the range

The `apply_tag`() method emits the "apply−tag" signal that causes the `gtk.TextTag` specified by *tag* to be applied to the range of text between *start* and *end* by the default signal handler. *start* and *end* do not have to be in order.

gtk.TextBuffer.remove_tag

```
    def remove_tag(tag, start, end)
```

tag :	a gtk.TextTag
start :	the start of the range
end :	the end of the range

The `delete_tag`() method emits the "remove_tag" signal that causes the default handler for the signal to remove all occurrences of the gtk.TextTag specified by *tag* from the text in the range between *start* and *end*. *start* and *end* don't have to be in order.

gtk.TextBuffer.apply_tag_by_name

```
def apply_tag_by_name(name, start, end)
```

name :	the name of a gtk.TextTag
start :	the start of the range
end :	the end of the range

The `apply_tag_by_name`() method calls the gtk.TextTagTable.lookup() method on the textbuffer's tag table to find the gtk.TextTag with the specified *name*, then calls the apply_tag() method to apply that tag to the text in the range between *start* and *end*. *start* and *end* don't have to be in order.

gtk.TextBuffer.remove_tag_by_name

```
def remove_tag_by_name(name, start, end)
```

name :	the name of a gtk.TextTag
start :	the start of the range
end :	the end of the range

The `delete_tag_by_name`() method calls the gtk.TextTagTable.lookup() method on the textbuffer's tag table to find the gtk.TextTag, then calls the remove_tag() method to remove that that tag from the text in the range between *start* and *end*. *start* and *end* don't have to be in order.

gtk.TextBuffer.remove_all_tags

```
def remove_all_tags(start, end)
```

start :	the start of the range
end :	the end of the range

The `remove_all_tags`() method removes all tags in the text in the range between *start* and *end*. Be careful with this function; it could remove tags added in code unrelated to the code you're currently writing. That is, using this function is probably a bad idea if you have two or more unrelated code sections that add tags. *start* and *end* don't have to be in order.

gtk.TextBuffer.create_tag

```
def create_tag(tag_name=None, ...)
```

tag_name :	the name of the new tag, or `None` if the tag is anonymous
`...` :	one or more property_name= value pairs
Returns :	a new tag

The `create_tag`() method creates a tag with the name specified by *tag_name* and adds it to the tag table for the textbuffer. If one or more `property_name=value` pairs are available they are used to set the properties of the tag. Note the `property_name` must be specified using underscores instead of dashes e.g. use pixels_above_lines=10 instead of pixels−above−lines=10. This method is equivalent to calling the gtk.TextTag() constructor and then adding the tag to the buffer's tag table with the `gtk.TextTagTable.add()` method. If *`tag_name`* is `None`, the tag is anonymous. If *`tag_name`* is non−`None`, a tag called *`tag_name`* must not already exist in the tag table for this buffer.

gtk.TextBuffer.get_iter_at_line_offset

```
    def get_iter_at_line_offset(line_number, char_offset)
```

`line_number` :	the line number counting from 0
`char_offset` :	the char offset from start of line
Returns :	an iterator

The `get_iter_at_line_offset`() returns an iterator pointing to the position specified by *`char_offset`* within the line specified by *`line_number`*. The *`char_offset`* must exist, offsets off the end of the line are not allowed. Note specify *characters*, not bytes; UTF−8 may encode one character as multiple bytes.

gtk.TextBuffer.get_iter_at_line_index

```
    def get_iter_at_line_index(line_number, byte_index)
```

`line_number` :	the line number counting from 0
`byte_index` :	the byte index from start of line
Returns :	an iterator

The `get_iter_at_line_index`() method returns an iterator pointing to the position specified by *`byte_index`* within the line specified by *`line_number`*. *`byte_index`* must be the start of a UTF−8 character, and must not be beyond the end of the line. Note specify *bytes*, not characters; UTF−8 may encode one character as multiple bytes.

gtk.TextBuffer.get_iter_at_offset

```
    def get_iter_at_offset(char_offset)
```

`char_offset` :	the char offset from start of buffer, counting from 0
Returns :	an iterator

The `get_iter_at_offset`() method returns an iterator pointing to the location specified by *`char_offset`* characters from the start of the entire buffer.

gtk.TextBuffer.get_iter_at_line

```
    def get_iter_at_line(line_number)
```

`line_number` :	line number counting from 0
Returns :	an iterator

The `get_iter_at_line`() method returns an iterator pointing to the start of the line specified by *`line_number`*.

gtk.TextBuffer.get_start_iter

```
def get_start_iter()
```

Returns : an iterator

The `get_start_iter()` method returns an iterator pointing at the location of the first position in the text buffer. This is the same as using the `get_iter_at_offset()` with an argument of 0.

gtk.TextBuffer.get_end_iter

```
def get_end_iter()
```

Returns : an iterator

The `get_end_iter()` method returns an iterator pointing at the "end iterator," one past the last valid character in the text buffer. If passed to the `gtk.TextIter.get_char()` method, the end iterator has a character value of 0. The entire buffer lies in the range from the first position in the buffer (call the `get_start_iter()` method to get character position 0) to the end iterator.

gtk.TextBuffer.get_bounds

```
def get_bounds()
```

Returns : a tuple containing iterators that point at the first and last positions in the buffer

The `get_bounds()` method returns a tuple containing the first and last iterators in the buffer, i.e. the entire buffer lies within the range.

gtk.TextBuffer.get_iter_at_mark

```
def get_iter_at_mark(mark)
```

`mark` : a `gtk.TextMark` in the textbuffer

The `get_iter_at_mark()` method returns an iterator that points at the current position of *`mark`*.

gtk.TextBuffer.get_iter_at_child_anchor

```
def get_iter_at_child_anchor(iter, anchor)
```

`anchor` : a child anchor that appears in the textbuffer

The get_iter_at_child_anchor() method returns an iterator that points at the location of *`anchor`* within the textbuffer.

gtk.TextBuffer.get_modified

```
def get_modified()
```

Returns : `TRUE` if the buffer has been modified

The `get_modified()` method returns `TRUE` if the buffer has been modified since the last call to the `set_modified()` method set the modification flag to `FALSE`. Used for example to enable a "save" function in a text editor.

gtk.TextBuffer.set_modified

```
def set_modified(setting)
```

setting :	the modification flag setting

The `set_modified()` method sets the modification flag of the textbuffer to the value specified by *`setting`*. The modification flag is used to keep track of whether the buffer has been modified since the last time it was saved. Whenever the buffer is saved to disk, call this method with a *`setting`* of `FALSE`. When the buffer is modified, it will automatically set the modification flag to *`TRUE`* and emit a "modified_changed" signal.

gtk.TextBuffer.add_selection_clipboard

```
def add_selection_clipboard(clipboard)
```

clipboard :	a `gtk.Clipboard`

Note

This method is available in PyGTK 2.2 and above.

The `add_selection_clipboard()` method adds the `gtk.Clipboard` specified by *`clipboard`* to the list of clipboards in which the selection contents of the buffer are available. In most cases, *`clipboard`* will be the `gtk.gdk.SELECTION_PRIMARY` clipboard

gtk.TextBuffer.remove_selection_clipboard

```
def remove_selection_clipboard(clipboard)
```

clipboard :	a `gtk.Clipboard` added to the buffer by the `add_selection_clipboard()` method.

Note

This method is available in PyGTK 2.2 and above.

The `remove_selection_clipboard()` method removes the `gtk.Clipboard` added with the `add_selection_clipboard()` method.

gtk.TextBuffer.cut_clipboard

```
def cut_clipboard(clipboard, default_editable)
```

clipboard :	the `gtk.Clipboard` object to cut to.
default_editable :	the default editability of the buffer

Note

This method is available in PyGTK 2.2 and above.

The `cut_clipboard()` method copies the currently−selected text to the `gtk.Clipboard` specified by *`clipboard`*, then deletes said text if it's editable as specified by *`default_editable`*. Typically the result of the `gtk.TextView.get_editable()` method is appropriate here.

gtk.TextBuffer.copy_clipboard

```
    def copy_clipboard(clipboard)
```

clipboard :	the gtk.Clipboard object to copy to.

Note

This method is available in PyGTK 2.2 and above.

The `copy_clipboard`() method copies the currently−selected text to the gtk.ClipBoard specified by *clipboard*.

gtk.TextBuffer.paste_clipboard

```
    def paste_clipboard(clipboard, override_location, default_editable)
```

clipboard :	the gtk.Clipboard to paste from
override_location :	the gtk.TextIter specifying the location to insert pasted text, or `None` for at the cursor
default_editable :	the default editability of the buffer

Note

This method is available in PyGTK 2.2 and above.

The `paste_clipboard`() method pastes the contents of the gtk.ClipBoard specified by *clipboard* at the insertion point, or at the location specified by *override_location* (if not `None`). (Note: pasting is asynchronous, that is, we'll ask for the paste data and return, and at some point later after the main loop runs, the paste data will be inserted.)

gtk.TextBuffer.get_selection_bounds

```
    def get_selection_bounds()
```

Returns :	a tuple containing iterators pointing to the selection start and end or an empty tuple if there is no selection

The `get_selection_bounds`() method returns a tuple containing iterators that point at the start and end of the selection, if any. If there is no selection an empty tuple is returned.

gtk.TextBuffer.delete_selection

```
    def delete_selection(interactive, default_editable)
```

interactive :	if `TRUE` the deletion is caused by user interaction
default_editable :	if *`TRUE`* the buffer is editable by default
Returns :	`TRUE` if there was a non−empty selection to delete

The `delete_selection`() method deletes the text in the range between the "insert" and "selection_bound" marks, i.e. the currently−selected text. If *interactive* is `TRUE`, the editability of the selection will be considered (users can't delete uneditable text) and default_editable is used to determine the default editability of the textbuffer usually as a result of a call to the gtk.TextView.get_editable() method.

gtk.TextBuffer.begin_user_action

```
    def begin_user_action()
```

The `begin_user_action`() method is called to indicate that the textbuffer operations until a call to the `end_user_action()` method are part of a single user–visible operation. The operations between the `begin_user_action`() and `end_user_action()` methods can then be grouped when creating an undo stack. `gtk.TextBuffer` maintains a count of calls to the `begin_user_action`() method that have not been closed with a call to the `end_user_action()` method, and emits the "begin_user_action" and "end_user_action" signals only for the outermost pair of calls. This allows you to chain user actions.

The "interactive" textbuffer mutation methods, such as the `insert_interactive()` method, automatically call the begin and end user action methods around the textbuffer operations they perform, so there's no need to add extra calls if you user action consists solely of a single call to one of those methods.

gtk.TextBuffer.end_user_action

```
    def end_user_action()
```

The `end_user_action`() method should be paired with a call to the `begin_user_action()` method.

gtk.TextBuffer.backspace

```
    def backspace(iter, interactive, default_editable)
```

`iter` :	a `gtk.TextIter`
`interactive` :	if `TRUE` the deletion is caused by user interaction
`default_editable` :	if *`TRUE`* the buffer is editable by default
Returns :	`TRUE` if the buffer was modified

Note

This method is available in PyGTK 2.6 and above.

The `backspace`() method performs the appropriate action as if the user hit the delete key with the cursor at the position specified by *`iter`*. In the normal case a single character will be deleted, but when combining accents are involved, more than one character can be deleted, and when precomposed character and accent combinations are involved, less than one character will be deleted.

Because the buffer is modified, all outstanding iterators become invalid after calling this function; however, *`iter`* will be re–initialized to point to the location where text was deleted.

Signals

The "apply–tag" gtk.TextBuffer Signal

```
    def callback(textbuffer, texttag, start, end, user_param1, ...)
```

`textbuffer` :	the textbuffer that received the signal
`texttag` :	the `gtk.TextTag` being applied
`start` :	an iterator pointing to the start of the range of text
`end` :	an iterator pointing to the end of the range of text

user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "apply−tag" signal is emitted when *texttag* is applied to the text in *textbuffer* in the range specified by *start* and *end*.

The "begin−user−action" gtk.TextBuffer Signal

```
def callback(textbuffer, user_param1, ...)
```

textbuffer :	the textbuffer that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "begin−user−action" signal is emitted on the first call to the begin_user_action() method.

The "changed" gtk.TextBuffer Signal

```
def callback(textbuffer, user_param1, ...)
```

textbuffer :	the textbuffer that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "changed" signal is emitted when text is inserted in *textbuffer*.

The "delete−range" gtk.TextBuffer Signal

```
def callback(textbuffer, start, end, user_param1, ...)
```

textbuffer :	the textbuffer that received the signal
start :	an iterator pointing to the start of the range of text
end :	an iterator pointing to the end of the range of text
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "delete−range" signal is emitted when the text in the range specified by *start* and *end* is to be deleted.

The "end−user−action" gtk.TextBuffer Signal

```
def callback(textbuffer, user_param1, ...)
```

textbuffer :	the textbuffer that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "end−user−action" signal is emitted when the call to the end_user_action() method reduces the user action count to zero i.e. undoes the first call to the begin_user_action() method.

The "insert−child−anchor" gtk.TextBuffer Signal

```
def callback(textbuffer, iter, anchor, user_param1, ...)
```

textbuffer :	the textbuffer that received the signal

iter :	a gtk.TextIter
anchor :	a gtk.TextChildAnchor
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "insert−child−anchor" signal is emitted when *anchor* is inserted into *textbuffer* at the location specified by *iter*.

The "insert−pixbuf" gtk.TextBuffer Signal

```
    def callback(textbuffer, iter, pixbuf, user_param1, ...)
```

textbuffer :	the textbuffer that received the signal
iter :	a gtk.TextIter
pixbuf :	a gtk.gdk.Pixbuf
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "insert−pixbuf" signal is emitted when *pixbuf* is inserted into *textbuffer* at the location specified by *iter*.

The "insert−text" gtk.TextBuffer Signal

```
    def callback(textbuffer, iter, text, length, user_param1, ...)
```

textbuffer :	the textbuffer that received the signal
iter :	a gtk.TextIter
text :	the text inserted in *textbuffer*
length :	the length of the text inserted in *textbuffer*
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "insert−text" signal is emitted when *text* of the size specified by *length* is inserted into *textbuffer* at the location specified by *iter*.

The "mark−deleted" gtk.TextBuffer Signal

```
    def callback(textbuffer, textmark, user_param1, ...)
```

textbuffer :	the textbuffer that received the signal
textmark :	the gtk.TextMark that is being deleted
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "mark−deleted" signal is emitted when *textmark* is being deleted from *textbuffer*.

The "mark−set" gtk.TextBuffer Signal

```
    def callback(textbuffer, user_param1, ...)
```

textbuffer :	the textbuffer that received the signal
iter :	an iterator pointing at the location where *textmark* will be set.
textmark :	the gtk.TextMark that is being set

user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "mark−set" signal is emitted when *textmark* is being set at the location specified by *iter* in *textbuffer*.

The "modified−changed" gtk.TextBuffer Signal

```
    def callback(textbuffer, user_param1, ...)
```

textbuffer :	the textbuffer that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "modified−changed" signal is emitted when the modification flag is changed.

The "remove−tag" gtk.TextBuffer Signal

```
    def callback(textbuffer, texttag, start, end, user_param1, ...)
```

textbuffer :	the textbuffer that received the signal
texttag :	the gtk.TextTag being removed
start :	an iterator pointing to the start of the range of text
end :	an iterator pointing to the end of the range of text
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "remove−tag" signal is emitted when *texttag* is being removed from the *textbuffer* text in the range specified by *start* and *end*.

Prev	Up	Next
gtk.TextAttributes	Home	gtk.TextChildAnchor

	gtk.TextChildAnchor	
Prev	**The gtk Class Reference**	Next

gtk.TextChildAnchor

gtk.TextChildAnchor   a location in a textbuffer for placing widgets

Synopsis

```
class gtk.TextChildAnchor(gobject.GObject):
    gtk.TextChildAnchor()
    def get_widgets()
    def get_deleted()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.TextChildAnchor
```

Description

A gtk.TextChildAnchor provides a location in a gtk.TextBuffer for placing child widgets in a gtk.TextView. Since a gtk.TextBuffer can be associated with more than one gtk.TextView a gtk.TextChildAnchor can have a different child widget inserted for each textview it is displayed in.

Constructor

```
gtk.TextChildAnchor()
```

Returns :	a new gtk.TextChildAnchor object

Creates a new gtk.TextChildAnchor. Usually you would then insert it into a gtk.TextBuffer with the gtk.TextBuffer.insert_child_anchor() method. To perform the creation and insertion in one step, use the convenience method gtk.TextBuffer.create_child_anchor() method.

Methods

gtk.TextChildAnchor.get_widgets

```
def get_widgets()
```

Returns :	a list of widgets anchored at the child anchor

The `get_widgets`() method returns a list of all widgets anchored at this child anchor from all the associated textviews.

gtk.TextChildAnchor.get_deleted

```
def get_deleted()
```

Returns :	`TRUE` if the child anchor has been deleted from its textbuffer

The `get_deleted`() method returns `TRUE` if the child anchor has been deleted from its textbuffer.

Prev Up Next
gtk.TextBuffer Home gtk.TextIter

gtk.TextIter
Prev **The gtk Class Reference** Next

gtk.TextIter

gtk.TextIter — an object pointing at a location in a gtk.TextBuffer

Synopsis

```
class gtk.TextIter(gobject.GBoxed):
    def get_buffer()
    def copy()
    def get_offset()
    def get_line()
    def get_line_offset()
    def get_line_index()
    def get_visible_line_offset()
    def get_visible_line_index()
    def get_char()
    def get_slice(end)
    def get_text(end)
    def get_visible_slice(end)
    def get_visible_text(end)
    def get_pixbuf()
    def get_marks()
    def get_child_anchor()
    def get_toggled_tags(toggled_on)
    def begins_tag(tag=None)
    def ends_tag(tag=None)
    def toggles_tag(tag=None)
    def has_tag(tag)
    def get_tags()
    def editable(default_setting)
    def can_insert(default_editability)
    def starts_word()
    def ends_word()
    def inside_word()
    def starts_sentence()
    def ends_sentence()
    def inside_sentence()
    def starts_line()
    def ends_line()
    def is_cursor_position()
    def get_chars_in_line()
    def get_bytes_in_line()
    def get_attributes(values)
    def get_language()
    def is_end()
    def is_start()
    def forward_char()
    def backward_char()
    def forward_chars(count)
    def backward_chars(count)
    def forward_line()
    def backward_line()
    def forward_lines(count)
    def backward_lines(count)
    def forward_word_end()
    def backward_word_start()
    def forward_word_ends(count)
    def backward_word_starts(count)
    def forward_visible_word_end()
    def backward_visible_word_start()
    def forward_visible_word_ends(count)
    def backward_visible_word_starts(count)
    def forward_sentence_end()
    def backward_sentence_start()
    def forward_sentence_ends(count)
    def backward_sentence_starts(count)
    def forward_cursor_position()
    def backward_cursor_position()
    def forward_cursor_positions(count)
    def backward_cursor_positions(count)
    def set_offset(char_offset)
    def set_line(line_number)
    def set_line_offset(char_on_line)
    def set_line_index(byte_on_line)
    def forward_to_end()
    def forward_to_line_end()
    def set_visible_line_offset(char_on_line)
    def set_visible_line_index(byte_on_line)
    def forward_to_tag_toggle(tag)
    def backward_to_tag_toggle(tag)
    def forward_find_char(pred, user_data, limit)
    def backward_find_char(pred, user_data, limit)
    def forward_search(str, flags, limit=None)
    def backward_search(str, flags, limit=None)
    def equal(rhs)
    def compare(rhs)
    def in_range(start, end)
    def order(second)
```

Description

A `gtk.TextIter` points to a position between two characters in a `gtk.TextBuffer`. A `gtk.TextIter` is usually created using a `gtk.TextBuffer` method and are invalidated when the number of characters in the `gtk.TextBuffer` changes (with some exceptions when inserting or deleting) including inserting or deleting pixbufs or child anchors. There are a huge number of `gtk.TextIter` methods mostly dealing with moving the textiter location in the textbuffer, checking the location or retrieving text or objects at a location.

Methods

gtk.TextIter.get_buffer

```
  def get_buffer()
```

Returns :	the textbuffer

The `get_buffer()` method returns the `gtk.TextBuffer` object this iterator is associated with.

gtk.TextIter.copy

```
  def copy()
```

Returns :	a copy of the textiter

The `copy()` method creates a copy of the textiter.

gtk.TextIter.get_offset

```
  def get_offset()
```

Returns :	a character offset

The `get_offset()` method returns the character offset of the textiter. Each character in a `gtk.TextBuffer` has an offset, starting with 0 for the first character in the textbuffer. Use `gtk.TextBuffer.get_iter_at_offset()` to convert an offset back into a textiter.

gtk.TextIter.get_line

```
  def get_line()
```

Returns :	a line number

The `get_line()` method returns the line number containing the textiter. Lines in a `gtk.TextBuffer` are numbered beginning with 0 for the first line.

gtk.TextIter.get_line_offset

```
def get_line_offset()
```

Returns :	the offset from the start of the line

The `get_line_offset()` method returns the character offset of the textiter location, counting from the start of the line containing the textiter location. The first character on the line has offset 0.

gtk.TextIter.get_line_index

```
def get_line_index()
```

Returns :	the number of bytes from the start of the line

The `get_line_offset()` method returns the byte index of the textiter location, counting from the start of the line containing the textiter location. Remember that `gtk.TextBuffer` encodes text in UTF–8, and that characters can require a variable number of bytes to represent.

gtk.TextIter.get_visible_line_offset

```
def get_visible_line_offset()
```

Returns :	the offset in visible characters from the start of the line

The `get_visible_line_offset()` method returns the offset in characters of the textiter location from the start of the line containing the textiter location, not counting characters that are invisible due to tags with the "invisible" attribute set.

gtk.TextIter.get_visible_line_index

```
def get_visible_line_index()
```

Returns :	a byte index from the start of the line

The `get_visible_line_index()` method returns the byte index of the textiter location from the start of the line, not counting bytes that are invisible due to tags with the "invisible" attribute set.

gtk.TextIter.get_char

```
def get_char()
```

Returns :	a Unicode character, or 0 if the textiter is not dereferenceable

The `get_char()` method returns the Unicode character at this textiter location. If the textiter points at a non–character element, such as an image embedded in the buffer, the Unicode "unknown" character 0xFFFC is returned. If invoked on the end textiter, zero is returned; zero is not a valid Unicode character. So you can write a loop which ends when the `get_char()` method returns 0.

gtk.TextIter.get_slice

```
def get_slice(end)
```

`end` :	the textiter at the end of a range

Returns :	a slice of text from the textbuffer

The `get_slice()` method returns the text in the range between the locations specified by the textiter and *end*. A "slice" is an array of characters encoded in UTF−8 format, including the Unicode "unknown" character 0xFFFC for iterable non−character elements in the textbuffer, such as images. Because images are encoded in the slice, byte and character offsets in the returned array will correspond to byte offsets in the textbuffer. Note that 0xFFFC can occur in normal text as well, so it is not a reliable indicator that a pixbuf or widget is in the textbuffer.

gtk.TextIter.get_text

```
def get_text(end)
```

end :	textiter at end of a range
Returns :	array of characters from the buffer

The `get_text()` method returns the text in the range between the locations specified by the textiter and *end*. If the range contains non−text elements such as images, the character and byte offsets in the returned string will not correspond to character and byte offsets in the textbuffer. If you want the offsets to correspond, use the `get_slice()` method.

gtk.TextIter.get_visible_slice

```
def get_visible_slice(end)
```

end :	textiter at end of range
Returns :	a slice of visible text from the textbuffer

The `get_visible_slice()` method is similar to the `get_slice()` method, but invisible text is excluded. Invisible text is text with the "invisible" attribute set on it.

gtk.TextIter.get_visible_text

```
def get_visible_text(end)
```

end :	textiter at end of range
Returns :	a string containing visible text from the textbuffer

The `get_visible_text()` method is similar to the `get_text()`, but invisible text is excluded. Invisible text is text with the "invisible" attribute set on it.

gtk.TextIter.get_pixbuf

```
def get_pixbuf()
```

Returns :	a pixbuf or `None`

The `get_pixbuf()` method returns the `gtk.gdk.Pixbuf` object at the textiter location, if any; otherwise, `None` is returned.

gtk.TextIter.get_marks

```
def get_marks()
```

Returns :	a list of `gtk.TextMark` objects

The `get_marks()` method returns a list of all `gtk.TextMark` objects at the textiter location. Because marks don't take up any "space" in the buffer, multiple marks can exist in the same location. The returned list is not in any meaningful order.

gtk.TextIter.get_child_anchor

```
    def get_child_anchor()
```

Returns :	a child anchor or `None`

The `get_child_anchor()` method returns the `gtk.TextChildAnchor` at the textiter location, if any; otherwise, `None` is returned.

gtk.TextIter.get_toggled_tags

```
    def get_toggled_tags(toggled_on)
```

`toggled_on` :	if `TRUE` get toggled–on tags; otherwise get toggle–off tags
Returns :	a list of tags toggled at this point

The `get_toggled_tags()` method returns a list of `gtk.TextTag` objects that are toggled on or off at this point. If *`toggled_on`* is `TRUE`, the list contains tags that are toggled on. If a tag is toggled on at the textiter location, some non–empty range of characters following the textiter has that tag applied to it. If a tag is toggled off, then some non–empty range following the textiter location does not have the tag applied to it.

gtk.TextIter.begins_tag

```
    def begins_tag(tag=None)
```

`tag` :	a `gtk.TextTag`, or `None`
Returns :	`TRUE` if the textiter is the start of a range tagged with *`tag`*

The `begins_tag()` method returns `TRUE` if *`tag`* is toggled on at exactly this point. If *`tag`* is `None`, this method returns `TRUE` if any tag is toggled on at this point. Note that the `begins_tag()` method returns `TRUE` only if the textiter location is the *start* of the tagged range; the `has_tag()` indicates if a textiter location is *within* a tagged range.

gtk.TextIter.ends_tag

```
    def ends_tag(tag=None)
```

`tag` :	a `gtk.TextTag`, or `None`
Returns :	`TRUE` if the textiter is the end of a range tagged with *`tag`*

The `ends_tag()` method returns `TRUE` if *`tag`* is toggled off at the location the textiter points to. If *`tag`* is `None`, this method returns `TRUE` if any tag is toggled off at this point. Note that the `ends_tag()` returns `TRUE` only if the textiter location is the *end* of the tagged range; the `has_tag()` indicates if a textiter location is *within* a tagged range.

gtk.TextIter.toggles_tag

```
    def toggles_tag(tag=None)
```

`tag` :	a `gtk.TextTag`, or `None`
Returns :	`TRUE` if *`tag`* is toggled on or off at the textiter location

The `toggles_tag`() method returns `TRUE` if a range of text with *`tag`* applied to it begins *or* ends at the textiter location. If *`tag`* is `None` this method returns `TRUE` if any tag begins or ends at the textiter location.

gtk.TextIter.has_tag

```
def has_tag(tag)
```

`tag` :	a `gtk.TextTag`
Returns :	`TRUE` if the textiter location is tagged with *`tag`*

The `has_tag`() method returns `TRUE` if the textiter location is within a range of text tagged with *`tag`*.

gtk.TextIter.get_tags

```
def get_tags()
```

Returns :	a list of `gtk.TextTag` objects

The `get_tags`() method returns a list of tags that apply to the textiter location, in ascending order of priority (highest–priority tags are last).

gtk.TextIter.editable

```
def editable(default_setting)
```

`default_setting` :	if `TRUE` the text is editable by default
Returns :	`TRUE` if the textiter location is inside an editable range or text

The `editable`() method returns `TRUE` if the character at the textiter location is within an editable range of text. Non–editable text is "locked" and can't be changed by the user via a `gtk.TextView`. This method is a convenience wrapper around the `get_attributes()` method. If no tags applied to this text location affect editability, the value of *`default_setting`* will be returned.

Do not use this method to determine if text can be inserted at the textiter location. For insertion you don't want to know if the char at the textiter location is inside an editable range of text, you want to know whether a new character inserted at the textiter location would be inside an editable range of text. Use the `can_insert()` method to determine if text can be inserted.

gtk.TextIter.can_insert

```
def can_insert(default_editability)
```

`default_editability` :	if `TRUE` the text is editable by default
Returns :	`TRUE` if text inserted at *`iter`* would be editable

The `can_insert()` method considers the default editability of the buffer, and the tags that affect editability, to determine if text inserted at the textiter location would be editable. If so, the user should be allowed to insert text at the textiter location. The `gtk.TextBuffer.insert_interactive()` uses this function to determine if insertions are allowed at a given position.

gtk.TextIter.starts_word

```
def starts_word()
```

Returns :	`TRUE` if the textiter location is at the start of a word

The `starts_word`() method returns `TRUE` if the textiter location begins a natural–language word. Word breaks are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms).

gtk.TextIter.ends_word

```
def ends_word()
```

Returns : `TRUE` if the textiter location is at the end of a word

The `ends_word`() method returns `TRUE` if the textiter location ends a natural–language word. Word breaks are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms).

gtk.TextIter.inside_word

```
def inside_word()
```

Returns : `TRUE` if the textiter location is inside a word

The `inside_word`() method returns `TRUE` if the textiter location is inside a natural–language word (as opposed to say inside some whitespace). Word breaks are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms).

gtk.TextIter.starts_sentence

```
def starts_sentence()
```

Returns : `TRUE` if the textiter location is at the start of a sentence.

The `starts_sentence`() method returns `TRUE` if the textiter location begins a sentence. Sentence boundaries are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango text boundary algorithms).

gtk.TextIter.ends_sentence

```
def ends_sentence()
```

Returns : `TRUE` if the textiter location is at the end of a sentence.

The `ends_sentence`() method returns `TRUE` if the textiter location ends a sentence. Sentence boundaries are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango text boundary algorithms).

gtk.TextIter.inside_sentence

```
def inside_sentence()
```

Returns : `TRUE` if the textiter location is inside a sentence.

The `inside_sentence`() method returns `TRUE` if the textiter location is inside a sentence (as opposed to in between two sentences, e.g. after a period and before the first letter of the next sentence). Sentence boundaries are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango text boundary algorithms).

gtk.TextIter.starts_line

```
def starts_line()
```

Returns :	`TRUE` if the textiter location begins a line

The `starts_line`() method returns `TRUE` if the textiter location begins a paragraph, i.e. if the `.get_line_offset()` method would return 0.

gtk.TextIter.ends_line

```
def ends_line()
```

Returns :	`TRUE` if the textiter location is at the end of a line

The `ends_line`() method returns `TRUE` if the textiter location points to the start of the paragraph delimiter characters for a line (delimiters will be either a newline, a carriage return, a carriage return followed by a newline, or a Unicode paragraph separator character). Note that an textiter pointing to the \n of a \r\n pair will not be counted as the end of a line, the line ends before the \r. The end textiter is considered to be at the end of a line, even though there are no paragraph delimiter chars there.

gtk.TextIter.is_cursor_position

```
def is_cursor_position()
```

Returns :	`TRUE` if the cursor can be placed at the textiter location

The `is_cursor_position`() method returns `TRUE` if the cursor can be placed at the textiter location. See the `forward_cursor_position()` method for details on what a cursor position is.

gtk.TextIter.get_chars_in_line

```
def get_chars_in_line()
```

Returns :	the number of characters in the line

The `get_chars_in_line`() method returns the number of characters in the line containing the textiter location, including the paragraph delimiters.

gtk.TextIter.get_bytes_in_line

```
def get_bytes_in_line()
```

Returns :	the number of bytes in the line

The `get_bytes_in_line`() method returns the number of bytes in the line containing the textiter location, including the paragraph delimiters.

gtk.TextIter.get_attributes

```
def get_attributes(values)
```

`values` :	a `gtk.TextAttributes` object to be filled in
Returns :	`TRUE` if *`values`* was modified

The `get_attributes`() method computes the effect of any tags applied to the textiter location and applies those attributes to the `gtk.TextAttributes` object specified by *`values`* (which should be initialized to the default settings you wish to use if no tags are in effect). Typically the default attributes are obtained from the `gtk.TextView.get_default_attributes()` method. If any tags affected *`values`*, the method returns `TRUE`.

gtk.TextIter.get_language

```
def get_language()
```

Returns :	the pango language in effect at the textiter location

The `get_language()` method is a convenience wrapper around the `get_attributes()` method, that returns the language in effect at the textiter location. If no tags affecting language apply to the textiter location, the return value is identical to that of the `gtk.get_default_language()` function.

gtk.TextIter.is_end

```
def is_end()
```

Returns :	`TRUE` if the textiter is the end textiter

The `is_end()` method returns `TRUE` if the textiter is the end textiter, i.e. one past the last dereferenceable textiter in the buffer. The `is_end()` method is the most efficient way to check whether an textiter is the end textiter.

gtk.TextIter.is_start

```
def is_start()
```

Returns :	`TRUE` if the textiter location is at the start of the textbuffer

The `is_start()` method returns `TRUE` if the textiter location is at the start of the textbuffer, that is if the textiter location has a character offset of 0.

gtk.TextIter.forward_char

```
def forward_char()
```

Returns :	`TRUE` if the textiter location moved and is dereferenceable

The `forward_char()` method moves the textiter location forward by one character offset and returns `TRUE` if the textiter location moved and the new location is dereferenceable. Note that images embedded in the buffer occupy 1 character slot, so the `forward_char()` method may actually move onto an image instead of a character, if you have images in your buffer. If the textiter location is the end textiter or one character before it, the textiter location will now point at the end textiter, and the `forward_char()` method returns `FALSE`.

gtk.TextIter.backward_char

```
def backward_char()
```

Returns :	TRUE if the textiter location moved and is not the start textiter

The `backward_char()` method moves the textiter location backward by one character offset and returns `TRUE` if the textiter location moved. If the old textiter location was the first in the buffer (character offset 0), the `backward_char()` method returns `FALSE`.

gtk.TextIter.forward_chars

```
def forward_chars(count)
```

count :	the number of characters to move, may be negative
Returns :	TRUE if the textiter location moved and is dereferenceable

The `forward_chars`() method moves the textiter location forward *count* characters if possible. If the textiter location would move past the start or end of the buffer, the location moves to the start or end of the textbuffer. The `forward_chars`() method returns `TRUE` if the new position of the resulting textiter location is different from its original position, and is dereferenceable (the last textiter in the buffer is not dereferenceable). If *count* is 0, the function does nothing and returns `FALSE`.

gtk.TextIter.backward_chars

```
def backward_chars(count)
```

count :	the number of characters to move, may be negative
Returns :	TRUE if the textiter location moved and is dereferenceable

The `backward_chars`() method moves the textiter location backward forward *count* characters, if possible. If the textiter location would move past the start or end of the buffer, the location moves to the start or end of the textbuffer. The `backward_chars`() method returns `TRUE` if the new position of the resulting textiter location is different from its original position, and is dereferenceable (the last textiter in the buffer is not dereferenceable). If *count* is 0, the function does nothing and returns `FALSE`.

gtk.TextIter.forward_line

```
def forward_line()
```

Returns :	TRUE if the textiter location can be dereferenced

The `forward_line`() method moves the textiter location to the start of the next line and returns `TRUE` if the textiter location moved to a dereferenceable position, and `FALSE` if the textiter location moved to the end of the buffer, or if the textiter location was originally at the end of the buffer.

gtk.TextIter.backward_line

```
def backward_line()
```

Returns :	TRUE if the textiter location moved

The `backward_line`() method moves the textiter location to the start of the previous line and returns `TRUE` if the textiter location was moved. If the textiter location was at the textbuffer start, this method returns `FALSE`. For example if the textiter location was already on line 0, but not at the start of the line, the textiter location is snapped to the start of the line and the method returns `TRUE`.

gtk.TextIter.forward_lines

```
def forward_lines(count)
```

count :	the number of lines to move forward, may be negative
Returns :	TRUE if the textiter location moved and is dereferenceable

The `forward_lines`() method moves the textiter location forward *count* lines, if possible. If the textiter location would move past the start or end of the buffer, the location moves to the start or end of the textbuffer. The method returns:

- TRUE if the textiter moved to a dereferenceable position; or,
- FALSE if the textiter location didn't move, or moved onto the end textiter or if *count* was 0.

If *count* is negative, the textiter location moves backward by *count* lines.

gtk.TextIter.backward_lines

```
def backward_lines(count)
```

count :	the number of lines to move backward, may be negative
Returns :	TRUE if the textiter location moved to a dereferenceable position

The backward_lines() method moves the textiter location backward by *count* lines, if possible. If the textiter location would move past the start or end of the buffer, the location moves to the start or end of the textbuffer. The method returns:

- TRUE if the textiter moved to a dereferenceable position; or,
- FALSE if the textiter location didn't move, or moved onto the end textiter or if *count* was 0.

If *count* is negative, the textiter location moves forward by *count* lines.

gtk.TextIter.forward_word_end

```
def forward_word_end()
```

Returns :	TRUE if the textiter location moved to a dereferenceable position

The forward_word_end() method moves the textiter location forward to the next word end. If the textiter location is currently on a word end, the location moves forward to the next one after that. Word breaks are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms). The method returns TRUE if the textiter location moved to a dereferenceable position

gtk.TextIter.backward_word_start

```
def backward_word_start()
```

Returns :	TRUE if the textiter location moved

The backward_word_start() method moves the textiter location backward to the previous word start. If the textiter location is currently on a word start, the location moves backward to the next one before that. Word breaks are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms). The method returns TRUE if the textiter location moved.

gtk.TextIter.forward_word_ends

```
def forward_word_ends(count)
```

count :	the number of times to move
Returns :	TRUE if the textiter location moved and is not the end textiter

The forward_word_ends() method calls the forward_word_end() method up to *count* times or the backward_word_starts() method if *count* is negative. The method returns TRUE if the textiter location changed and the resulting location is not at the end of the textbuffer.

gtk.TextIter.backward_word_starts

```
    def backward_word_starts(count)
```

count :	the number of times to move
Returns :	TRUE if the textiter location moved and is not the end textiter

The `backward_word_starts()` method calls the `backward_word_start()` method up to *count* times or the `forward_word_ends()` method if *count* is negative. The method returns `TRUE` if the textiter location changed and the resulting location is not at the end of the textbuffer.

gtk.TextIter.forward_visible_word_end

```
    def forward_visible_word_end()
```

Returns :	TRUE if the textiter moved and is not the end iterator

Note

This method is available in PyGTK 2.4 and above.

The `forward_visible_word_end()` method moves the textiter forward to the next visible word end. (If the textiter is currently on a word end, it moves forward to the next one after that.) Word breaks are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms).

gtk.TextIter.backward_visible_word_start

```
    def backward_visible_word_start()
```

Returns :	TRUE if the textiter moved and is not the end iterator

Note

This method is available in PyGTK 2.4 and above.

The `backward_visible_word_start()` method moves the textiter backward to the previous visible word start. (If textiter is currently on a word start, it moves backward to the next one after that.) Word breaks are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms).

gtk.TextIter.forward_visible_word_ends

```
    def forward_visible_word_ends(count)
```

count :	the number of times to move
Returns :	TRUE if the textiter moved and is not the end iterator

Note

This method is available in PyGTK 2.4 and above.

The `forward_visible_word_ends()` method calls the `forward_visible_word_end()` method the number of times specified by *count*.

gtk.TextIter.backward_visible_word_starts

```
    def backward_visible_word_starts(count)
```

count :	the number of times to move
Returns :	`TRUE` if the textiter moved and is not the end iterator

Note

This method is available in PyGTK 2.4 and above.

The `backward_visible_word_starts`() method calls the `backward_visible_word_start()` method the number of times specified by *`count`*.

gtk.TextIter.forward_sentence_end

```
    def forward_sentence_end()
```

Returns :	`TRUE` if the textiter location moved and is not the end textiter

The `forward_sentence_end`() method moves the textiter location forward to the next sentence end. (If the textiter location is at the end of a sentence, the location moves to the next end of sentence.) Sentence boundaries are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango text boundary algorithms). The method returns `TRUE` if the textiter location changed and the resulting location is not at the end of the textbuffer.

gtk.TextIter.backward_sentence_start

```
    def backward_sentence_start()
```

Returns :	`TRUE` if the textiter location moved

The `backward_sentence_start`() method moves the textiter location backward to the previous sentence start. If the textiter location is already at the start of a sentence, the location moves backward to the next one. Sentence boundaries are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango text boundary algorithms).

gtk.TextIter.forward_sentence_ends

```
    def forward_sentence_ends(count)
```

count :	the number of sentences to move
Returns :	`TRUE` if the textiter location moved and is not the end textiter

The `forward_sentence_ends`() method calls the `forward_sentence_end()` method *`count`* times (or until the `forward_sentence_end()` method returns `FALSE`). If *`count`* is negative, the location moves backward instead of forward. The method returns `TRUE` if the textiter location changed and the resulting location is not at the end of the textbuffer.

gtk.TextIter.backward_sentence_starts

```
    def backward_sentence_starts(count)
```

count :	the number of sentences to move
Returns :	`TRUE` if the textiter location moved and is not the end textiter

The `backward_sentence_starts`() method calls the `backward_sentence_start()` method (or the `forward_sentence_end()`() method if count is negative) up to *`count`* times, or until it returns `FALSE`. If *`count`* is negative, the location moves forward instead of backward. The method returns `TRUE` if the textiter location changed and the resulting location is not at the end of the textbuffer.

gtk.TextIter.forward_cursor_position

```
    def forward_cursor_position()
```

Returns :	`TRUE` if we moved and the new position is dereferenceable

The `forward_cursor_position`() method moves the textiter location forward by a single cursor position. Cursor positions are (unsurprisingly) positions where the cursor can appear. Surprisingly, there may not be a cursor position between all characters. The most common example for European languages would be a carriage return/newline sequence. For some Unicode characters, the equivalent of say the letter "a" with an accent mark will be represented as two characters, first the letter then a "combining mark" that causes the accent to be rendered; so the cursor can't go between those two characters. The method returns `TRUE` if the textiter location changed and the resulting location is not at the end of the textbuffer.

gtk.TextIter.backward_cursor_position

```
    def backward_cursor_position()
```

Returns :	`TRUE` if we moved and the new position is dereferenceable

The `backward_cursor_position`() method is similar to the `forward_cursor_position()` method, except the location moves backward.

gtk.TextIter.forward_cursor_positions

```
    def forward_cursor_positions(count)
```

`count` :	the number of positions to move
Returns :	`TRUE` if the textiter location moved and the new position is dereferenceable

The `forward_cursor_positions`() method moves up to *`count`* cursor positions. See the `forward_cursor_position()` method for more details. The method returns `TRUE` if the textiter moved to a dereferenceable location.

gtk.TextIter.backward_cursor_positions

```
    def backward_cursor_positions(count)
```

`count` :	the number of positions to move
Returns :	`TRUE` if the textiter location moved and the new position is dereferenceable

The `backward_cursor_positions`() method moves the textiter location up to *`count`* cursor positions. See the `forward_cursor_position()` method for details.

gtk.TextIter.set_offset

```
    def set_offset(char_offset)
```

`char_offset` :	a character number

The `set_offset`() method sets the textiter location to point to the location that is *`char_offset`* counts from the start of the textbuffer (starting with 0).

gtk.TextIter.set_line

```
    def set_line(line_number)
```

line_number :	a line number (counted from 0)

The `set_line`() method sets the textiter location to the start of the line specified by *`line_number`*. If *`line_number`* is negative or larger than the number of lines in the textbuffer, the method moves the textiter location to the start of the last line in the buffer.

gtk.TextIter.set_line_offset

```
    def set_line_offset(char_on_line)
```

char_on_line :	a character offset relative to the start of the textiter location's current line

The `set_line_offset`() method moves the textiter location within a line, to the new *character* (not byte) offset specified by *`char_on_line`*. The character offset must be less than or equal to the number of characters in the line; if equal, the textiter location moves to the start of the next line. See the `set_line_index()` method if you have a byte index rather than a character offset.

gtk.TextIter.set_line_index

```
    def set_line_index(byte_on_line)
```

byte_on_line :	a byte index relative to the start of the textiter location's current line

The `set_line_index`() method is similar to the `set_line_offset()`, but works with a *byte* index instead of a character index. The given byte index must be at the start of a character, it can't be in the middle of a UTF−8 encoded character.

gtk.TextIter.forward_to_end

```
    def forward_to_end()
```

The `forward_to_end`() method moves the textiter location forward to the "end textiter," that points one past the last valid character in the buffer. The `get_char()` method called on the end textiter returns 0, which is convenient for writing loops.

gtk.TextIter.forward_to_line_end

```
    def forward_to_line_end()
```

Returns :	`TRUE` if we moved and the new location is not the end textiter

The `forward_to_line_end`() method moves the textiter to point to the paragraph delimiter characters at the end of the current line. The paragraph delimiter characters are a newline, a carriage return, a carriage return−newline in sequence, or the Unicode paragraph separator character. If the textiter is already at the paragraph delimiter characters, moves to the paragraph delimiter characters for the next line. If the textiter location is on the last line in the buffer, which does not end in paragraph delimiters, moves to the end textiter (end of the last line), and returns `FALSE`.

gtk.TextIter.set_visible_line_offset

```
    def set_visible_line_offset(char_on_line)
```

char_on_line :	a character offset

The `set_visible_char_offset()` method is similar to the set_line_offset() method, but the offset is in visible characters, i.e. text with the invisible attribute set is not counted in the offset.

gtk.TextIter.set_visible_line_index

```
def set_visible_line_index(byte_on_line)
```

byte_on_line :	a byte index

The `set_visible_line_index()` method is similar to the set_line_index() method, but the index is in visible bytes, i.e. text with the attribute set is not counted in the index.

gtk.TextIter.forward_to_tag_toggle

```
def forward_to_tag_toggle(tag)
```

tag :	a `gtk.TextTag`, or `None`
Returns :	`TRUE` if a tag toggle was found after the textiter location

The `forward_to_tag_toggle()` method moves the textiter location forward to the next toggle (on or off) of the `gtk.TextTag` specified by *tag*, or to the next toggle of any tag if *tag* is `None`. If no matching tag toggles are found, this method returns `FALSE` and sets the textiter location to the end of the textbuffer; otherwise, returns `TRUE`. The `forward_to_tag_toggle()` method does not recognize toggles located at the textiter location, only toggles after the textiter location.

gtk.TextIter.backward_to_tag_toggle

```
def backward_to_tag_toggle(tag)
```

tag :	a `gtk.TextTag`, or `None`
Returns :	`TRUE` if a tag toggle was found before the textiter location

The `backward_to_tag_toggle()` method moves the textiter location backward to the next toggle (on or off) of the `gtk.TextTag` specified by *tag*, or to the next toggle of any tag if *tag* is `None`. If no matching tag toggles are found, this method returns `FALSE` and sets the textiter location to the start of the textbuffer; otherwise, returns `TRUE`. The `backward_to_tag_toggle()` method does not recognize toggles located at the textiter location, only toggles before the textiter location.

gtk.TextIter.forward_find_char

```
def forward_find_char(pred, user_data, limit)
```

pred :	a function to be called on each character
user_data :	user data for *pred*
limit :	a `gtk.TextIter` pointing at a position to end the search, or `None` for the end of the buffer.
Returns :	`TRUE` if a match was found

Note

This method is available in PyGTK 2.4 and above.

The `forward_find_char()` method advances the textiter, calling the function specified by *pred* on each character. If *pred* returns `TRUE`, `forward_find_char` stops scanning and returns `TRUE`. If *pred* never returns `TRUE`, the textiter location is set to *limit* or the end textiter, if *limit* is `None`.

Warning

This method is likely to be very slow since the Python function *pred* is called for every character.

gtk.TextIter.backward_find_char

```
    def backward_find_char(pred, user_data, limit)
```

pred :	a function to be called on each character
user_data :	user data for *pred*
limit :	a gtk.TextIter pointing at a position to end the search, or None for the beginning of the buffer.
Returns :	TRUE if a match was found

Note

This method is available in PyGTK 2.4 and above.

The backward_find_char() method is similar to the forward_find_char() method, but goes backward from the textiter location.

Warning

This method is likely to be very slow since the Python function *pred* is called for every character.

gtk.TextIter.forward_search

```
    def forward_search(str, flags, limit=None)
```

str :	a search string
flags :	the flags affecting how the search is done
limit :	a bound for the search, or None to set the bound to the end of the buffer
Returns :	a tuple containing gtk.TextIter objects pointing at the start and end locations of the match

The forward_search() method searches forward for the text string specified by *str* and returns a tuple containing gtk.TextIter objects that point at the start and end locations of the match. The search will stop at the location specified by *limit* or the end of the textbuffer if *limit* is None or is not specified. Note that a search is a linear or O(n) operation, so you may wish to use *limit* to avoid locking up your UI when searching large buffers.

If the gtk.TEXT_SEARCH_VISIBLE_ONLY flag is present, the match may have invisible text interspersed in *str* (i.e. *str* will be a possibly–noncontiguous subsequence of the matched range). Likewise, if gtk.TEXT_SEARCH_TEXT_ONLY is present, the match may have pixbufs or child anchors mixed inside the matched range. If these flags are not given, the match must be exact i.e. the special 0xFFFC character in *str* will match embedded pixbufs or child widgets.

gtk.TextIter.backward_search

```
    def backward_search(str, flags, limit=None)
```

str :	a search string
flags :	the flags affecting the search
limit :	a bound for the search, or None to set the bound to the end of the buffer

Returns :	start of match and end of match

The `backward_search()` method is the same as the `forward_search()` method, except searches backward.

gtk.TextIter.equal

```
def equal(rhs)
```

rhs :	another `gtk.TextIter` object
Returns :	`TRUE` if the textiters point to the same place in the buffer

The `equal()` method tests if the textiter specified by *rhs* points to the same location in the textbuffer as the textiter.

gtk.TextIter.compare

```
def compare(rhs)
```

rhs :	another `gtk.TextIter` object
Returns :	−1 if the textiter location is less than the *rhs* location, 1 if the textiter location is greater, 0 if they are equal

The `compare()` method returns:

- −1 if the textiter location is less than the location of the textiter specified by *rhs*;
- 1 if the textiter location is greater than the location of the textiter specified by *rhs*; and,
- 0 if the textiter location is equal to the location of the textiter specified by *rhs*.

Ordering is in character offset order, i.e. the first character in the buffer is less than the second character in the buffer.

gtk.TextIter.in_range

```
def in_range(start, end)
```

start :	the start of the text range
end :	the end of the text range
Returns :	`TRUE` if the textiter location is in the text range

The `in_range()` method returns `TRUE` if the textiter location is in the text range specified by the `gtk.TextIter` objects *start* and *end*. *start* and *end* must be in ascending order.

gtk.TextIter.order

```
def order(second)
```

second :	another `gtk.TextIter` object

The order() method swaps the locations of *first* and *second* if *second* comes before *first* in the buffer. This method can be used to ensure that *first* and *second* are in sequence. Most text buffer methods that take a range call this automatically, so there's no real reason to call it yourself in those cases. There are some exceptions, such as the `in_range()`, that expect a pre−sorted range.

gtk.TextMark

gtk.TextMark — a position in a textbuffer that is preserved across textbuffer modifications

Synopsis

```
class gtk.TextMark(gobject.GObject):
    def set_visible(setting)
    def get_visible()
    def get_name()
    def get_deleted()
    def get_buffer()
    def get_left_gravity()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.TextMark
```

Description

A `gtk.TextMark` is like a bookmark in a textbuffer – it preserves a position in the text. You can get an iterator corresponding to a textmark by using the `gtk.TextBuffer.get_iter_at_mark()` method. Unlike iterators, textmarks remain valid across buffer modifications (e.g. when text is inserted or deleted). When text containing a textmark is deleted, the textmark remains in the position originally occupied by the deleted text. When text is inserted at a textmark, a textmark with left gravity will be moved to the beginning of the newly–inserted text, and a textmark with right gravity will be moved to the end. Textmarks optionally have names that can be used to avoid passing the `gtk.TextMark` object around. Textmarks are typically created using the `gtk.TextBuffer.create_mark()` method. A `gtk.TextBuffer` has two built–in `gtk.TextMark` objects named: `insert` and `selection_bound` which refer to the insertion point and the boundary of the selection (these may refer to the same location).

Methods

gtk.TextMark.set_visible

```
    def set_visible(setting)
```

`setting` :	if `TRUE` the textmark is visible

The `set_visible()` method sets the visibility of the textmark to the value specified by *`setting`*. If *`setting`* is `TRUE` the textmark will be visible as a vertical bar. The insertion point is normally visible but most textmarks are not visible by default. The text widget uses a visible textmark to indicate where a drop will occur when dragging–and–dropping text.

gtk.TextMark.get_visible

```
    def get_visible()
```

Returns :	`TRUE` if the textmark is visible

The `get_visible`() method returns `TRUE` if the textmark is visible (i.e. a vertical bar is displayed for it)

gtk.TextMark.get_name

```
    def get_name()
```

Returns :	the textmark name or `None`

The `get_name`() method returns the textmark name or `None` if the textmark is anonymous.

gtk.TextMark.get_deleted

```
    def get_deleted()
```

Returns :	`TRUE` if the textmark is deleted

The `get_deleted`() method returns `TRUE` if the textmark has been removed from its textbuffer with `gtk.TextBuffer.delete_mark`(). Textmarks can't be used once deleted.

gtk.TextMark.get_buffer

```
    def get_buffer()
```

Returns :	the textmark's `gtk.TextBuffer`

The `get_buffer`() method returns the `gtk.TextBuffer` object the textmark is located inside, or `None` if the textmark is deleted.

gtk.TextMark.get_left_gravity

```
    def get_left_gravity()
```

Returns :	`TRUE` if the textmark has left gravity

The `get_left_gravity`() method returns TRUE if the textmark has left gravity.

gtk.TextTag

gtk.TextTag — an object used to apply attributes to text in a `gtk.TextBuffer`

Synopsis

```
class gtk.TextTag(gobject.GObject):
    gtk.TextTag(name=None)
    def get_priority()
    def set_priority(priority)
    def event(event_object, event, iter)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.TextTag
```

Properties

"name"	Read−Write	The name of the texttag or `None` if anonymous
"background"	Write	The background color as a string
"foreground"	Write	The foreground color as a string
"background−gdk"	Read−Write	The background color as a (possibly unallocated) `gtk.gdk.Color`
"foreground−gdk"	Read−Write	The foreground color as a (possibly unallocated) `gtk.gdk.Color`
"background−stipple"	Read−Write	The bitmap to use as a mask when drawing the text background
"foreground−stipple"	Read−Write	The bitmap to use as a mask when drawing the text foreground
"font"	Read−Write	The font description as a string, e.g. "Sans Italic 12"
"font−desc"	Read−Write	The font description as a pango.FontDescription object
"family"	Read−Write	The name of the font family, e.g. Sans, Helvetica, Times, Monospace
"style"	Read−Write	The font style – one of: `pango.STYLE_NORMAL`, `pango.STYLE_OBLIQUE` or `pango.STYLE_ITALIC`.
"variant"	Read−Write	The font variant – either `pango.VARIANT_NORMAL` or `pango.VARIANT_SMALL_CAPS`.
"weight"	Read−Write	The font weight as an integer: `pango.WEIGHT_ULTRALIGHT` = 200, `pango.WEIGHT_LIGHT` = 300, `pango.WEIGHT_NORMAL` = 400, `pango.WEIGHT_BOLD` = 700, `pango.WEIGHT_ULTRABOLD` = 800, `pango.WEIGHT_HEAVY` = 900.
"stretch"	Read−Write	The font stretch – one of: `pango.STRETCH_ULTRA_CONDENSED`, `pango.STRETCH_EXTRA_CONDENSED`, `pango.STRETCH_CONDENSED`, `pango.STRETCH_SEMI_CONDENSED`, `pango.STRETCH_NORMAL`, `pango.STRETCH_SEMI_EXPANDED`, `pango.STRETCH_EXPANDED`, `pango.STRETCH_EXTRA_EXPANDED`, `pango.STRETCH_ULTRA_EXPANDED`

"size"	Read−Write	The font size in Pango units.
"size−points"	Read−Write	The font size in points
"scale"	Read−Write	The font size as a scale factor relative to the default font size. This properly adapts to theme changes etc. so is recommended. Pango predefines some scales such as `pango.SCALE_XX_SMALL`, `pango.SCALE_X_SMALL`, `pango.SCALE_SMALL`, `pango.SCALE_MEDIUM`, `pango.SCALE_LARGE`, `pango.SCALE_X_LARGE`, `pango.SCALE_XX_LARGE`.
"pixels−above−lines"	Read−Write	The number of pixels of blank space above paragraphs
"pixels−below−lines"	Read−Write	The number of pixels of blank space below paragraphs
"pixels−inside−wrap"	Read−Write	The number of pixels of blank space between wrapped lines in a paragraph
"editable"	Read−Write	It `TRUE` the text can be modified by the user
"wrap−mode"	Read−Write	The wrap mode of the text: `gtk.WRAP_NONE`, `gtk.WRAP_CHAR` or `gtk.WRAP_WORD`
"justification"	Read−Write	The text justification: `gtk.JUSTIFY_LEFT`, `gtk.JUSTIFY_RIGHT`, `gtk.JUSTIFY_CENTER` or `gtk.JUSTIFY_FILL`
"direction"	Read−Write	The text direction: `gtk.TEXT_DIR_NONE`, `gtk.TEXT_DIR_LTR` or `gtk.TEXT_DIR_RTL`
"left−margin"	Read−Write	The width of the left margin in pixels
"indent"	Read−Write	The amount to indent the paragraph, in pixels
"strikethrough"	Read−Write	If `TRUE`, strike through the text
"right−margin"	Read−Write	The width of the right margin in pixels
"underline"	Read−Write	The style of underline for this text: `pango.UNDERLINE_NONE`, `pango.UNDERLINE_SINGLE`, `pango.UNDERLINE_DOUBLE` or `pango.UNDERLINE_LOW`
"rise"	Read−Write	The offset of text above the baseline (below the baseline if rise is negative) in pixels
"background−full−height"	Read−Write	If `TRUE`, the background color fills the entire line height
"language"	Read−Write	The language this text is in, as an ISO code. Pango can use this as a hint when rendering the text. If you don't understand this parameter, you probably don't need it.
"tabs"	Read−Write	The custom tabs for this text
"invisible"	Read−Write	If `TRUE`, this text is hidden
"background−set"	Read−Write	If `TRUE`, this tag affects the background color
"foreground−set"	Read−Write	If `TRUE`, this tag affects the foreground color

"background−stipple−set"	Read−Write	If TRUE, this tag affects the background stipple
"foreground−stipple−set"	Read−Write	If TRUE, this tag affects the foreground stipple
"family−set"	Read−Write	If TRUE, this tag affects the font family
"style−set"	Read−Write	If TRUE, this tag affects the font style
"variant−set"	Read−Write	If TRUE, this tag affects the font variant
"weight−set"	Read−Write	If TRUE, this tag affects the font weight
"stretch−set"	Read−Write	If TRUE, this tag affects the font stretch
"size−set"	Read−Write	If TRUE, this tag affects the font size
"scale−set"	Read−Write	If TRUE, this tag scales the font size by a factor
"pixels−above−lines−set"	Read−Write	If TRUE, this tag affects the number of pixels above lines
"pixels−below−lines−set"	Read−Write	If TRUE, this tag affects the number of pixels above lines
"pixels−inside−wrap−set"	Read−Write	If TRUE, this tag affects the number of pixels between wrapped lines
"editable−set"	Read−Write	If TRUE, this tag affects text editability
"wrap−mode−set"	Read−Write	If TRUE, this tag affects line wrap mode
"justification−set"	Read−Write	If TRUE, this tag affects paragraph justification
"left−margin−set"	Read−Write	If TRUE, this tag affects the left margin
"indent−set"	Read−Write	If TRUE, this tag affects indentation
"strikethrough−set"	Read−Write	If TRUE, this tag affects strikethrough
"right−margin−set"	Read−Write	If TRUE, this tag affects the right margin
"underline−set"	Read−Write	If TRUE, this tag affects underlining
"rise−set"	Read−Write	If TRUE, this tag affects the rise
"background−full−height−set"	Read−Write	If TRUE, this tag affects background height
"language−set"	Read−Write	If TRUE, this tag affects the language the text is rendered as
"tabs−set"	Read−Write	If TRUE, this tag affects tabs
"invisible−set"	Read−Write	If TRUE, this tag affects text visibility

Signal Prototypes

```
"event"   def callback(texttag, widget, event, iter, user_param1, ...)
```

Description

A gtk.TextTag object holds attributes that can be applied to a range of text in a gtk.TextBuffer. A texttag can be associated with more than one gtk.TextBuffer by adding it to the gtk.TextTagTable objects of the textbuffers. The attributes of a texttag can be set using the GObject.set_property() method or as part of texttag creation using the gtk.TextBuffer.create_tag() method. Since not every attribute property of a gtk.TextTag may be set each attribute property has a boolean property that indicates whether the attribute property is set by this texttag. Therefore before retrieving an attribute value from a texttag you have to check if the associated boolean property of the attribute property is TRUE.

Constructor

```
gtk.TextTag(name=None)
```

name :	tag name, or `None` if the texttag is anonymous
Returns :	a new `gtk.TextTag`

Creates a `gtk.TextTag` with the name specified by *name*. If *name* is `None` the texttag will be anonymous. The texttag attributes are configured using the `GObject.set_property()` method.

Methods

gtk.TextTag.get_priority

```
def get_priority()
```

Returns :	the texttag's priority.

The get_priority() method returns the priority or the texttag.

gtk.TextTag.set_priority

```
def set_priority(priority)
```

priority :	the new priority

The `set_priority`() method sets the priority of a `gtk.TextTag` to the value specified by *priority*. Valid priorities start at 0 and go to one less than the value returned by the `gtk.TextTagTable.get_size()` method. Each texttag in a table has a unique priority; setting the priority of one texttag shifts the priorities of all the other texttags in the table to maintain a unique priority for each texttag. Higher priority tags "win" if two texttags both set the same text attribute for a range of text. When adding a texttag to a `gtk.TextTagTable`, it will be assigned the highest priority in the table by default; so normally the precedence of a set of texttags is the order in which they were added to the table, or created with the `gtk.TextBuffer.create_tag()` method, that adds the texttag to the buffer's table automatically.

gtk.TextTag.event

```
def event(event_object, event, iter)
```

event_object :	the object that received the event, such as a widget
event :	the event
iter :	the location where the event was received
Returns :	the result of signal emission (whether the event was handled)

The `event`() method emits the "event" signal on the `gtk.TextTag` for the widget specified by *event_object* with the event specified by *event* at the textbuffer location specified by *iter*. This method returns `TRUE` if the event was handled.

Signals

The "event" gtk.TextTag Signal

```
  def callback(texttag, widget, event, iter, user_param1, ...)
```

texttag :	the texttag that received the signal
widget :	the widget that received *event*
event :	the event
iter :	the `gtk.TextIter` pointing to the location where the event was received
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "event" signal is emitted when an event occurs in a range of text that is enclosed in the *texttag*. The widget that the event occurred in is specified by *widget*. *iter* holds the location that the event occurred at and *event* describes the event.

gtk.TextTagTable

gtk.TextTagTable A collection of `gtk.TextTag` objects that can be used together

Synopsis

```
class gtk.TextTagTable(gobject.GObject):
    gtk.TextTagTable()
    def add(tag)
    def remove(tag)
    def lookup(name)
    def foreach(func, data=None)
    def get_size()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.TextTagTable
```

Signal Prototypes

"tag−added"	`def callback(texttagtable, texttag, user_param1, ...)`
"tag−changed"	`def callback(texttagtable, texttag, size_changed, user_param1, ...)`
"tag−removed"	`def callback(texttagtable, texttag, user_param1, ...)`

Description

A `gtk.TextTagTable` object holds a set of `gtk.TextTag` objects for use with a `gtk.TextBuffer`. Usually a `gtk.TextTagTable` is created automatically when a `gtk.TextBuffer` is created but a standalone `gtk.TextTagTable` can be created using the gtk.TextTagTable() constructor. A `gtk.TextTagTable` can be passed as an argument to the gtk.TextBuffer() constructor to associate an existing `gtk.TextTagTable` rather than creating a new one. This is useful when several textbuffers need to use the same texttags.

Constructor

```
gtk.TextTagTable()
```

Returns :	a new `gtk.TextTagTable`

Creates a new `gtk.TextTagTable`. The table contains no tags by default.

Methods

gtk.TextTagTable.add

```
def add(tag)
```

`tag` :	a `gtk.TextTag`

The `add`() method adds a texttag to the texttagtable. The texttag is assigned the highest priority in the texttagtable. A `ValueError` exception is raised if *`tag`* is in a texttag table already, or has the same name as another texttag in the texttagtable.

gtk.TextTagTable.remove

```
def remove(tag)
```

`tag` :	a `gtk.TextTag`

The `remove`() method removes a texttag from the texttagtable.

gtk.TextTagTable.lookup

```
def lookup(name)
```

`name` :	the name of a texttag
Returns :	The texttag, or `None` if none by that name is in the texttagtable.

The `lookup`() method looks in the texttagtable for a `gtk.TextTag` with the name specified by *`name`* and returns it if found. This method returns `None` if *`name`* does not identify a `gtk.TextTag` in the texttagtable.

gtk.TextTagTable.foreach

```
def foreach(func, data=None)
```

`func` :	a function to call on each texttag
`data` :	user data to pass to *`func`* or `None`

Note

This method is available in PyGTK 2.4 and above.

The `foreach`() method calls the function specified by *`func`* on each texttag in the text tag table passing the user data specified by *`data`*. The signature of *`func`* is:

```
def func(texttag, user_data):
```

where *texttag* is a gtk.TextTag in the text tag table and *user_data* is *data*.

gtk.TextTagTable.get_size

```
def get_size()
```

Returns :	the number of texttags in the texttagtable

The get_size() method returns the size of the texttagtable (number of texttags).

Signals

The "tag−added" gtk.TextTagTable Signal

```
def callback(texttagtable, texttag, user_param1, ...)
```

texttagtable :	the texttagtable that received the signal
texttag :	a gtk.TextTag
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "tag−added" signal is emitted when the gtk.TextTag specified by *texttag* is added to *texttagtable*.

The "tag−changed" gtk.TextTagTable Signal

```
def callback(texttagtable, texttag, size_changed, user_param1, ...)
```

texttagtable :	the texttagtable that received the signal
texttag :	a gtk.TextTag
size_changed :	if TRUE a *texttag* property has changed that may affect the size of the text enclosed by the texttag
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "tag−changed" signal is emitted when a property of the gtk.TextTag specified by *texttag* is changed. If *size_changed* is TRUE the text enclosed by *texttag* will change size.

The "tag−removed" gtk.TextTagTable Signal

```
def callback(texttagtable, texttag, user_param1, ...)
```

texttagtable :	the texttagtable that received the signal
texttag :	a gtk.TextTag
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "tag−removed" signal is emitted when the gtk.TextTag specified by *texttag* is removed from *texttagtable*

Prev Up Next

gtk.TextView

gtk.TextView a widget that displays the contents of a gtk.TextBuffer

Synopsis

```
class gtk.TextView(gtk.Container):
    gtk.TextView(buffer=None)
    def set_buffer(buffer)
    def get_buffer()
    def scroll_to_iter(iter, within_margin, use_align=FALSE, xalign=0.5, yalign=0.5)
    def scroll_to_mark(mark, within_margin, use_align=FALSE, xalign=0.5, yalign=0.5)
    def scroll_mark_onscreen(mark)
    def move_mark_onscreen(mark)
    def place_cursor_onscreen()
    def get_visible_rect()
    def set_cursor_visible(setting)
    def get_cursor_visible()
    def get_iter_location(iter)
    def get_iter_at_location(iter, x, y)
    def get_line_yrange(iter)
    def get_line_at_y(target_iter)
    def buffer_to_window_coords(win, buffer_x, buffer_y)
    def window_to_buffer_coords(win, window_x, window_y)
    def get_window(win)
    def get_window_type(window)
    def set_border_window_size(type, size)
    def get_border_window_size(type)
    def forward_display_line(iter)
    def backward_display_line(iter)
    def forward_display_line_end(iter)
    def backward_display_line_start(iter)
    def starts_display_line(iter)
    def move_visually(iter, count)
    def add_child_at_anchor(child, anchor)
    def add_child_in_window(child, which_window, xpos, ypos)
    def move_child(child, xpos, ypos)
    def set_wrap_mode(wrap_mode)
    def get_wrap_mode()
    def set_editable(setting)
    def get_editable()
    def set_overwrite(overwrite)
    def get_overwrite()
    def set_accepts_tab(accepts_tab)
    def get_accepts_tab()
    def set_pixels_above_lines(pixels_above_lines)
    def get_pixels_above_lines()
    def set_pixels_below_lines(pixels_below_lines)
    def get_pixels_below_lines()
    def set_pixels_inside_wrap(pixels_inside_wrap)
    def get_pixels_inside_wrap()
    def set_justification(justification)
    def get_justification()
    def set_left_margin(left_margin)
    def get_left_margin()
    def set_right_margin(right_margin)
    def get_right_margin()
    def set_indent(indent)
    def get_indent()
    def set_tabs(tabs)
    def get_tabs()
    def get_default_attributes()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.TextView
```

Properties

"accepts–tab"	Read–Write	If `TRUE`, pressing **Tab** will result in a tab character being entered; otherwise the focus will be moved. Available in GTK+ 2.4 and above.
"buffer"	Read–Write	The buffer that is displayed. Available in GTK+ 2.4 and above.
"cursor–visible"	Read–Write	If `TRUE`, the insertion cursor is shown
"editable"	Read–Write	If `TRUE`, the text can be modified by the user by default
"indent"	Read–Write	The default amount to indent the paragraph, in pixels
"justification"	Read–Write	The default text justification: `gtk.JUSTIFY_LEFT`, `gtk.JUSTIFY_RIGHT`, `gtk.JUSTIFY_CENTER` or `gtk.JUSTIFY_FILL`
"left–margin"	Read–Write	The default width of the left margin in pixels
"overwrite"	Read–Write	If `TRUE`, the entered text overwrites existing contents. Available in GTK+ 2.4 and above.
"pixels–above–lines"	Read–Write	The default number of pixels of blank space above paragraphs
"pixels–below–lines"	Read–Write	The default number of pixels of blank space below paragraphs
"pixels–inside–wrap"	Read–Write	The default number of pixels of blank space between wrapped lines in a paragraph
"right–margin"	Read–Write	The default width of the right margin in pixels
"tabs"	Read–Write	The default custom tabs
"wrap–mode"	Read–Write	The default wrap mode: `gtk.WRAP_NONE`, `gtk.WRAP_CHAR` or `gtk.WRAP_WORD`

Style Properties

"error–underline–color"	Read–Write	The `gtk.gdk.Color` with which to draw error–indication underlines. Available in GTK+ 2.4 and above.

Signal Prototypes

"copy–clipboard"	`def callback(`*`textview, user_param1, ...`*`)`
"cut–clipboard"	`def callback(`*`textview, user_param1, ...`*`)`
"delete–from–cursor"	`def callback(`*`textview, delete_type, count, user_param1, ...`*`)`
"insert–at–cursor"	`def callback(`*`textview, string, user_param1, ...`*`)`

"move−cursor"	`def callback(textview, step_size, count, extend_selection, user_param1, ...)`
"move−focus"	`def callback(textview, direction, user_param1, ...)`
"move−viewport"	`def callback(textview, scrollstep, count, user_param1, ...)`
"page−horizontally"	`def callback(textview, count, extend_selection, user_param1, ...)`
"paste−clipboard"	`def callback(textview, user_param1, ...)`
"populate−popup"	`def callback(textview, menu, user_param1, ...)`
"select−all"	`def callback(textview, select, user_param1, ...)`
"set−anchor"	`def callback(textview, user_param1, ...)`
"set−scroll−adjustments"	`def callback(textview, hadjustment, vadjustment, user_param1, ...)`
"toggle−overwrite"	`def callback(textview, user_param1, ...)`

Description

A `gtk.TextView` widget provides the display for the contents of a `gtk.TextBuffer` object. A textview provides a set of attributes for the default display of text from a textbuffer. The attributes set by `gtk.TextTag` objects override the attributes set on a `gtk.TextView` widget. Since a `gtk.TextBuffer` can be associated with multiple `gtk.TextView` widgets each having a different set of default attributes, the display of the same text in different textview widgets can be quite different except for those attributes that are overridden by texttags.

A `gtk.TextView` widget has several `gtk.gdk.Window` widgets most of which are not displayed by default:

`gtk.TEXT_WINDOW_WIDGET`	The widget window
`gtk.TEXT_WINDOW_TEXT`	The window that holds the text
`gtk.TEXT_WINDOW_LEFT`	The left border window − not displayed by default
`gtk.TEXT_WINDOW_RIGHT`	The right border window − not displayed by default
`gtk.TEXT_WINDOW_TOP`	The top border window − not displayed by default
`gtk.TEXT_WINDOW_BOTTOM`	The bottom border window − not displayed by default
`gtk.TEXT_WINDOW_PRIVATE`	An internal inaccessible `gtk.gdk.Window`

The border windows are not created until they are given a size by using the `set_border_window_size()` method.

Depending on the wrap mode of the textview a textbuffer line may be displayed as more than one screen display line. The textview has methods to navigate through the display lines.

A `gtk.TextChildAnchor` in a `gtk.TextBuffer` provides a place for a `PyGTK` widget to be placed in a `gtk.TextView`. Each textview displaying the same textbuffer with a child anchor must have a different widget placed at that child anchor. In addition, a widget can be placed at a specific location in one of the above `gtk.TextView` windows using the `add_child_in_window()` method. The widget will be clipped to the window boundaries if it is larger than the window or located where it extends beyond the window boundaries.

A `gtk.TextView` has a default popup menu that includes the usual cut, copy and paste capabilities. In PyGTK 2.2 `gtk.Clipboard` objects are supported so your program can access the contents of the cut, copy and paste clipboard through the `gdk.SELECTION_CLIPBOARD` clipboard. Also the selected text in a `gtk.TextView` is available on the `gdk.SELECTION_PRIMARY` clipboard.

Constructor

```
gtk.TextView(buffer=None)
```

buffer :	a `gtk.TextBuffer` or `None`
Returns :	a new `gtk.TextView`.

Creates a new `gtk.TextView` widget displaying the `gtk.TextBuffer` specified by *`buffer`*. If *`buffer`* is `None`, a new `gtk.TextBuffer` will be created. One textbuffer can be shared among many widgets.

Methods

gtk.TextView.set_buffer

```
def set_buffer(buffer)
```

buffer :	a `gtk.TextBuffer`

The `set_buffer()` method sets the `gtk.TextBuffer` specified by *`buffer`* as the textbuffer being displayed by the textview.

gtk.TextView.get_buffer

```
def get_buffer()
```

Returns :	a `gtk.TextBuffer`

The `get_buffer()` method returns the `gtk.TextBuffer` being displayed by this textview.

gtk.TextView.scroll_to_iter

```
def scroll_to_iter(iter, within_margin, use_align=FALSE, xalign=0.5, yalign=0.5)
```

iter :	a `gtk.TextIter` object
within_margin :	the margin as a [0.0,0.5) fraction of screen size
use_align :	if `TRUE` use the alignment arguments; if `FALSE`, just get *`iter`* on screen
xalign :	the horizontal alignment of *`iter`* within visible area.
yalign :	the vertical alignment of *`iter`* within visible area
Returns :	`TRUE` if scrolling occurred

The `scroll_to_iter()` method scrolls the textview so that the `gtk.TextIter` location specified by *`iter`* is on the screen in the position indicated by *`xalign`* and *`yalign`*. If *`use_align`* is `TRUE` the alignments specify the fraction of screen space to the left of or above the location of *`iter`*. If *`use_align`* is `FALSE`, the text scrolls the minimal distance to get *`iter`* on screen, possibly not scrolling at all. The effective screen for purposes of this method is reduced by a margin of size specified by *`within_margin`*.

Note

This method uses the currently−computed height of the lines in the text buffer. The line heights are computed in an idle handler so this method may not have the desired effect if it's called before the height computations are complete. To avoid oddness, consider using the `scroll_to_mark()` method that saves a point to be scrolled to after line validation.

gtk.TextView.scroll_to_mark

```
def scroll_to_mark(mark, within_margin, use_align=FALSE, xalign=0.5, yalign=0.5)
```

`mark` :	a `gtk.TextMark` object
`within_margin` :	the margin as a [0.0,0.5) fraction of screen size
`use_align` :	if `TRUE` use the alignment arguments; if `FALSE`, just get *`mark`* on screen
`xalign` :	the horizontal alignment of *`mark`* within the visible area.
`yalign` :	the vertical alignment of *`mark`* within the visible area

The `scroll_to_mark()` method scrolls the textview so that the `gtk.TextMark` location specified by *`mark`* is on the screen in the position specified by *`xalign`* and *`yalign`*. If *`use_align`* is `TRUE` the alignments specify the fraction of screen space to the left of or above the location of *`mark`*. If *`use_align`* is `FALSE`, the text scrolls the minimal distance to get *`mark`* on screen, possibly not scrolling at all. The effective screen for purposes of this function is reduced by a margin of size specified by *`within_margin`*.

gtk.TextView.scroll_mark_onscreen

```
def scroll_mark_onscreen(mark)
```

`mark` :	a `gtk.TextMark` in the textbuffer for textview

The `scroll_mark_onscreen()` method scrolls the textview the minimum distance to place the `gtk.TextMark` location specified by *`mark`* within the visible area of the widget.

gtk.TextView.move_mark_onscreen

```
def move_mark_onscreen(mark)
```

`mark` :	a `gtk.TextMark` object
Returns :	`TRUE` if *`mark`* moved (wasn't already on screen)

The `move_mark_onscreen()` moves the `gtk.TextMark` location specified by *`mark`* to a location within the currently−visible text area of the textview.

gtk.TextView.place_cursor_onscreen

```
def place_cursor_onscreen()
```

Returns :	TRUE if the cursor had to be moved.

The `place_cursor_onscreen()` method moves the cursor to a new location within the currently visible region of the buffer, if it isn't there already.

gtk.TextView.get_visible_rect

```
def get_visible_rect()
```

Returns :	a `gtk.gdk.Rectangle`

The `get_visible_rect()` method returns a `gtk.gdk.Rectangle` containing the coordinates of the currently−visible region of the buffer. The rectangle is in buffer coordinates that can be converted to window coordinates with the buffer_to_window_coords() method.

gtk.TextView.set_cursor_visible

```
def set_cursor_visible(setting)
```

setting :	if TRUE show the insertion cursor

The set_cursor_visible() method sets the "cursor−visible" property to the value of *setting*. If *setting* is TRUE the cursor is visible; if FALSE, it is not. A buffer with no editable text probably shouldn't have a visible cursor, so you may want to turn the cursor off.

gtk.TextView.get_cursor_visible

```
def get_cursor_visible()
```

Returns :	TRUE if the insertion mark is visible

The get_cursor_visible() method returns the value of the "cursor−visible" property that determines if the insertion point is visible.

gtk.TextView.get_iter_location

```
def get_iter_location(iter)
```

iter :	a gtk.TextIter
Returns :	a gtk.gdk.Rectangle containing the bounds of the character at *iter*

The get_iter_location() method returns a gtk.gdk.Rectangle that contains the character at the location specified by *iter*. The rectangle position is in buffer coordinates that can be converted to window coordinates with the buffer_to_window_coords() method.

gtk.TextView.get_iter_at_location

```
def get_iter_at_location(x, y)
```

x :	x position, in buffer coordinates
y :	y position, in buffer coordinates
Returns :	a gtk.TextIter

The get_iter_at_location() method returns a gtk.TextIter that points at the location specified by the buffer coordinates *x* and *y*. Buffer coordinates are coordinates for the entire buffer, not just the currently−displayed portion. Window coordinates from an event, must be converted to buffer coordinates with the window_to_buffer_coords() method before using them with this method.

gtk.TextView.get_line_yrange

```
def get_line_yrange(iter)
```

iter :	a gtk.TextIter
Returns :	a tuple containing a y coordinate and a height

The get_line_yrange() method locates the line containing the gtk.TextIter specified by *iter* and returns a tuple containing the y coordinate of the top of the line and the height of the line. The coordinate is a buffer coordinate that can be converted to window coordinates with the buffer_to_window_coords() method.

gtk.TextView.get_line_at_y

```
    def get_line_at_y(y)
```

y :	a y coordinate
Returns :	a tuple containing a gtk.TextIter pointing at the line start and the top coordinate of the line

The get_line_at_y() method returns a tuple containing:

- a gtk.TextIter pointing at the start of the line with the vertical coordinate specified by *y* and,
- the vertical coordinate of the top edge of the line.

y is in buffer coordinates that can be converted from window coordinates with the window_to_buffer_coords() method.

gtk.TextView.buffer_to_window_coords

```
    def buffer_to_window_coords(win, buffer_x, buffer_y)
```

win :	one of the textview windows except gtk.TEXT_WINDOW_PRIVATE
buffer_x :	buffer x coordinate
buffer_y :	buffer y coordinate
Returns :	a tuple containing the window x and y coordinates

The buffer_to_window_coords() method returns a tuple containing the x and y coordinates for the window specified by *win* that correspond to the textbuffer coordinates specified by *buffer_x* and *buffer_y*. See the above description for more details on textview window types.

Note

You can't convert coordinates for a non–existing window (see the set_border_window_size() method).

gtk.TextView.window_to_buffer_coords

```
    def window_to_buffer_coords(win, window_x, window_y)
```

win :	a textview window except gtk.TEXT_WINDOW_PRIVATE
window_x :	window x coordinate
window_y :	window y coordinate
Returns :	a tuple containing the textbuffer x and y coordinates

The window_to_buffer_coords() method returns a tuple containing the textbuffer x and y coordinates corresponding to the *window_x* and *window_y* coordinates in the window specified by *win*, See the above description for more details on textview window types.

Note

You can't convert coordinates for a non–existing window (see the set_border_window_size()) method.

gtk.TextView.get_window

```
    def get_window(win)
```

win :	a textview window type

Returns :	a gtk.gdk.Window, or None

The get_window() method returns the gtk.gdk.Window corresponding to an area of the textview specified by *win*:

gtk.TEXT_WINDOW_WIDGET	The widget window
gtk.TEXT_WINDOW_TEXT	The window that holds the text
gtk.TEXT_WINDOW_LEFT	The left border window – not displayed by default
gtk.TEXT_WINDOW_RIGHT	The right border window – not displayed by default
gtk.TEXT_WINDOW_TOP	The top border window – not displayed by default
gtk.TEXT_WINDOW_BOTTOM	The bottom border window – not displayed by default
gtk.TEXT_WINDOW_PRIVATE	An internal inaccessible gtk.gdk.Window

This method returns None if the window is nonexistent i.e. if its width or height is 0.

gtk.TextView.get_window_type

```
def get_window_type(window)
```

window :	a window type
Returns :	the window type.

The get_window_type() method returns the type of the gtk.gdk.Window specified by *window*. This method is used to find out what window type an event corresponds to. If you connect to an event signal on the textview, this method can be called on event.window to see what window type it was. See the get_window() method for more details on window types.

gtk.TextView.set_border_window_size

```
def set_border_window_size(type, size)
```

type :	a textview border window type
size :	the width or height of the window

The set_border_window_size() method sets the width of a gtk.TEXT_WINDOW_LEFT or gtk.TEXT_WINDOW_RIGHT window, or the height of a gtk.TEXT_WINDOW_TOP or gtk.TEXT_WINDOW_BOTTOM window. This method automatically destroys the corresponding window if the size is set to 0, and creates the window if the size is set to non–zero. This method can only be used for the "border windows", it doesn't work with the gtk.TEXT_WINDOW_WIDGET, gtk.TEXT_WINDOW_TEXT, or gtk.TEXT_WINDOW_PRIVATE windows.

gtk.TextView.get_border_window_size

```
def get_border_window_size(type)
```

type :	a textview border window type
Returns :	the width or height of the textview border window

The get_border_window_size() method returns the width or height of the border window of the type specified by *type*. See the set_border_window_size() method for more details.

gtk.TextView.forward_display_line

```
def forward_display_line(iter)
```

`iter` :	a `gtk.TextIter`
Returns :	`TRUE` if *`iter`* was moved and is not on the end iterator

The `forward_display_line`() method moves the location of *`iter`* forward by one display line. A textview display line is different from a textbuffer line. Textbuffer lines are paragraphs and are separated by newlines or other paragraph separator characters. Display lines are created by line−wrapping a textbuffer line. If wrapping is turned off, display lines and textbuffer lines will be the same. Display lines are divided differently for each textview, since they depend on the textview's width and the textview's default wrap mode. Paragraphs are the same in all views, since they depend on the contents of the `gtk.TextBuffer`. This method returns `TRUE` if the location of *`iter`* moves to a dereferenceable position (i.e. not the end position).

gtk.TextView.backward_display_line

```
    def backward_display_line(iter)
```

`iter` :	a `gtk.TextIter`
Returns :	`TRUE` if *`iter`* was moved

The `backward_display_line`() method moves the location of *`iter`* backward by one display line. A textview display line is different from a textbuffer line. Textview lines are paragraphs and are separated by newlines or other paragraph separator characters. Display lines are created by line−wrapping a textbuffer line. If wrapping is turned off, display lines and textbuffer lines will be the same. Display lines are divided differently for each textview, since they depend on the textview's width and the textview's default wrap mode. Paragraphs are the same in all views, since they depend on the contents of the `gtk.TextBuffer`. This method returns `TRUE` if the location of *`iter`* moves to a dereferenceable position (i.e. not the end position).

gtk.TextView.forward_display_line_end

```
    def forward_display_line_end(iter)
```

`iter` :	a `gtk.TextIter`
Returns :	`TRUE` if *`iter`* was moved and is not on the end iterator

The `forward_display_line_end`() method moves the location of *`iter`* forward to the next display line end. A textview display line is different from a textbuffer line. Textbuffer lines are paragraphs and are separated by newlines or other paragraph separator characters. Display lines are created by line−wrapping a textbuffer line. If wrapping is turned off, display lines and textbuffer lines will be the same. Display lines are divided differently for each textview, since they depend on the textview's width and the textview's default wrap mode. Paragraphs are the same in all views, since they depend on the contents of the `gtk.TextBuffer`. This method returns `TRUE` if the location of *`iter`* moves to a dereferenceable position (i.e. not the end position).

gtk.TextView.backward_display_line_start

```
    def backward_display_line_start(iter)
```

`iter` :	a `gtk.TextIter`
Returns :	`TRUE` if *`iter`* was moved

The `backward_display_line_start`() method moves the location of *`iter`* backward to the next display line start. A textview display line is different from a textbuffer line. Textbuffer lines are paragraphs and are separated by newlines or other paragraph separator characters. Display lines are created by line−wrapping a textbuffer line. If wrapping is turned off, display lines and textbuffer lines will be the same. Display lines are divided differently for each textview, since they depend on the textview's width and the textview's default wrap mode. Paragraphs are the same in all views, since they depend on the contents of the `gtk.TextBuffer`. This method returns `TRUE` if the location of *`iter`* moves to a dereferenceable position (i.e. not the end position).

gtk.TextView.starts_display_line

```
def starts_display_line(iter)
```

iter :	a gtk.TextIter
Returns :	TRUE if the location of *iter* is at the beginning of a display line

The starts_display_line() method returns TRUE if the location of *iter* is at the start of a display line. See the forward_display_line() method for an explanation of display lines vs. textbuffer lines (paragraphs).

gtk.TextView.move_visually

```
def move_visually(iter, count)
```

iter :	a gtk.TextIter
count :	the number of characters to move (may be negative)
Returns :	TRUE if *iter* moved and is not on the end iterator

The move_visually() method moves the location of *iter* by *count* cursor positions. If count is negative the location moves against the normal text direction. Note a cursor position move may move over multiple characters when multiple characters combine to form one grapheme.

gtk.TextView.add_child_at_anchor

```
def add_child_at_anchor(child, anchor)
```

child :	a gtk.Widget
anchor :	a gtk.TextChildAnchor in the textview's gtk.TextBuffer

The add_child_at_anchor() method adds the widget specified by *child* in the textview, at the gtk.TextChildAnchor specified by *anchor*.

gtk.TextView.add_child_in_window

```
def add_child_in_window(child, which_window, xpos, ypos)
```

child :	a gtk.Widget
which_window :	the textview window *child* should appear in
xpos :	the X position of *child* in window coordinates
ypos :	the Y position of *child* in window coordinates

The add_child_in_window() method adds the widget specified by *child* at the fixed coordinates specified by *xpos* and *ypos* in one of the text widget's windows specified by *which_window*. The window must have nonzero size (see the set_border_window_size() method).

Note

The *child* coordinates are given relative to the gtk.gdk.Window specified by *which_window*, and these coordinates have no sane relationship to scrolling. When placing *child* in a gtk.TEXT_WINDOW_WIDGET window, scrolling is irrelevant, *child* floats above all scrollable areas. However, placing *child* in one of the scrollable windows (a border windows or the text window), you'll need to compute the correct position of *child* in textbuffer coordinates any time scrolling occurs or textbuffer changes occur, and then call the move_child() method to update the child's position. Unfortunately there's no good way to detect that scrolling has occurred, using the current API; a possible hack would be to update all child positions when the scroll adjustments change or the text buffer changes.

gtk.TextView.move_child

```
def move_child(child, xpos, ypos)
```

child :	a child widget in the textview
xpos :	the new X position in window coordinates
ypos :	the new Y position in window coordinates

The `move_child()` method moves the position of *child* to the location (in its current window) specified by *xpos* and *ypos*.

gtk.TextView.set_wrap_mode

```
def set_wrap_mode(wrap_mode)
```

wrap_mode :	a wrap mode value

The `set_wrap_mode()` method sets the "wrap–mode" property of the textview to the value specified by *wrap_mode*. The value of wrap_mode can be one of: `gtk.WRAP_NONE`, `gtk.WRAP_CHAR` or `gtk.WRAP_WORD`. The "wrap–mode" property defines the wrap mode for text that is not influenced by a gtk.TextTag that sets its "wrap_mode" attribute.

gtk.TextView.get_wrap_mode

```
def get_wrap_mode()
```

Returns :	the line wrap setting

The `get_wrap_mode()` method returns the value of the "wrap–mode" property. The value of "wrap–mode" can be one of: `gtk.WRAP_NONE`, `gtk.WRAP_CHAR` or `gtk.WRAP_WORD`. The "wrap–mode" property defines the wrap mode for text that is not influenced by a gtk.TextTag that sets its "wrap_mode" attribute.

gtk.TextView.set_editable

```
def set_editable(setting)
```

setting :	if `TRUE` the text is editable by default

The `set_editable()` method sets the "editable" property to the value of *setting*. If *setting* is `TRUE` the text in textview is editable by default. The "editable" property determines the editability of the gtk.TextView text that is not influenced by a gtk.TextTag that sets its "editable" attribute.

gtk.TextView.get_editable

```
def get_editable()
```

Returns :	*TRUE* if text is editable by default

The `get_editable()` method returns the value of the "editable" property. The "editable" property determines the editability of the gtk.TextView text that is not influenced by a gtk.TextTag that sets its "editable" attribute.

gtk.TextView.set_overwrite

```
    def set_overwrite(overwrite)
```

overwrite :	if TRUE turn on overwrite mode; if FALSE turn it off

Note

This method is available in PyGTK 2.4 and above.

The `set_overwrite()` method sets the "overwrite" property to the value of *overwrite*. If *overwrite* is TRUE, inserted text overwrites the existing text.

gtk.TextView.get_overwrite

```
    def get_overwrite()
```

Returns :	TRUE if the textview is in overwrite mode

Note

This method is available in PyGTK 2.4 and above.

The `get_overwrite()` method returns the value of the "overwrite" property. see the `set_overwrite()` method for more information.

gtk.TextView.set_accepts_tab

```
    def set_accepts_tab(accepts_tab)
```

accepts_tab :	if TRUE pressing the **Tab** key should insert a tab character; if FALSE, pressing the **Tab** key should move the keyboard focus.

Note

This method is available in PyGTK 2.4 and above.

The `set_accepts_tab()` method sets the "accepts_tab" property to the value of *accepts_tab*. If *accepts_tab* is TRUE a tab character is inserted. If *accepts_tab* is FALSE the keyboard focus is moved to the next widget in the focus chain.

gtk.TextView.get_accepts_tab

```
    def get_accepts_tab()
```

Returns :	TRUE if pressing the **Tab** key inserts a tab character, FALSE if pressing the **Tab** key moves the keyboard focus.

Note

This method is available in PyGTK 2.4 and above.

The `get_accepts_tab()` method returns the value of the "accepts_tab" property. See the `set_accepts_tab()` method for more information.

gtk.TextView.set_pixels_above_lines

```
    def set_pixels_above_lines(pixels_above_lines)
```

`pixels_above_lines` :	the number of pixels above paragraphs

The `set_pixels_above_lines()` method sets the "pixels–above–lines" property to the value of *`pixels_above_lines`*. The "pixels–above–lines" property determines the number of blank pixels to place above textbuffer lines (paragraphs) in the textview for text that is not influenced by a `gtk.TextTag` that sets its "pixels–above–lines" attribute.

gtk.TextView.get_pixels_above_lines

```
    def get_pixels_above_lines()
```

Returns :	the default number of pixels above paragraphs

The `get_pixels_above_lines()` method returns the value of the "pixels–above–lines" property. The "pixels–above–lines" property determines the number of pixels to put above textbuffer lines (paragraphs) in the textview for text that is not influenced by a `gtk.TextTag` that sets its "pixels–above–lines" attribute.

gtk.TextView.set_pixels_below_lines

```
    def set_pixels_below_lines(pixels_below_lines)
```

`pixels_below_lines` :	the default number of pixels below paragraphs

The `set_pixels_below_lines()` method sets the "pixels–below–lines" property to the value of *`pixels_below_lines`*. The "pixels–below–lines" property determines the number of blank pixels to place below textbuffer lines (paragraphs) in the textview for text that is not influenced by a `gtk.TextTag` that sets its "pixels–below–lines" attribute.

gtk.TextView.get_pixels_below_lines

```
    def get_pixels_below_lines()
```

Returns :	the default number of blank pixels below paragraphs

The `get_pixels_below_lines()` method returns the value of the "pixels–below–lines" property. The "pixels–below–lines" property determines the number of pixels to put below textbuffer lines (paragraphs) in the textview for text that is not influenced by a `gtk.TextTag` that sets its "pixels–below–lines" attribute.

gtk.TextView.set_pixels_inside_wrap

```
    def set_pixels_inside_wrap(pixels_inside_wrap)
```

`pixels_inside_wrap` :	the default number of pixels between wrapped lines

The `set_pixels_inside_wrap()` method sets the "pixels–inside_wrap" property to the value of *`pixels_inside_wrap`*. The "pixels–inside_wrap" property determines the number of blank pixels to place between wrapped textbuffer lines (inside paragraphs) for text that is not influenced by a `gtk.TextTag` that sets its "pixels–inside_wrap" attribute.

gtk.TextView.get_pixels_inside_wrap

```
    def get_pixels_inside_wrap()
```

Returns :	the default number of pixels of blank space between wrapped lines

The `get_pixels_inside_wrap()` method returns the value of the "pixels−inside−wrap" property. The "pixels−inside−wrap" property determines the number of pixels to put between wrapped textbuffer lines (inside paragraphs) for text that is not influenced by a `gtk.TextTag` that sets its "pixels−inside−wrap" attribute.

gtk.TextView.set_justification

```
def set_justification(justification)
```

`justification` :	the text justification

The `set_justification()` method sets the "justification" property to the value of *`justification`*. The value of justification must be one of: `gtk.JUSTIFY_LEFT`, `gtk.JUSTIFY_RIGHT`, `gtk.JUSTIFY_CENTER` or `gtk.JUSTIFY_FILL`. The "justification" property determines the justification of text in the textview that is not influenced by a `gtk.TextTag` that set its "justification" attribute.

gtk.TextView.get_justification

```
def get_justification()
```

Returns :	the default justification

The `get_justification()` method returns the value of the "justification" property. the default justification of paragraphs in *`text_view`*. The value of "justification" must be one of: `gtk.JUSTIFY_LEFT`, `gtk.JUSTIFY_RIGHT`, `gtk.JUSTIFY_CENTER` or `gtk.JUSTIFY_FILL`. The "justification" property determines the justification of text in the textview that is not influenced by a `gtk.TextTag` that set its "justification" attribute.

gtk.TextView.set_left_margin

```
def set_left_margin(left_margin)
```

`left_margin` :	the default left margin in pixels

The `set_left_margin()` method sets the "left−margin" property to the value of *`left_margin`*. The "left−margin"property determines the number of pixels of space for the left margin of text that is not influenced by a `gtk.TextTag` that sets its "left_margin" attribute.

gtk.TextView.get_left_margin

```
def get_left_margin()
```

Returns :	the default left margin in pixels

The `get_left_margin()` method returns the value of the "left_margin" property. The "left−margin"property determines the number of pixels of space for the left margin of text that is not influenced by a `gtk.TextTag` that sets its "left_margin" attribute.

gtk.TextView.set_right_margin

```
def set_right_margin(right_margin)
```

`right_margin` :	the default right margin in pixels

The `set_right_margin()` method sets the "right−margin" property to the value of *`right_margin`*. The "right−margin"property determines the number of pixels of space for the right margin of text that is not influenced by a gtk.TextTag that sets its "right_margin" attribute.

gtk.TextView.get_right_margin

```
def get_right_margin()
```

Returns :	the default right margin in pixels

The `get_right_margin()` method returns the value of the "right_margin" property. The "right–margin"property determines the number of pixels of space for the right margin of text that is not influenced by a gtk.TextTag that sets its "right_margin" attribute.

gtk.TextView.set_indent

```
def set_indent(indent)
```

indent :	the default indentation in pixels

The `set_indent()` method sets the "indent" property to the value of *indent*. The "indent" property determines the indentation for textview paragraphs that are not influenced by a gtk.TextTag that sets its "indent" attribute. The indentation may be negative.

gtk.TextView.get_indent

```
def get_indent()
```

Returns :	the default number of pixels of indentation

The `get_indent()` method returns the value of the "indent" property. The "indent" property determines the indentation for textview paragraphs that are not influenced by a gtk.TextTag that sets its "indent" attribute. The indentation may be negative.

gtk.TextView.set_tabs

```
def set_tabs(tabs)
```

tabs :	the default tabs as a pango.TabArray

The `set_tabs()` method sets the "tabs" property to a copy of the value of *tabs*. The "tabs" property contains the custom tab stops for the textview paragraphs that are not influenced by a gtk.TextTag that sets its "tabs" attribute.

gtk.TextView.get_tabs

```
def get_tabs()
```

Returns :	a copy of default tab array, or `None` if "standard" tabs are used

The `get_tabs()` method returns the value of the "tabs" property. The "tabs" property contains the custom tab stops for the textview paragraphs that are not influenced by a gtk.TextTag that sets its "tabs" attribute. The returned value will be `None` if "standard" (8–space) tabs are used.

gtk.TextView.get_default_attributes

```
def get_default_attributes()
```

Returns :	a new gtk.TextAttributes

The `get_default_attributes()` method returns a copy of the default `gtk.TextAttributes`. These attributes are used for text unless the text is influenced by a `gtk.TextTag`. You'd typically pass the default attributes in to the `gtk.TextIter.get_attributes()` method to get the attributes in effect at a given text position.

Signals

The "copy−clipboard" gtk.TextView Signal

```
def callback(textview, user_param1, ...)
```

textview :	the textview that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "copy−clipboard" signal is emitted when a selection is copied to the clipboard from *textview*.

The "cut−clipboard" gtk.TextView Signal

```
def callback(textview, user_param1, ...)
```

textview :	the textview that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "cut−clipboard" signal is emitted when a selection is cut from *textview* to the clipboard.

The "delete−from−cursor" gtk.TextView Signal

```
def callback(textview, delete_type, count, user_param1, ...)
```

textview :	the textview that received the signal
delete_type :	the type of deletion
count :	the number of deletions to do
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "delete−from−cursor" signal is emitted when a deletion of the type specified by *delete_type* is initiated by user action (e.g. pressing the **Delete** or **Backspace** keys). The value of *delete_type* must be one of:

- `gtk.DELETE_CHARS`
- `gtk.DELETE_WORD_ENDS`
- `gtk.DELETE_WORDS`
- `gtk.DELETE_DISPLAY_LINES`
- `gtk.DELETE_DISPLAY_LINE_ENDS`
- `gtk.DELETE_PARAGRAPH_ENDS`
- `gtk.DELETE_PARAGRAPHS`
- `gtk.DELETE_WHITESPACE`

count specifies the number of times that deletion should be applied.

The "insert−at−cursor" gtk.TextView Signal

```
def callback(textview, string, user_param1, ...)
```

textview :	the textview that received the signal
string :	the text to be inserted
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "insert−at−cursor" signal is emitted when *string* is being inserted in *textview*.

The "move−cursor" gtk.TextView Signal

```
def callback(textview, step_size, count, extend_selection, user_param1, ...)
```

textview :	the textview that received the signal
step_size :	the step size to move
count :	the number of steps to move
extend_selection :	if `TRUE` extend the selection
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "move−cursor" signal is emitted when the cursor is moved by *count* steps of *step_size*. If *extend_selection* is `TRUE` the selection is extended by the cursor movement. The value of step_size must be one of:

`gtk.MOVEMENT_LOGICAL_POSITIONS`	move by graphemes
`gtk.MOVEMENT_VISUAL_POSITIONS`	move by graphemes
`gtk.MOVEMENT_WORDS`	move by words
`gtk.MOVEMENT_DISPLAY_LINES`	move by lines(wrapped lines)
`gtk.MOVEMENT_DISPLAY_LINE_ENDS`	move to line ends(wrapped lines)
`gtk.MOVEMENT_PARAGRAPHS`	move by paragraphs(newline−ended lines)
`gtk.MOVEMENT_PARAGRAPH_ENDS`	move to ends of a paragraph
`gtk.MOVEMENT_PAGES`	move by pages
`gtk.MOVEMENT_BUFFER_ENDS`	move to ends of the buffer

The "move−focus" gtk.TextView Signal

```
def callback(textview, direction, user_param1, ...)
```

textview :	the textview that received the signal
direction :	the direction to move the focus
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "move−focus" signal is emitted when the focus moves from *textview* in the direction specified by *direction* that must be one of: `gtk.DIR_TAB_FORWARD`, `gtk.DIR_TAB_BACKWARD`, `gtk.DIR_UP`, `gtk.DIR_DOWN`, `gtk.DIR_LEFT` or `gtk.DIR_RIGHT`

The "move−viewport" gtk.TextView Signal

```
def callback(textview, scrollstep, count, user_param1, ...)
```

textview :	the textview that received the signal
scrollstep :	the size of the scroll step: `gtk.SCROLL_STEPS`, `gtk.SCROLL_PAGES`, `gtk.SCROLL_ENDS`, `gtk.SCROLL_HORIZONTAL_STEPS`, `gtk.SCROLL_HORIZONTAL_PAGES` or `gtk.SCROLL_HORIZONTAL_ENDS`
count :	the number of scroll steps of size *scrollstep* to take
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.4 and above.

The "move−viewport" signal is emitted when the viewport is being moved usually as the result of user action in moving the cursor or using the scrollbars.

The "page−horizontally" gtk.TextView Signal

```
def callback(textview, count, extend_selection, user_param1, ...)
```

textview :	the textview that received the signal
count :	the number of pages to move
extend_selection :	if `TRUE` extend the selection
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "page−horizontally" signal is emitted when user initiates horizontal paging by pressing one of the key combinations:

Control+Page Up	Page horizontally with *extend_selection* set to `FALSE`
Shift+Control+Page Up	Page horizontally with *extend_selection* set to `TRUE`

The "paste−clipboard" gtk.TextView Signal

```
def callback(textview, user_param1, ...)
```

textview :	the textview that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "paste−clipboard" signal is emitted when the contents of the clipboard are pasted into *textview*.

The "populate−popup" gtk.TextView Signal

```
def callback(textview, menu, user_param1, ...)
```

textview :	the textview that received the signal
menu :	the menu to populate
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "populate−popup" signal is emitted when the popup menu (specified by *menu*) associated with *textview* needs to be populated.

The "select−all" gtk.TextView Signal

```
def callback(textview, select, user_param1, ...)
```

textview :	the textview that received the signal
select :	if `TRUE` select the buffer contents; otherwise deselect the buffer contents
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.2 and above.

The "select−all" signal is emitted when the user presses one of:

- **Control+a** or **Control+/** to select all text in a buffer
- **Control+\** to deselect all text in a buffer

The "set−anchor" gtk.TextView Signal

```
def callback(textview, user_param1, ...)
```

textview :	the textview that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "set−anchor" signal is emitted when an application calls the `GObject.emit()`() method on textview with "set−anchor" as the signal.

The "set−scroll−adjustments" gtk.TextView Signal

```
def callback(textview, hadjustment, vadjustment, user_param1, ...)
```

textview :	the textview that received the signal
hadjustment :	the horizontal adjustment
vadjustment :	the vertical adjustment
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "set−scroll−adjustments" signal is emitted when one or both adjustments (specified by *hadjustment* and *vadjustment*) are set on *textview*.

The "toggle−overwrite" gtk.TextView Signal

```
def callback(textview, user_param1, ...)
```

textview :	the textview that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "toggle−overwrite" signal is emitted when the user presses the **Insert** key.

gtk.ToggleAction

gtk.ToggleAction — an action which can be toggled between two states (new in PyGTK 2.4)

Synopsis

```
class gtk.ToggleAction(gtk.Action):
    gtk.ToggleAction(name, label, tooltip, stock_id)
    def toggled()
    def set_active(is_active)
    def get_active()
    def set_draw_as_radio(draw_as_radio)
    def get_draw_as_radio()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Action
    +-- gtk.ToggleAction
```

Properties

"draw–as–radio"	Read–Write	If `TRUE`, the proxies for this action look like radio action proxies. Available in GTK+ 2.4 and above.

Signal Prototypes

"toggled"	`def callback(`*`toggleaction, user_param1, ...`*`)`

Description

Note

This object is available in PyGTK 2.4 and above.

A `gtk.ToggleAction` which is a subclass of `gtk.Action` corresponds roughly to a `gtk.CheckMenuItem`. It has an "active" state specifying whether the action has been checked or not.

Constructor

```
  gtk.ToggleAction(name, label, tooltip, stock_id)
```

name :	a unique name for the action

`label` :	the label displayed in menu items and on buttons
`tooltip` :	a tooltip for the action
`stock_id` :	the stock icon to display in widgets representing the action
Returns :	a new `gtk.ToggleAction`

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new `gtk.ToggleAction` object. To add the action to a `gtk.ActionGroup` and set the accelerator for the action, call the `gtk.ActionGroup.add_action_with_accel()` method.

Methods

gtk.ToggleAction.toggled

```
    def toggled()
```

Note

This method is available in PyGTK 2.4 and above.

The `toggled`() method emits the "toggled" signal on the toggle action.

gtk.ToggleAction.set_active

```
    def set_active(is_active)
```

`is_active` :	if `TRUE` the action should be checked

Note

This method is available in PyGTK 2.4 and above.

The `set_active`() method sets the checked state on the toggle action.

gtk.ToggleAction.get_active

```
    def get_active()
```

Returns :	`TRUE` if the toggle action is checked

Note

This method is available in PyGTK 2.4 and above.

The `get_active`() method returns `TRUE` if the toggle action is checked.

gtk.ToggleAction.set_draw_as_radio

```
    def set_draw_as_radio(draw_as_radio)
```

`draw_as_radio` :	if `TRUE` the action should have proxies like a radio action

Note

This method is available in PyGTK 2.4 and above.

The `set_draw_as_radio`() method sets the "draw–as–radio" property to the value of *`draw_as_radio`*. If *`draw_as_radio`* is `TRUE` the action should have proxies like a radio action.

gtk.ToggleAction.get_draw_as_radio

```
    def get_draw_as_radio()
```

Returns :	`TRUE` if the action should have proxies like a radio action.

Note

This method is available in PyGTK 2.4 and above.

The `get_draw_as_radio`() method returns the value of the "draw–as–radio" property. If "draw–as–radio" is `TRUE` the action should have proxies like a radio action.

Signals

The "toggled" gtk.ToggleAction Signal

```
    def callback(toggleaction, user_param1, ...)
```

`toggleaction` :	the toggleaction that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.4 and above.

The "toggled" signal is emitted when the toggle action changes its active state.

gtk.ToggleButton

gtk.ToggleButton — a button that retains its state

Synopsis

```
class gtk.ToggleButton(gtk.Button):
    gtk.ToggleButton(label=None, use_underline=TRUE)
    def set_mode(draw_indicator)
    def get_mode()
    def set_active(is_active)
    def get_active()
    def toggled()
    def set_inconsistent(setting)
    def get_inconsistent()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Button
            +-- gtk.ToggleButton
```

Properties

"active"	Read–Write	If `TRUE`, the toggle button should be pressed in.
"inconsistent"	Read–Write	If `TRUE`, the toggle button is in an "in between" state.
"draw–indicator"	Read–Write	If `TRUE`, the toggle part of the button is displayed.

Attributes

"draw_indicator"	Read	If `TRUE`, the toggle part of the button is displayed.

Signal Prototypes

"toggled"	`def callback(`*`togglebutton`*`, `*`user_param1`*`, ...)`

Description

A `gtk.ToggleButton` is a `gtk.Button` that has two stable states: 'pressed–in' ("on" or "active") and "normal" ("off" or "inactive"). The state changes to the alternate state each time the togglebutton is clicked. The state of a `gtk.ToggleButton` can be set specifically using the `set_active()` method, and retrieved using the `get_active()` method. To simply switch the state of a toggle button, use the `toggled()` method.

Constructor

```
    gtk.ToggleButton(label=None, use_underline=TRUE)
```

`label` :	the text to be displayed by the button label including an underscore to indicate the mnemonic character if desired or `None` if no label is required.
`use_underline` :	if `TRUE`, an underscore in the label text indicates the next character should be underlined and used for the mnemonic accelerator key if it is the first character so marked. Available in PyGTK 2.4 and above.
Returns :	a new `gtk.ToggleButton` widget

Creates a new `gtk.ToggleButton` widget with the text label specified by *`label`*. *`label`* will be parsed for underscore characters that indicate mnemonic accelerators. If *`label`* is `None` or not specified, no label will be created.

In PyGTK 2.4 and above the *`use_underline`* parameter is available and defaults to `TRUE`. If *`use_underline`* is set to `FALSE` the label text will not be parsed for mnemonic characters.

Methods

gtk.ToggleButton.set_mode

```
def set_mode(draw_indicator)
```

`draw_indicator` :	if `TRUE` display the button as an indicator with a label; if `FALSE` just display as a normal button

The `set_mode()` method sets the "draw−indicator" property to the value of *`draw_indicator`*. If *`draw_indicator`* is `TRUE` the button is displayed as an indicator with a label; otherwise, the button is displayed as a normal button.

This method only affects subclasses of `gtk.ToggleButton` like `gtk.CheckButton` and `gtk.RadioButton`.

gtk.ToggleButton.get_mode

```
def get_mode()
```

Returns :	`TRUE` if the button is displayed as an indicator with a label; `FALSE` if displayed as a normal button.

The `get_mode()` method returns the value of the "draw−indicator" property. If "draw−indicator" is `TRUE` the button is displayed as an indicator with a label; if `FALSE`, the button is displayed as a normal button. See the `set_mode()` method.

gtk.ToggleButton.set_active

```
def set_active(is_active)
```

`is_active` :	if `TRUE` the togglebutton state is active ("on")

The `set_active()` method sets the "active" property to the value of *`is_active`*. If *`is_active`* is `TRUE` the `gtk.ToggleButton` is 'pressed in', and if `FALSE` it's 'normal'. This method causes the "toggled" signal to be emitted.

gtk.ToggleButton.get_active

```
def get_active()
```

Returns :	TRUE if the togglebutton is active

The `get_active()` method returns the value of the "active" property. If "active" is `TRUE` the togglebutton is 'pressed in'; if `FALSE`, it's 'normal'.

gtk.ToggleButton.toggled

```
def toggled()
```

The `toggled()` method emits the "toggled" signal on the togglebutton.

gtk.ToggleButton.set_inconsistent

```
    def set_inconsistent(setting)
```

`setting` :	if `TRUE` the state is inconsistent

The `set_inconsistent()` method sets the "inconsistent" property to the value of *`setting`*. If *`setting`* is `TRUE` the togglebutton is displayed in the inconsistent state – an "in between" state. This method is useful if the user has selected a range of elements (such as some text or spreadsheet cells) that are affected by a toggle button, and the current values in that range are inconsistent and you want to indicate that by setting the toggle button to an "in between" display. Normally you would turn off the inconsistent state again if the user clicks the toggle button.

gtk.ToggleButton.get_inconsistent

```
    def get_inconsistent()
```

Returns :	`TRUE` if the state is inconsistent

The `get_inconsistent()` method returns the value of the "inconsistent" property. If "inconsistent" is `TRUE` the togglebutton is displayed in an 'in between' state. See the `set_inconsistent()` method for more details.

Signals

The "toggled" gtk.ToggleButton Signal

```
    def callback(togglebutton, user_param1, ...)
```

`togglebutton` :	the togglebutton that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "toggled" signal is emitted when the togglebutton state changes either programmatically or by user action.

gtk.ToggleToolButton

gtk.ToggleToolButton — A `gtk.ToolItem` containing a toggle button (new in PyGTK 2.4)

Synopsis

```
class gtk.ToggleToolButton(gtk.ToolButton):
    gtk.ToggleToolButton(stock_id=None)
    def set_active(is_active)
    def get_active()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.ToolItem
            +-- gtk.ToolButton
              +-- gtk.ToggleToolButton
```

Signal Prototypes

"toggled"	`def callback(toggletoolbutton, user_param1, ...)`

Description

Note

This widget is available in PyGTK 2.4 and above.

A `gtk.ToggleToolButton` is a `gtk.ToolItem` that contains a toggle button. Use the gtk.ToggleToolButton() constructor to create a new `gtk.ToggleToolButton`.

Constructor

```
  gtk.ToggleToolButton(stock_id=None)
```

`stock_id` :	the name of a stock item
Returns :	a newly created `gtk.ToggleToolButton`

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new `gtk.ToggleToolButton`. If *`stock_id`* is not `None` the toggle tool button contains the image and text from the specified stock item.

Methods

gtk.ToggleToolButton.set_active

```
  def set_active(is_active)
```

`is_active` :	if `TRUE` the toggle tool button is active

Note

This method is available in PyGTK 2.4 and above.

The `set_active()` method sets the status of the toggle tool button to the value specified by *`is_active`*. If *`is_active`* is `TRUE` the `gtk.ToggleButton` is 'pressed in' (active). This method causes the toggled signal to be emitted.

gtk.ToggleToolButton.get_active

```
    def get_active()
```

Returns :	`TRUE` if the toggle tool button is pressed in (active)

Note

This method is available in PyGTK 2.4 and above.

The `get_active()` method returns `TRUE` if the toggle tool button is pressed in (active) and `FALSE` if it is raised.

Signals

The "toggled" gtk.ToggleToolButton Signal

```
    def callback(togglетoolbutton, user_param1, ...)
```

`togglетoolbutton` :	the toggletoolbutton that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.4 and above.

The "toggled" signal is emitted when the toggle tool button changes state.

Prev | Up | Next
gtk.ToggleButton | Home | gtk.Toolbar

gtk.Toolbar
Prev **The gtk Class Reference** Next

gtk.Toolbar

gtk.Toolbar — a bar holding buttons and other widgets.

Synopsis

```
class gtk.Toolbar(gtk.Container):
    gtk.Toolbar()
    def insert(item, pos)
    def get_item_index(item)
    def get_n_items()
    def get_nth_item(n)
    def get_drop_index(x, y)
    def set_drop_highlight_item(tool_item, index)
    def set_show_arrow(show_arrow)
    def get_show_arrow()
    def get_relief_style()
    def append_item(text, tooltip_text, tooltip_private_text, icon, callback, user_data=None)
    def prepend_item(text, tooltip_text, tooltip_private_text, icon, callback, user_data)
    def insert_item(text, tooltip_text, tooltip_private_text, icon, callback, user_data, positi
    def insert_stock(stock_id, tooltip_text, tooltip_private_text, callback, user_data, positio
    def append_space()
    def prepend_space()
    def insert_space(position)
    def remove_space(position)
    def append_element(type, widget, text, tooltip_text, tooltip_private_text, icon, callback,
    def prepend_element(type, widget, text, tooltip_text, tooltip_private_text, icon, callback,
    def insert_element(type, widget, text, tooltip_text, tooltip_private_text, icon, callback,
    def append_widget(widget, tooltip_text, tooltip_private_text)
    def prepend_widget(widget, tooltip_text, tooltip_private_text)
    def insert_widget(widget, tooltip_text, tooltip_private_text, position)
    def set_orientation(orientation)
    def set_style(style)
    def set_icon_size(icon_size)
    def set_tooltips(enable)
    def unset_style()
    def unset_icon_size()
    def get_orientation()
    def get_style()
    def get_icon_size()
    def get_tooltips()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Toolbar
```

Properties

"orientation"	Read–Write	The orientation of the toolbar: `gtk.ORIENTATION_HORIZONTAL` or `gtk.ORIENTATION_VERTICAL`
"show–arrow"	Read–Write	If `TRUE` an arrow should be shown if the toolbar doesn't fit. Available in GTK+ 2.4 and above.
"toolbar–style"	Read–Write	The toolbar style: `gtk.TOOLBAR_ICONS`, `gtk.TOOLBAR_TEXT`, `gtk.TOOLBAR_BOTH` or `gtk.TOOLBAR_BOTH_HORIZ`

Child Properties

"expand"	Read–Write	If `TRUE`, the item should receive extra space when the toolbar grows. Available in GTK+ 2.4 and above.

Style Properties

"button–relief"	Read	The type of bevel around toolbar buttons: `gtk.RELIEF_NORMAL`, `gtk.RELIEF_HALF` or `gtk.RELIEF_NONE`
"internal–padding"	Read	The amount of border space between the toolbar shadow and the buttons
"shadow–type"	Read	The style of bevel around the toolbar: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN` or

		`gtk.SHADOW_ETCHED_OUT`
"space–size"	Read	The size of spacers
"space–style"	Read	The spacer style: `gtk.TOOLBAR_SPACE_EMPTY` or `gtk.TOOLBAR_SPACE_LINE`

Signal Prototypes

"orientation–changed"	`def callback(`*`toolbar`*`, `*`orientation`*`, `*`user_param1`*`, ...)`
"popup–context–menu"	`def callback(`*`toolbar`*`, `*`x`*`, `*`y`*`, `*`button`*`, `*`user_param1`*`, ...)`
"style–changed"	`def callback(`*`toolbar`*`, `*`style`*`, `*`user_param1`*`, ...)`

Description

A `gtk.Toolbar` is a subclass of `gtk.Container` that holds and manages a set of buttons and widgets in a horizontal or vertical bar. A `gtk.Toolbar` is usually used in an application as an alternative to a menu to provide a more direct means to activate dialogs or set options. Items in the toolbar can be visibly grouped by adding space between the elements. The toolbar style can be set to display only icons, only text or both icons and text. Each toolbar item has an associated `gtk.Tooltips` to provide a brief description of the items purpose.

In `PyGTK` 2.4 the interface of the `gtk.Toolbar` has changed to take advantage of the new `gtk.ToolItem` widgets. The following describes the new features.

A toolbar can contain instances of a subclass of `gtk.ToolItem` (`gtk.ToolButton`, `gtk.RadioToolButton`, `gtk.ToggleToolButton` and `gtk.SeparatorToolItem`). To add a `gtk.ToolItem` to the a toolbar, use the `insert()` method. To remove an item from the toolbar use the `gtk.Container.remove()` method. To add a button to the toolbar, add an instance of `gtk.ToolButton`. Toolbar items can be visually grouped by adding instances of `gtk.SeparatorToolItem` to the toolbar. If a `gtk.SeparatorToolItem` has the "expand" property set to `TRUE` and the "draw" property set to `FALSE` the effect is to force all following items to the end of the toolbar. Creating a context menu for the toolbar can be done by connecting to the "popup–context–menu" signal.

Constructor

```
  gtk.Toolbar()
```

Returns :	a new `gtk.Toolbar` widget

Creates a new `gtk.Toolbar` widget.

Methods

gtk.Toolbar.insert

```
  def insert(item, pos)
```

`item` :	a `gtk.ToolItem`

pos :	the position of the new item

Note

This method is available in PyGTK 2.4 and above

The `insert()` method inserts the `gtk.ToolItem` specified by *`item`* into the toolbar at the position specified by *`pos`*. If *`pos`* is 0 *`item`* is prepended to the start of the toolbar. If *`pos`* is negative, *`item`* is appended to the end of the toolbar.

gtk.Toolbar.get_item_index

```
    def get_item_index(item)
```

item :	a `gtk.ToolItem` that is a child of the toolbar
Returns :	the position of item on the toolbar.

Note

This method is available in PyGTK 2.4 and above

The `get_item_index()` method returns the position (starting from 0) on the toolbar of the `gtk.ToolItem` specified by *`item`*. It is an error if *`item`* is not a child of the toolbar.

gtk.Toolbar.get_n_items

```
    def get_n_items()
```

Returns :	the number of items on the toolbar

Note

This method is available in PyGTK 2.4 and above

The `get_n_items()` method returns the number of items on the toolbar.

gtk.Toolbar.get_nth_item

```
    def get_nth_item(n)
```

n :	a position on the toolbar
Returns :	The `gtk.ToolItem` on the toolbar at position *n*, or `None` if there isn't an item at position *n*

Note

This method is available in PyGTK 2.4 and above

The `get_nth_item()` method returns the toolbar `gtk.ToolItem` at the position specified by *`n`*, or `None` if the toolbar does not contain an item at position *`n`*.

gtk.Toolbar.get_drop_index

```
    def get_drop_index(x, y)
```

x :	the x coordinate of a point on the toolbar

y :	the y coordinate of a point on the toolbar
Returns :	The toolbar position corresponding to the point (*x*, *y*).

Note

This method is available in PyGTK 2.4 and above

The `get_drop_index`() method returns the position on the toolbar corresponding to the point specified by *x* and *y*. This is useful when dragging items to the toolbar. This method returns the position index where a new item should be inserted.

The *x* and *y* coordinates are relative to the toolbar.

gtk.Toolbar.set_drop_highlight_item

```
    def set_drop_highlight_item(tool_item, index)
```

tool_item :	a `gtk.ToolItem`, or `None` to turn off highlighting
index :	a position index on the toolbar

Note

This method is available in PyGTK 2.4 and above

The `set_drop_highlight_item`() method highlights the toolbar to give an idea of what it would look like if the `gtk.ToolItem` specified by *`tool_item`* was added at the position specified by *`index`*. If *`tool_item`* is `None`, highlighting is turned off and *`index`* is ignored.

The *`tool_item`* passed to this method must not be part of any widget hierarchy. When an item is set as drop highlight item it can not added to any widget hierarchy or used as highlight item for another toolbar.

gtk.Toolbar.set_show_arrow

```
    def set_show_arrow(show_arrow)
```

show_arrow :	if `TRUE`, show an arrow to indicate menu overflow

Note

This method is available in PyGTK 2.4 and above

The `set_show_arrow`() method sets the "show−arrow" property to the value of *`show_arrow`*. If *`show_arrow`* is `TRUE` an arrow is displayed (for an overflow menu) when the toolbar doesn't have room for all items on it. Items that are not displayed due to a lack of room are available through the overflow menu.

gtk.Toolbar.get_show_arrow

```
    def get_show_arrow()
```

Returns :	`TRUE` if an overflow menu can be used

Note

This method is available in PyGTK 2.4 and above

The `get_show_arrow()` method returns the value of the "show−arrow" property. If "show−arrow" is `TRUE` the toolbar has an overflow menu. See the `set_show_arrow()` method for more information.

gtk.Toolbar.get_relief_style

```
def get_relief_style()
```

Returns :	the relief style of buttons on the toolbar

Note

This method is available in PyGTK 2.4 and above

The `get_relief_style()` method returns the relief style of buttons on the toolbar. See the `gtk.Button.set_relief()` method for more information. The return value will be one of:

- `gtk.RELIEF_NORMAL`
- `gtk.RELIEF_HALF`
- `gtk.RELIEF_NONE`

gtk.Toolbar.append_item

```
def append_item(text, tooltip_text, tooltip_private_text, icon, callback, user_data=None)
```

`text` :	the text label or `None`
`tooltip_text` :	the tooltip text or `None`
`tooltip_private_text` :	the private tooltip text or `None`
`icon` :	a `gtk.Widget` or `None`
`callback` :	a callback function or method or `None`
`user_data` :	a user data object or `None`
Returns :	a `gtk.Button` widget

Warning

This method is deprecated in PyGTK 2.4 and above

The `append_item()` method adds a new `gtk.Button` to the end (right or bottom) of the toolbar with:

- the label specified by *`text`*,
- the `gtk.Tooltips` text and private text specified by *`tooltip_text`* and *`tooltip_private_text`* respectively and
- an icon (or any `gtk.Widget`) specified by *`icon`*.

A reference to the new button is returned. When the button is clicked the function or method specified by *`callback`* will be called with the user data specified by *`user_data`*. All or any of the arguments can have the value *`None`*.

gtk.Toolbar.prepend_item

```
def prepend_item(text, tooltip_text, tooltip_private_text, icon, callback, user_data)
```

`text` :	the text label or `None`
`tooltip_text` :	the tooltip text or `None`

`tooltip_private_text` :	the private tooltip text or `None`
`icon` :	a `gtk.Widget` or `None`
`callback` :	a callback function or method or `None`
`user_data` :	a user data object or `None`
Returns :	a `gtk.Button` widget

Warning

This method is deprecated in PyGTK 2.4 and above

The `prepend_item()` method adds a new `gtk.Button` to the beginning (left or top) of the toolbar with:

- the label specified by *`text`*,
- the `gtk.Tooltips` text and private text specified by *`tooltip_text`* and *`tooltip_private_text`* respectively and
- an icon (or any `gtk.Widget`) specified by *`icon`*.

A reference to the new button is returned. When the button is clicked the function or method specified by *`callback`* will be called with the user data specified by *`user_data`*. All or any of the arguments can have the value *`None`*.

gtk.Toolbar.insert_item

```
    def insert_item(text, tooltip_text, tooltip_private_text, icon, callback, user_data, positio
```

`text` :	the text label or `None`
`tooltip_text` :	the tooltip text or `None`
`tooltip_private_text` :	the private tooltip text or `None`
`icon` :	a `gtk.Widget` or `None`
`callback` :	a callback function or method or `None`
`user_data` :	a user data object or `None`
`position` :	The position to insert the button or −1 to append
Returns :	a `gtk.Button` widget

Warning

This method is deprecated in PyGTK 2.4 and above

The `insert_item()` method inserts a new `gtk.Button` the toolbar at the position specified by *`position`* with:

- the label specified by *`text`*,
- the `gtk.Tooltips` text and private text specified by *`tooltip_text`* and *`tooltip_private_text`* respectively and
- an icon (or any `gtk.Widget`) specified by *`icon`*.

A reference to the new button is returned. When the button is clicked the function or method specified by *`callback`* will be called with the user data specified by *`user_data`*. All or any of the arguments (except *`position`*) can have the value *`None`*. If *`position`* is negative the button will be appended to the toolbar.

gtk.Toolbar.insert_stock

```
def insert_stock(stock_id, tooltip_text, tooltip_private_text, callback, user_data, positio
```

stock_id :	the ID of the stock item to use as the button label and icon
tooltip_text :	the tooltip text or `None`
tooltip_private_text :	the private tooltip text or `None`
callback :	a callback function or method or `None`
user_data :	a user data object or `None`
position :	The position to insert the button or −1 to append
Returns :	a `gtk.Button` widget

Warning

This method is deprecated in PyGTK 2.4 and above

The `insert_stock`() method inserts a new `gtk.Button` the toolbar at the position specified by *`position`* with:

- the stock item specified by *`stock_id`* used for the label text and icon,
- the `gtk.Tooltips` text and private text specified by *`tooltip_text`* and *`tooltip_private_text`* respectively.

A reference to the new button is returned. When the button is clicked the function or method specified by *`callback`* will be called with the user data specified by *`user_data`*. All or any of the arguments (except *`position`*) can have the value *`None`*. If *`position`* is negative the button will be appended to the toolbar. If *`stock_id`* is not a known stock item ID, it's inserted verbatim, except that underscores are used to mark mnemonic accelerators.

gtk.Toolbar.append_space

```
def append_space()
```

Warning

This method is deprecated in PyGTK 2.4 and above

The `append_space`() method appends a space to the end of the toolbar.

gtk.Toolbar.prepend_space

```
def prepend_space()
```

Warning

This method is deprecated in PyGTK 2.4 and above

The `prepend_space`() method prepends a space to the beginning of the toolbar.

gtk.Toolbar.insert_space

```
def insert_space(position)
```

position :	The position to insert the space or –1 to append

Warning

This method is deprecated in PyGTK 2.4 and above

The `insert_space`() method inserts a space at the specified *`position`* in the toolbar.

gtk.Toolbar.remove_space

```
    def remove_space(position)
```

position :	the index of the space to remove.

Warning

This method is deprecated in PyGTK 2.4 and above

The `remove_space`() method removes a space from the specified *`position`*.

gtk.Toolbar.append_element

```
    def append_element(type, widget, text, tooltip_text, tooltip_private_text, icon, callback, u
```

type :	the type of *`widget`* – one of: `gtk.TOOLBAR_CHILD_SPACE`, `gtk.TOOLBAR_CHILD_BUTTON`, `gtk.TOOLBAR_CHILD_TOGGLEBUTTON`, `gtk.TOOLBAR_CHILD_RADIOBUTTON` or `gtk.TOOLBAR_CHILD_WIDGET`
widget :	a widget or `None`
text :	the text label or `None`
tooltip_text :	the tooltip text or `None`
tooltip_private_text :	the private tooltip text or `None`
icon :	a `gtk.Widget` or `None`
callback :	a callback function or method or `None`
user_data :	a user data object or `None`
Returns :	the new toolbar element as a `gtk.Widget`.

Warning

This method is deprecated in PyGTK 2.4 and above

The `append_element`() method adds a new element of the specified *`type`* to the end (right or bottom) of the toolbar with the `gtk.Tooltips` text and private text specified by *`tooltip_text`* and *`tooltip_private_text`* respectively. The behavior of the method depends on the type of element being added:

`gtk.TOOLBAR_CHILD_WIDGET`	The specified *`widget`* is the element added to the toolbar. The *`text`*, *`icon`*, *`callback`* and *`user_data`* arguments are ignored.
`gtk.TOOLBAR_CHILD_BUTTON`	The string specified by *`text`* and the `gtk.Widget` specified by *`icon`* are used to create the label for a `gtk.Button` to add to the toolbar. The function or method specified by *`callback`* and the object specified by *`user_data`* are connected to the

	button's "clicked" signal. The *widget* argument must have the value `None`.
`gtk.TOOLBAR_CHILD_TOGGLEBUTTON`	The string specified by *text* and the gtk.Widget specified by *icon* are used to create the label for a gtk.ToggleButton to add to the toolbar. The function or method specified by *callback* and the object specified by *user_data* are connected to the button's "clicked" signal. The *widget* argument must have the value `None`.
`gtk.TOOLBAR_CHILD_RADIOBUTTON`	The string specified by *text* and the gtk.Widget specified by *icon* are used to create the label for a gtk.RadioButton to add to the toolbar. The gtk.RadioButton specified by *widget* is used to set the group for the radiobutton. If *widget* is `None` a new radiobutton group is created. The function or method specified by *callback* and the object specified by *user_data* are connected to the button's "clicked" signal.
`gtk.TOOLBAR_CHILD_SPACE`	A space element is added to the toolbar. The *widget* argument must have the value `None`. The *text*, *icon*, *tooltip_text*, *tooltip_private_text*, *callback* and *user_data* arguments are ignored.

The *text*, *icon*, *callback*, *user_data*, *tooltip_text* and *tooltip_private_text* arguments may have the value *None*.

gtk.Toolbar.prepend_element

```
    def prepend_element(type, widget, text, tooltip_text, tooltip_private_text, icon, callback,
```

type :	the type of *widget* – one of: `gtk.TOOLBAR_CHILD_SPACE`, `gtk.TOOLBAR_CHILD_BUTTON`, `gtk.TOOLBAR_CHILD_TOGGLEBUTTON`, `gtk.TOOLBAR_CHILD_RADIOBUTTON` or `gtk.TOOLBAR_CHILD_WIDGET`
widget :	a widget or `None`
text :	the text label or `None`
tooltip_text :	the tooltip text or `None`
tooltip_private_text :	the private tooltip text or `None`
icon :	a gtk.Widget or `None`
callback :	a callback function or method or `None`
user_data :	a user data object or `None`
Returns :	the new toolbar element as a gtk.Widget.

Warning

This method is deprecated in PyGTK 2.4 and above

The `prepend_element()` method adds a new element of the specified *type* to the beginning (left or top) of the toolbar with the gtk.Tooltips text and private text specified by *tooltip_text* and *tooltip_private_text* respectively. The behavior of the method depends on the type of element being added:

`gtk.TOOLBAR_CHILD_WIDGET`	The widget specified by *widget* is the element added to the toolbar, otherwise *widget* should be `None`. The *text*, *icon*,

	callback and *user_data* arguments are ignored.
gtk.TOOLBAR_CHILD_BUTTON	The string specified by *text* and the gtk.Widget specified by *icon* are used to create the label for a gtk.Button to add to the toolbar. The function or method specified by *callback* and the object specified by *user_data* are connected to the button's "clicked" signal. The *widget* argument must have the value None.
gtk.TOOLBAR_CHILD_TOGGLEBUTTON	The string specified by *text* and the gtk.Widget specified by *icon* are used to create the label for a gtk.ToggleButton to add to the toolbar. The function or method specified by *callback* and the object specified by *user_data* are connected to the button's "clicked" signal. The *widget* argument must have the value None.
gtk.TOOLBAR_CHILD_RADIOBUTTON	The string specified by *text* and the gtk.Widget specified by *icon* are used to create the label for a gtk.RadioButton to add to the toolbar. The gtk.RadioButton specified by *widget* is used to set the group for the radiobutton. If *widget* is None a new radiobutton group is created. The function or method specified by *callback* and the object specified by *user_data* are connected to the button's "clicked" signal.
gtk.TOOLBAR_CHILD_SPACE	A space element is added to the toolbar. The *widget* argument must have the value None. The *text*, *icon*, *tooltip_text*, *tooltip_private_text*, *callback* and *user_data* arguments are ignored.

The *text*, *icon*, *callback*, *user_data*, *tooltip_text* and *tooltip_private_text* arguments may have the value *None*.

gtk.Toolbar.insert_element

```
    def insert_element(type, widget, text, tooltip_text, tooltip_private_text, icon, callback, u
```

type :	the type of *widget* – one of: gtk.TOOLBAR_CHILD_SPACE, gtk.TOOLBAR_CHILD_BUTTON, gtk.TOOLBAR_CHILD_TOGGLEBUTTON, gtk.TOOLBAR_CHILD_RADIOBUTTON or gtk.TOOLBAR_CHILD_WIDGET
widget :	a widget or None
text :	the text label or None
tooltip_text :	the tooltip text or None
tooltip_private_text :	the private tooltip text or None
icon :	a gtk.Widget or None
callback :	a callback function or method or None
user_data :	a user data object or None
position :	the position to insert the new element at.
Returns :	the new toolbar element as a gtk.Widget.

Warning

This method is deprecated in PyGTK 2.4 and above

The `insert_element()` method adds a new element of the specified *`type`* at the specified *`position`* in the toolbar with the `gtk.Tooltips` text and private text specified by *`tooltip_text`* and *`tooltip_private_text`* respectively. The behavior of the method depends on the type of element being added:

`gtk.TOOLBAR_CHILD_WIDGET`	The widget specified by *`widget`* is the element added to the toolbar, otherwise *`widget`* should be `None`. The *`text`*, *`icon`*, *`callback`* and *`user_data`* arguments are ignored.
`gtk.TOOLBAR_CHILD_BUTTON`	The string specified by *`text`* and the `gtk.Widget` specified by *`icon`* are used to create the label for a `gtk.Button` to add to the toolbar. The function or method specified by *`callback`* and the object specified by *`user_data`* are connected to the button's "clicked" signal. The *`widget`* argument must have the value `None`.
`gtk.TOOLBAR_CHILD_TOGGLEBUTTON`	The string specified by *`text`* and the `gtk.Widget` specified by *`icon`* are used to create the label for a `gtk.ToggleButton` to add to the toolbar. The function or method specified by *`callback`* and the object specified by *`user_data`* are connected to the button's "clicked" signal. The *`widget`* argument must have the value `None`.
`gtk.TOOLBAR_CHILD_RADIOBUTTON`	The string specified by *`text`* and the `gtk.Widget` specified by *`icon`* are used to create the label for a `gtk.RadioButton` to add to the toolbar. The `gtk.RadioButton` specified by *`widget`* is used to set the group for the radiobutton. If *`widget`* is `None` a new radiobutton group is created. The function or method specified by *`callback`* and the object specified by *`user_data`* are connected to the button's "clicked" signal.
`gtk.TOOLBAR_CHILD_SPACE`	A space element is added to the toolbar. The *`widget`* argument must have the value `None`. The *`text`*, *`icon`*, *`tooltip_text`*, *`tooltip_private_text`*, *`callback`* and *`user_data`* arguments are ignored.

The *`text`*, *`icon`*, *`callback`*, *`user_data`*, *`tooltip_text`* and *`tooltip_private_text`* arguments may have the value *`None`*.

gtk.Toolbar.append_widget

```
    def append_widget(widget, tooltip_text, tooltip_private_text)
```

`widget` :	a `gtk.Widget` to add to the toolbar.
`tooltip_text` :	the tooltip text or `None`
`tooltip_private_text` :	the private tooltip text or `None`

Warning

This method is deprecated in PyGTK 2.4 and above

The `append_widget()` method adds the specified *`widget`* to the end (right or bottom) of the toolbar. *`tooltip_text`* and *`tooltip_private_text`* specify the tooltip text and private text respectively.

gtk.Toolbar.prepend_widget

```
    def prepend_widget(widget, tooltip_text, tooltip_private_text)
```

widget :	a `gtk.Widget` to add to the toolbar.
tooltip_text :	the tooltip text or `None`
tooltip_private_text :	the private tooltip text or `None`

Warning

This method is deprecated in PyGTK 2.4 and above

The `prepend_widget()` method adds the specified *`widget`* to the start (left or top) of the toolbar. *`tooltip_text`* and *`tooltip_private_text`* specify the tooltip text and private text respectively.

gtk.Toolbar.insert_widget

```
    def insert_widget(widget, tooltip_text, tooltip_private_text, position)
```

widget :	a `gtk.Widget` to add to the toolbar.
tooltip_text :	the tooltip text or `None`
tooltip_private_text :	the private tooltip text or `None`
position :	the position to insert this widget at.

Warning

This method is deprecated in PyGTK 2.4 and above

The `insert_widget()` method adds the specified *`widget`* at the specified *`position`* in the toolbar. *`tooltip_text`* and *`tooltip_private_text`* specify the tooltip text and private text respectively.

gtk.Toolbar.set_orientation

```
    def set_orientation(orientation)
```

orientation :	the new orientation either `gtk.ORIENTATION_HORIZONTAL` or `gtk.ORIENTATION_VERTICAL`

The `set_orientation()` method sets the "orientation" property to the value of *`orientation`*. The value of orientation is either `gtk.ORIENTATION_HORIZONTAL` or `gtk.ORIENTATION_VERTICAL`

gtk.Toolbar.set_style

```
    def set_style(style)
```

style :	the new style – one of: `gtk.TOOLBAR_ICONS`, `gtk.TOOLBAR_TEXT`, `gtk.TOOLBAR_BOTH` or `gtk.TOOLBAR_BOTH_HORIZ`

The `set_style()` method sets the "toolbar–style" property to the value of *`style`*. The value of *`style`* must be one of the GTK Toolbar Style Constants. Setting the style overrides the user preferences for the toolbar style.

Note

A `gtk.ToolItem` label will not be displayed if the toolbar style is `gtk.TOOLBAR_BOTH_HORIZ` and the `gtk.ToolItem` "is–important" property is `FALSE` (the default). See the gtk.ToolItem.set_is_important() method for more information.

gtk.Toolbar.set_icon_size

```
def set_icon_size(icon_size)
```

icon_size :	The size of stock icons in the toolbar – one of: `gtk.ICON_SIZE_MENU`, `gtk.ICON_SIZE_SMALL_TOOLBAR`, `gtk.ICON_SIZE_LARGE_TOOLBAR`, `gtk.ICON_SIZE_BUTTON`, `gtk.ICON_SIZE_DND` or `gtk.ICON_SIZE_DIALOG`

Warning

This method is deprecated in PyGTK 2.4 and above

The `set_icon_size`() method sets the size of stock icons in the toolbar to the value specified by *icon_size*. The value of *icon_size* must be one of:

- `gtk.ICON_SIZE_MENU`
- `gtk.ICON_SIZE_SMALL_TOOLBAR`
- `gtk.ICON_SIZE_LARGE_TOOLBAR`
- `gtk.ICON_SIZE_BUTTON`
- `gtk.ICON_SIZE_DND`, or
- `gtk.ICON_SIZE_DIALOG`

This method can be called both before and after adding the icons. Setting the icon size will override the user preferences for the default icon size.

gtk.Toolbar.set_tooltips

```
def set_tooltips(enable)
```

enable :	if `TRUE` tooltips should be used

The `set_tooltips`() method enables or disables tooltips for the toolbar depending on the value of *enable*. If *enable* is `TRUE`, tooltips will be used.

gtk.Toolbar.unset_style

```
def unset_style()
```

The `unset_style`() method unsets a toolbar style set with the set_style() method, allowing the user preferences to determine the toolbar style.

gtk.Toolbar.unset_icon_size

```
def unset_icon_size()
```

Warning

This method is deprecated in PyGTK 2.4 and above

The `unset_icon_size`() method unsets toolbar icon size set with the set_icon_size(), allowing the user preferences to determine the icon size.

gtk.Toolbar.get_orientation

```
    def get_orientation()
```

Returns :	the orientation

The `get_orientation()` method returns the value of the "orientation" property that determines the current orientation of the toolbar. See the `set_orientation()` method for more details.

gtk.Toolbar.get_style

```
    def get_style()
```

Returns :	the current toolbar style

The `get_style()` method returns the value of the "toolbar−style" property. See the `set_style()` method for more details.

gtk.Toolbar.get_icon_size

```
    def get_icon_size()
```

Returns :	the current icon size for the icons on the toolbar.

The `get_icon_size()` method returns the current icon size for the toolbar. See the `set_icon_size()` method for more details.

gtk.Toolbar.get_tooltips

```
    def get_tooltips()
```

Returns :	`TRUE` if tooltips are enabled

The `get_tooltips()` method returns `TRUE` if tooltips are enabled. See the `set_tooltips()` method for more details.

Signals

The "orientation−changed" gtk.Toolbar Signal

```
    def callback(toolbar, orientation, user_param1, ...)
```

`toolbar` :	the toolbar that received the signal
`orientation` :	the new orientation
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "orientation−changed" signal is emitted when the *`orientation`* of *`toolbar`* is changed.

The "popup−context−menu" gtk.Toolbar Signal

```
    def callback(toolbar, x, y, button, user_param1, ...)
```

`toolbar` :	the toolbar that received the signal
`x` :	the x coordinate of the mouse event

y :	the u coordinate of the mouse event
button :	the number of the mouse button
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled

Note

This signal is available in GTK+ 2.4 and above.

The "popup−context−menu" signal is emitted when the user right−clicks the toolbar or uses the keybinding to display a popup menu. Application developers should handle this signal if they want to display a context menu on the toolbar. The context−menu should appear at the coordinates given by *x* and *y*. The mouse button number is given by the *button* parameter. If the menu was popped up using the keyboard, *button* is −1.

The "style−changed" gtk.Toolbar Signal

```
  def callback(toolbar, style, user_param1, ...)
```

toolbar :	the toolbar that received the signal
style :	the new style
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "style−changed" signal is emitted when the *style* of *toolbar* is changed.

Prev Up Next
gtk.ToggleToolButton Home gtk.ToolButton
gtk.ToolButton
Prev The gtk Class Reference Next

gtk.ToolButton

gtk.ToolButton a `gtk.ToolItem` subclass that displays buttons (new in PyGTK 2.4)

Synopsis

```
class gtk.ToolButton(gtk.ToolItem):
    gtk.ToolButton(icon_widget=None, label=None)
    gtk.ToolButton(stock_id)
    def set_label(label)
    def get_label()
    def set_use_underline(use_underline)
    def get_use_underline()
    def set_stock_id(stock_id)
    def get_stock_id()
    def set_icon_widget(icon_widget)
    def get_icon_widget()
    def set_label_widget(label_widget)
    def get_label_widget()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.ToolItem
            +-- gtk.ToolButton
```

Properties

"icon–widget"	Read–Write	The icon widget to display in the item.
"label"	Read–Write	The text to show in the item.
"label–widget"	Read–Write	The widget to use as the item label instead of "label".
"stock–id"	Read–Write	The stock icon displayed on the item if "label" and "label–widget" are None.
"use–underline"	Read–Write	If `TRUE`, an underline in the "label" property indicates that the next character should be used for the mnemonic accelerator key in the overflow menu.

Signal Prototypes

```
"clicked"            def callback(toolbutton, user_param1, ...)
```

Description

Note

This widget is available in PyGTK 2.4 and above.

A `gtk.ToolButton` is a sub class of `gtk.ToolItem` that contains a button. Use the gtk.ToolButton() constructor to create a new `gtk.ToolButton` specifying a widget to use as the icon and a label for the text. Alternatively use the other gtk.ToolButton() constructor to create a `gtk.ToolButton` from a stock item.

The label of a `gtk.ToolButton` is determined by the properties "label_widget", "label", and "stock_id". If "label_widget" specifies a `gtk.Widget`, that widget is used as the label. If "label–widget" is `None`, then the string in "label" is used as the label. If both "label–widget" and "label" are `None`, the label is determined by the stock item specified by "stock–id". Finally, if "label–widget", "label" and "stock–id" are all `None`, the button does not have a label.

The icon of a `gtk.ToolButton` is determined by the properties "icon–widget" and "stock–id". If "icon–widget" specifies a `gtk.Widget`, that widget is used as the icon. If "icon–widget" is `None`, the icon is determined by the stock item specified by "stock–id". If both "icon–widget" and "stock–id" are `None`, the button does not have an icon.

Constructor

gtk.ToolButton

```
  gtk.ToolButton(icon_widget=None, label=None)
```

`icon_widget` :	a `gtk.Widget` that will be used as the icon widget, or `None`
`label` :	a string that will be used as the label, or `None`
Returns :	A new `gtk.ToolButton`

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new `gtk.ToolButton` optionally using the icon specified by *`icon_widget`* and the label text specified by *`label`*. If both *`icon_widget`* and *`label`* are `None`, the tool button will be empty.

gtk.ToolButton

```
    gtk.ToolButton(stock_id)
```

`stock_id` :	a string that specifies a stock item
Returns :	A new `gtk.ToolButton`

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new `gtk.ToolButton` using the stock item specified by *`stock_id`* to determine the icon and label text. It is an error if *`stock_id`* is not a name of a stock item.

Methods

gtk.ToolButton.set_label

```
    def set_label(label)
```

`label` :	a string that will be used as label, or `None`.

Note

This method is available in PyGTK 2.4 and above.

The `set_label`() method sets the "label" property to the value of *`label`*. If the "label_widget" property is `None`, label will be used as the label of the tool button.

gtk.ToolButton.get_label

```
    def get_label()
```

Returns :	The label, or `None`

Note

This method is available in PyGTK 2.4 and above.

The `get_label`() method returns the value of the "label" property that is used as the label of the tool button if the "label–widget" property is `None`.

gtk.ToolButton.set_use_underline

```
    def set_use_underline(use_underline)
```

use_underline :	if TRUE, an underline in the label string specifies a mnemonic key for the overflow menu

Note

This method is available in PyGTK 2.4 and above.

The set_use_underline() method sets the "use−underline" property to the value of *use_underline*. If *use_underline* is TRUE, an underline in the "label" property indicates that the next character should be used for the mnemonic accelerator key in the overflow menu. For example, if the label property is "_Open" and *use_underline* is TRUE, the label on the tool button will be "Open" and the item on the overflow menu will have an underlined 'O'. Labels shown on tool buttons never have mnemonics on them; this property only affects the menu item on the overflow menu.

gtk.ToolButton.get_use_underline

```
    def get_use_underline()
```

Returns :	TRUE if underscores in the "label" property are used as mnemonics on menu items on the overflow menu.

Note

This method is available in PyGTK 2.4 and above.

The get_use_underline() method returns the value of the "use−underline" property. If "use−underline" is TRUE, underscores in the label property are used as mnemonics on menu items on the overflow menu. See the set_use_underline() method for more information.

gtk.ToolButton.set_stock_id

```
    def set_stock_id(stock_id)
```

stock_id :	a name of a stock item, or None

Note

This method is available in PyGTK 2.4 and above.

The set_stock_id() method sets the "stock_id" property to the value of *stock_id*. The stock item specified by stock_id is used to determine the icon and label if not overridden by the "label" and "icon_widget" properties. See the gtk.ToolButton() constructor for more information.

gtk.ToolButton.get_stock_id

```
    def get_stock_id()
```

Returns :	the name of the stock item.

Note

This method is available in PyGTK 2.4 and above.

The `get_stock_id()` method returns the value of the "stock–id" property that contains the name of a stock item or `None`. See the gtk.ToolButton() constructor for more information.

gtk.ToolButton.set_icon_widget

```
    def set_icon_widget(icon_widget)
```

`icon_widget` :	the widget used as icon, or `None`

Note

This method is available in PyGTK 2.4 and above.

The `set_icon_widget()` method sets the "icon–widget" property to the value of *`icon_widget`*. If *`icon_widget`* specifies a `gtk.Widget`, it is used as the icon of the tool button. If *`icon_widget`* is `None` the icon is determined by the "stock_id" property. If the "stock_id" property is also `None`, the tool button will not have an icon.

gtk.ToolButton.get_icon_widget

```
    def get_icon_widget()
```

Returns :	The widget used as icon on *`button`*, or `None`.

Note

This method is available in PyGTK 2.4 and above.

The `get_icon_widget()` method returns the value of the "icon–widget" property that contains the `gtk.Widget` used as the icon on the tool button. See the `set_icon_widget()` method for more information.

gtk.ToolButton.set_label_widget

```
    def set_label_widget(label_widget)
```

`label_widget` :	the widget used as the label, or `None`

Note

This method is available in PyGTK 2.4 and above.

The `set_label_widget()` method sets the "label–widget" property to the `gtk.Widget` specified by *`label_widget`* that will be used as the label for the tool button. If *`label_widget`* is `None` the "label" property is used as label. If "label–widget" and "label" are both `None`, the label in the stock item determined by the "stock_id" property is used as the label. If "label–widget", "label" and "stock_id" are all `None`, the tool button will not have a label.

gtk.ToolButton.get_label_widget

```
    def get_label_widget()
```

Returns :	The widget used as label on *button*, or None.

Note

This method is available in PyGTK 2.4 and above.

The `get_label_widget`() method returns the value of the "label−widget" property that is used as the label on the tool button. See the `gtk.ToolButton.set_label_widget()` method for more information.

Signals

The "clicked" gtk.ToolButton Signal

```
    def callback(toolbutton, user_param1, ...)
```

toolbutton :	the toolbutton that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "clicked" signal is emitted when the tool button is clicked with the mouse or activated with the keyboard.

gtk.ToolItem

gtk.ToolItem — the base class of widgets that can be added to `gtk.Toolbar` (new in PyGTK 2.4)

Synopsis

```
class gtk.ToolItem(gtk.Bin):
    gtk.ToolItem()
    def set_homogeneous(homogeneous)
    def get_homogeneous()
    def set_expand(expand)
    def get_expand()
    def set_tooltip(tooltips, tip_text=None, tip_private=None)
    def set_use_drag_window(use_drag_window)
    def get_use_drag_window()
    def set_visible_horizontal(visible_horizontal)
    def get_visible_horizontal()
    def set_visible_vertical(visible_vertical)
    def get_visible_vertical()
    def set_is_important(is_important)
    def get_is_important()
    def get_icon_size()
    def get_orientation()
    def get_toolbar_style()
    def get_relief_style()
    def retrieve_proxy_menu_item()
    def set_proxy_menu_item(menu_item_id, menu_item)
    def get_proxy_menu_item(menu_item_id)
    def rebuild_menu()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.ToolItem
```

Properties

Note

These properties are available in GTK+ 2.4 and above.

"is–important"	Read–Write	If `TRUE`, the toolbar item is considered important and the toolbar buttons show text in `gtk.TOOLBAR_BOTH_HORIZ` mode. Default value: `FALSE`
"visible–horizontal"	Read–Write	If `TRUE`, the toolbar item is visible when the toolbar is in a horizontal orientation. Default value: `TRUE`
"visible–vertical"	Read–Write	If `TRUE`, the toolbar item is visible when the toolbar is in a vertical orientation. Default value: `TRUE`

Signal Prototypes

"create–menu–proxy"	`def callback(`*`toolitem, user_param1, ...`*`)`
"set–tooltip"	`def callback(`*`toolitem, tooltips, tip_text, tip_private, user_param1, ...`*`)`
"toolbar–reconfigured"	`def callback(`*`toolitem, user_param1, ...`*`)`

Description

Note

This widget is available in PyGTK 2.4 and above.

A `gtk.ToolItem` is a widget that can appear on a `gtk.Toolbar`. To create a toolbar item that contains something else than a button, use the gtk.ToolItem() constructor then use the `gtk.Container.add()` method to add a child widget to the tool item.

To create and use toolbar items that contain buttons, see the `gtk.ToolButton`, `gtk.ToggleToolButton` and `gtk.RadioToolButton` classes. See the `gtk.Toolbar` class for a description of the toolbar widget.

Constructor

```
gtk.ToolItem()
```

Returns :	the new `gtk.ToolItem`

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new empty `gtk.ToolItem`

Methods

gtk.ToolItem.set_homogeneous

```
def set_homogeneous(homogeneous)
```

`homogeneous` :	if `TRUE` the tool item is the same size as other homogeneous items

Note

This method is available in PyGTK 2.4 and above.

The `set_homogeneous()` method sets the homogeneous setting of the tool item to the value of *`homogeneous`*. If *`homogeneous`* is `TRUE` the tool item is to be allocated the same size as other homogeneous items. The effect is that all homogeneous items will have the same width as the widest of the items.

gtk.ToolItem.get_homogeneous

```
def get_homogeneous()
```

Returns :	`TRUE` if the item is the same size as other homogeneous items.

Note

This method is available in PyGTK 2.4 and above.

The `get_homogeneous()` method returns the setting of the homogeneous setting of the tool item. If `TRUE` the tool item is the same size as other homogeneous items. See the `set_homogeneous()` method for more detail.

gtk.ToolItem.set_expand

```
def set_expand(expand)
```

`expand` :	If `TRUE` the tool item is allocated extra space when available

Note

This method is available in PyGTK 2.4 and above.

The `set_expand()` method sets the expand setting of the tool item to the value of *`expand`*. If *`expand`* is `TRUE` the tool item is allocated extra space when there is more room on the toolbar than needed for the items.

The effect is that the item gets bigger when the toolbar gets bigger and smaller when the toolbar gets smaller.

gtk.ToolItem.get_expand

```
def get_expand()
```

Returns :	`TRUE` if the tool item is allocated extra space when available.

Note

This method is available in PyGTK 2.4 and above.

The `get_expand()` method returns the value of the expand setting of the tool item. If `TRUE` the tool item is allocated extra space. See the `gtk.ToolItem.set_expand()` method for more detail.

gtk.ToolItem.set_tooltip

```
def set_tooltip(tooltips, tip_text=None, tip_private=None)
```

`tooltips` :	The `gtk.Tooltips` object to be used
`tip_text` :	the text to be used as tooltip text for the tool item or `None`
`tip_private` :	the text to be used as private tooltip text or `None`

Note

This method is available in PyGTK 2.4 and above.

The `set_tooltip()` method sets the `gtk.Tooltips` object specified by *`tooltips`* to be used for the tool item with the tooltip text specified by *`tip_text`* and the private text specified by *`tip_private`*. See the `gtk.Tooltips.set_tip()` method for more information.

gtk.ToolItem.set_use_drag_window

```
def set_use_drag_window(use_drag_window)
```

`use_drag_window` :	if `TRUE` the tool item has a drag window.

Note

This method is available in PyGTK 2.4 and above.

The `set_use_drag_window()` method determines whether the tool item has a drag window according to the value of *`use_drag_window`*. If *`use_drag_window`* is `TRUE` the toolitem can be used as a drag source through the `gtk.Widget.drag_source_set()` method. When the tool item has a drag window it will intercept all events, even those that would otherwise be sent to a child of the tool item

gtk.ToolItem.get_use_drag_window

```
def get_use_drag_window()
```

Returns :	`TRUE` if the tool item uses a drag window.

Note

This method is available in PyGTK 2.4 and above.

The `get_use_drag_window()` returns the setting that determines if the tool item has a drag window. See the `set_use_drag_window()` for more information.

gtk.ToolItem.set_visible_horizontal

```
def set_visible_horizontal(visible_horizontal)
```

`visible_horizontal` :	if `TRUE` the tool item is visible when in horizontal mode

Note

This method is available in PyGTK 2.4 and above.

The `set_visible_horizontal()` method sets the "visible−horizontal" property to the value of *`visible_horizontal`*. If *`visible_horizontal`* is `TRUE`, the tool item is visible when the toolbar is docked horizontally.

gtk.ToolItem.get_visible_horizontal

```
def get_visible_horizontal()
```

Returns :	`TRUE` if the tool item is visible on toolbars that are docked horizontally.

Note

This method is available in PyGTK 2.4 and above.

The `get_visible_horizontal()` method returns the value of the "visible−horizontal" property. If "visible−horizontal" is `TRUE`, the tool item is visible on toolbars that are docked horizontally.

gtk.ToolItem.set_visible_vertical

```
def set_visible_vertical(visible_vertical)
```

`visible_vertical` :	if `TRUE`, the tool item is visible when the toolbar is in vertical mode

Note

This method is available in PyGTK 2.4 and above.

The `set_visible_vertical()` method sets the "visible−vertical" property to the value of *`visible_vertical`*. If *`visible_vertical`* is `TRUE`, the tool item is visible when the toolbar is docked vertically. Some tool items, such as text entries, are too wide to be useful on a vertically docked toolbar. If *`visible_vertical`* is `FALSE` the tool item will not appear on toolbars that are docked vertically.

gtk.ToolItem.get_visible_vertical

```
def get_visible_vertical()
```

Returns :	TRUE if the tool item is visible when the toolbar is docked vertically

Note

This method is available in PyGTK 2.4 and above.

The `get_visible_vertical()` method returns the value of the "visible−vertical" property. If "visible−vertical" is `TRUE`, the tool item is visible when the toolbar is docked vertically. See the `set_visible_vertical()` method for more information.

gtk.ToolItem.set_is_important

```
    def set_is_important(is_important)
```

`is_important` :	if `TRUE`, the tool item should be considered important

Note

This method is available in PyGTK 2.4 and above.

The `set_is_important()` method sets the "is−important" property to the value of *`is_important`*. If *`is_important`* is `TRUE` the tool item should be considered important. The `gtk.ToolButton` class uses this property to determine whether to show its label when the toolbar style is `gtk.TOOLBAR_BOTH_HORIZ`. The result is that only tool buttons with the "is_important" property set have labels, an effect known as "priority text".

gtk.ToolItem.get_is_important

```
    def get_is_important()
```

Returns :	`TRUE` if the tool item is considered important.

Note

This method is available in PyGTK 2.4 and above.

The `get_is_important()` method returns the value of the "is−important" property. If "is−important" is `TRUE`, the tool item is considered important. See the `set_is_important()` method for more information.

gtk.ToolItem.get_icon_size

```
    def get_icon_size()
```

Returns :	the icon size used for the tool item

Note

This method is available in PyGTK 2.4 and above.

The `get_icon_size()` method returns the icon size used for the tool item. Custom subclasses of `gtk.ToolItem` should call this method to find out what size icons they should use. The return value should be one of: `gtk.ICON_SIZE_MENU`, `gtk.ICON_SIZE_SMALL_TOOLBAR`, `gtk.ICON_SIZE_LARGE_TOOLBAR`, `gtk.ICON_SIZE_BUTTON`, `gtk.ICON_SIZE_DND`, `gtk.ICON_SIZE_DIALOG` or an integer value returned from the `gtk.icon_size_register()` function.

gtk.ToolItem.get_orientation

```
def get_orientation()
```

Returns :	the orientation used for the tool item

Note

This method is available in PyGTK 2.4 and above.

The `get_orientation`() method returns the orientation used for the tool item. Custom subclasses of `gtk.ToolItem` should call this method to find out what size icons they should use. The return value should be either `gtk.ORIENTATION_HORIZONTAL` or `gtk.ORIENTATION_VERTICAL`.

gtk.ToolItem.get_toolbar_style

```
def get_toolbar_style()
```

Returns :	the toolbar style used for the tool item

Note

This method is available in PyGTK 2.4 and above.

The `get_toolbar_style`() method returns the toolbar style used for the tool item. Custom subclasses of `gtk.ToolItem` should call this method in the "toolbar–reconfigured" signal handler to find out in what style the toolbar is displayed and change themselves accordingly.

Possibilities are:

- `gtk.TOOLBAR_BOTH`, meaning the tool item should show both an icon and a label, stacked vertically
- `gtk.TOOLBAR_ICONS`, meaning the toolbar shows only icons
- `gtk.TOOLBAR_TEXT`, meaning the tool item should only show text
- `gtk.TOOLBAR_BOTH_HORIZ`, meaning the tool item should show both an icon and a label, arranged horizontally..

gtk.ToolItem.get_relief_style

```
def get_relief_style()
```

Returns :	the relief style used for the tool item

Note

This method is available in PyGTK 2.4 and above.

The `get_relief_style`() method returns the relief style of the tool item. See the `gtk.Button.set_relief()` method for more information. Custom subclasses of `gtk.ToolItem` should call this method in the handler of the `gtk.ToolItem` "toolbar–reconfigured" signal to find out the relief style of buttons.

The return value should be one of: `gtk.RELIEF_NORMAL`, `gtk.RELIEF_HALF` or `gtk.RELIEF_NONE`.

gtk.ToolItem.retrieve_proxy_menu_item

```
def retrieve_proxy_menu_item()
```

Returns :	The `gtk.MenuItem` that is going to appear in the overflow menu for the tool item

Note

This method is available in PyGTK 2.4 and above.

The `retrieve_proxy_menu_item()` method returns the `gtk.MenuItem` that was last set by the `set_proxy_menu_item()` method, i.e. the `gtk.MenuItem` that is going to appear in the overflow menu.

gtk.ToolItem.set_proxy_menu_item

```
def set_proxy_menu_item(menu_item_id, menu_item)
```

`menu_item_id` :	a string used to identify *`menu_item`*
`menu_item` :	a `gtk.MenuItem` to be used in the overflow menu or `None`

Note

This method is available in PyGTK 2.4 and above.

The `set_proxy_menu_item()` method sets the `gtk.MenuItem` specified by *`menu_item`* to be used in the toolbar overflow menu. *`menu_item_id`* is used to identify the caller of this method and should also be used with the `get_proxy_menu_item()` method. If *`menu_item`* is `None` the tool item will not appear in the overflow menu.

gtk.ToolItem.get_proxy_menu_item

```
def get_proxy_menu_item(menu_item_id)
```

`menu_item_id` :	a string used to identify the menu item
Returns :	The `gtk.MenuItem` matching *`menu_item_id`*.

Note

This method is available in PyGTK 2.4 and above.

The `get_proxy_menu_item()` method returns the `gtk.MenuItem` corresponding to the string specified by *`menu_item_id`* as passed to the `set_proxy_menu_item()` method.

Custom subclasses of `gtk.ToolItem` should use this method to update their menu item when the `gtk.ToolItem` changes. Forcing a match with *`menu_item_id`* ensures that a `gtk.ToolItem` will not inadvertently change a menu item that they did not create.

gtk.ToolItem.rebuild_menu

```
def rebuild_menu()
```

Note

This method is available in PyGTK 2.6 and above.

The `rebuild_menu`() method ignals to the toolbar that the overflow menu item has changed. If the overflow menu is visible when this method it called, the menu will be rebuilt. The method must be called when the tool item changes what it will do in response to the "create_menu_proxy" signal.

Signals

The "create−menu−proxy" gtk.ToolItem Signal

```
def callback(toolitem, user_param1, ...)
```

toolitem :	the toolitem that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled

Note

This signal is available in GTK+ 2.4 and above.

The "create−menu−proxy" signal is emitted when the toolbar is displaying an overflow menu and is trying to determine if *toolitem* should appear in the overflow menu. In response *toolitem* should either

- call the `set_proxy_menu_item()` method specifying *menu_item* as `None` and return `TRUE` to indicate that the item should not appear in the overflow menu
- call the `set_proxy_menu_item()` method with a new menu item and return `TRUE`, or
- return `FALSE` to indicate that the signal was not handled by the item. This means that the item will not appear in the overflow menu unless a later handler installs a menu item.

The toolbar may cache the result of this signal. When the tool item changes how it will respond to this signal it must call the `rebuild_menu()`) method to invalidate the cache and ensure that the toolbar rebuilds its overflow menu.

The "set−tooltip" gtk.ToolItem Signal

```
def callback(toolitem, tooltips, tip_text, tip_private, user_param1, ...)
```

toolitem :	the toolitem that received the signal
tooltips :	the `gtk.Tooltips`
tip_text :	the tooltip text
tip_private :	the tooltip private text
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled

Note

This signal is available in GTK+ 2.4 and above.

The "set−tooltip" signal is emitted when the tool item's tooltip changes. Application developers can use the `set_tooltip()` method to set the item's tooltip.

The "toolbar−reconfigured" gtk.ToolItem Signal

```
    def callback(toolitem, user_param1, ...)
```

toolitem :	the toolitem that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.4 and above.

The "toolbar−reconfigured" signal is emitted when some property of the toolbar that the item is a child of changes. For custom subclasses of `gtk.ToolItem`, the default handler of this signal uses the methods:

- `gtk.Toolbar.get_orientation()`
- `gtk.Toolbar.get_style()`
- `gtk.Toolbar.get_icon_size()`
- `gtk.Toolbar.get_relief_style()`

to find out what the toolbar should look like and change themselves accordingly.

Prev Up Next
gtk.ToolButton Home gtk.Tooltips

gtk.Tooltips

gtk.Tooltips — add tips to your widgets.

Synopsis

```
class gtk.Tooltips(gtk.Object):
    gtk.Tooltips()
    def enable()
    def disable()
    def set_tip(widget, tip_text, tip_private=None)
    def force_window()
```

```
Functions

    def gtk.tooltips_data_get(widget)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Tooltips
```

Attributes

"tip_window"	Read	The window that the tooltip is displayed in.
"tip_label"	Read	The label that displays the tooltip text.
"active_tips_data"	Read	The data associated with the active tooltip.
"tips_data_list"	Read	A list containing the data associated with the tooltips in a tooltips group. For each tooltip the data is a tuple containing: the tooltip object, the associated widget, the tooltip text and the tooltip private text.
"delay"	Read	The delay between the mouse pausing over the widget and the display of the tooltip in msec.
"enabled"	Read	If `TRUE` the tooltips are enabled
"use_sticky_delay"	Read	If `TRUE` shorten the delay for showing a tooltip on another widget is already showing a tooltip.
"timer_tag"	Read	The tag of the timeout handler used for the delay.

Description

Tooltips are the messages that appear next to a widget when the mouse pointer is held over it for a short amount of time. They are especially helpful for adding more verbose descriptions of things such as buttons in a toolbar. An individual tooltip belongs to a group of tooltips. A group is created with a call to the gtk.Tooltips() constructor. Every tooltip in the group can then be turned off with a call to the `disable()` method and enabled with the `enable()` method. To assign a tip to a particular `gtk.Widget`, use the `set_tip()` method.

Note

Tooltips can only be set on widgets which have their own X window. To check if a widget has its own window use `widget.flags()&gtk.NO_WINDOW`. To add a tooltip to a widget that doesn't have its own window, place the widget inside a `gtk.EventBox` and add a tooltip to the eventbox instead.

The default appearance of all tooltips in a program is determined by the current theme that the user has selected. Information about the tooltip (if any) associated with an arbitrary widget can be retrieved using the `gtk.tooltips_data_get()` function.

Constructor

```
  gtk.Tooltips()
```

Returns :	a new `gtk.Tooltips` object

Creates an empty `gtk.Tooltips` group.

Methods

gtk.Tooltips.enable

```
def enable()
```

The `enable()` method enables a group of tooltips. A tooltip will be displayed over its associated widget when the mouse pointer pauses over the widget.

gtk.Tooltips.disable

```
def disable()
```

The `disable()` method disables a group of tooltips. A tooltip will not be displayed over its associated widget when the mouse pointer pauses over the widget.

gtk.Tooltips.set_tip

```
def set_tip(widget, tip_text, tip_private=None)
```

`widget` :	a `gtk.Widget`
`tip_text` :	the tooltip text
`tip_private` :	the tooltip private text for context sensitive display

The `set_tips()` method creates a tooltip for the specified *`widget`*. The text specified by *`tooltip_text`* will be displayed when the mouse pointer pauses over *`widget`* if the tooltips are enabled..

gtk.Tooltips.force_window

```
def force_window()
```

The `force_window()` method ensures that the window used for displaying the given tooltips is created. Applications should never have to call this function, since PyGTK takes care of this.

Functions

gtk.tooltips_data_get

```
def gtk.tooltips_data_get(widget)
```

`widget` :	a widget
Returns :	a tuple containing the tooltip data associated with *`widget`* or `None`

The `gtk.tooltips_data_get()` function returns a tuple containing the tooltip data associated with *`widget`*. The tuple contains:

- the `gtk.Tooltips` group containing the tooltip
- the widget
- the tooltip text string
- the tooltip private text string or `None`

If *`widget`* does not have an associated tooltip this function returns `None`.

gtk.TreeDragDest

gtk.TreeDragDest an interface that manages the data transfer for a destination of a `gtk.TreeView` drag and drop operation

Synopsis

```
class gtk.TreeDragDest(gobject.GInterface):
    def drag_data_received(dest, selection_data)
    def row_drop_possible(dest_path, selection_data)
```

Description

The `gtk.TreeDragDest` is an interface for checking and receiving the data for the destination of a `gtk.TreeView` drag and drop operation.

Methods

gtk.TreeDragDest.drag_data_received

```
    def drag_data_received(dest, selection_data)
```

dest :	the row to drop the data in front of
selection_data :	the data to drop
Returns :	`TRUE` if a new row was created before position *dest*

The `drag_data_received()` method asks the `gtk.TreeDragDest` to insert a row before the path *dest*, deriving the contents of the row from *selection_data*. If *dest* is outside the tree so that inserting before it is impossible, `FALSE` will be returned. Also, `FALSE` may be returned if the new row is not created for some model−specific reason.

gtk.TreeDragDest.row_drop_possible

```
    def row_drop_possible(dest_path, selection_data)
```

dest_path :	a destination row
selection_data :	the data being dragged
Returns :	TRUE if a drop is possible before *dest_path*

The `row_drop_possible()` method determines if a drop is possible before the tree path specified by *dest_path* and at the same depth as *dest_path*. That is, can we drop the data specified by *selection_data* at that location. *dest_path* does not have to exist but the return value will almost certainly be `FALSE` if the parent of *dest_path* doesn't exist, though.

gtk.TreeDragSource

gtk.TreeDragSource — an interface that manages the source data transfer for a `gtk.TreeView` drag and drop operation

Synopsis

```
class gtk.TreeDragSource(gobject.GInterface):
    def row_draggable(path)
    def drag_data_delete(path)
    def drag_data_get(path, selection_data)
```

Description

A `gtk.TreeDragSource` is an interface that provides for the management of the source data for a `gtk.TreeView` drag and drop operation.

Methods

gtk.TreeDragSource.row_draggable

```
    def row_draggable(path)
```

path :	the row from which the user is initiating a drag
Returns :	`TRUE` if the row can be dragged

The `row_draggable`() method asks the `gtk.TreeDragSource` if the row specified by *`path`* can be used as the source of a DND operation. If the `gtk.TreeDragSource` doesn't implement this interface, the row is assumed draggable.

gtk.TreeDragSource.drag_data_delete

```
    def drag_data_delete(path)
```

path :	the row that was being dragged
Returns :	`TRUE` if the row was successfully deleted

The `drag_data_delete`() method asks the `gtk.TreeDragSource` to delete the row specified by *`path`*, because it was moved somewhere else via drag–and–drop. This method returns `FALSE` if the deletion fails because *`path`* no longer exists, or for some other model–specific reason.

gtk.TreeDragSource.drag_data_get

```
    def drag_data_get(path, selection_data)
```

`path` :	the row that was dragged
`selection_data` :	a `gtk.SelectionData` to fill with data from the dragged row
Returns :	`TRUE` if data of the required type was provided

The `drag_data_get`() method asks the `gtk.TreeDragSource` to fill in the selection data object specified by *`selection_data`* with a representation of the row specified by *`path`*. The *`selection_data`* target attribute gives the required type of the data.

gtk.TreeIter

gtk.TreeIter An object that points at a path in a `gtk.TreeModel`.

Synopsis

```
class gtk.TreeIter(gobject.GBoxed):
    def copy()
    def free()
```

Description

A `gtk.TreeIter` is an object that points at a path in a `gtk.TreeModel`. A `gtk.TreeIter` is created using one of the `gtk.TreeModel` or `gtk.TreeModelSort` methods:

- `gtk.TreeModel.get_iter()`
- `gtk.TreeModel.get_iter_from_string()`
- `gtk.TreeModel.get_iter_first()`
- `gtk.TreeModel.get_iter_root()`
- `gtk.TreeModel.iter_children()`
- `gtk.TreeModel.iter_parent()`
- `gtk.TreeModelSort.convert_child_iter_to_iter()`
- `gtk.TreeModelSort.convert_child_iter_to_child_iter()`

Methods

gtk.TreeIter.copy

```
    def copy()
```

Returns :	a copy of the treeiter.

The `copy`() method returns a copy of the treeiter. This iter must be freed with the `free()` method.

gtk.TreeIter.free

```
   def free()
```

The `free()` method frees a `gtk.TreeIter`.

gtk.TreeModel

gtk.TreeModel — the tree interface used by `gtk.TreeView`

Synopsis

```
class gtk.TreeModel(gobject.GInterface):
    def get_flags()
    def get_n_columns()
    def get_column_type(index)
    def get_iter(path)
    def get_iter_from_string(path_string)
    def get_string_from_iter(iter)
    def get_iter_root()
    def get_iter_first()
    def get_path(iter)
    def get_value(iter, column)
    def iter_next(iter)
    def iter_children(parent)
    def iter_has_child(iter)
    def iter_n_children(iter)
    def iter_nth_child(parent, n)
    def iter_parent(child)
    def ref_node(iter)
    def unref_node(iter)
    def get(iter, column, ...)
    def foreach(func, user_data)
    def row_changed(path, iter)
    def row_inserted(path, iter)
    def row_has_child_toggled(path, iter)
    def row_deleted(path)
    def rows_reordered(path, iter, new_order)
    def filter_new(root=None)
Functions

    def gtk.tree_row_reference_inserted(proxy, path)
    def gtk.tree_row_reference_deleted(proxy, path)
```

Signal Prototypes

"row–changed"	`def callback(treemodel, path, iter, user_param1, ...)`
"row–deleted"	`def callback(treemodel, path, user_param1, ...)`
"row–has–child–toggled"	`def callback(treemodel, path, iter, user_param1, ...)`

"row−inserted"	`def callback(treemodel, path, iter, user_param1, ...)`
"rows−reordered"	`def callback(treemodel, path, iter, new_order, user_param1, ...)`

Description

The `gtk.TreeModel` interface defines a generic tree interface for use by the `gtk.TreeView` widget. It is an abstract interface, and is designed to be usable with any appropriate data structure. The programmer just has to implement this interface on their own data type for it to be viewable by a `gtk.TreeView` widget.

The model is represented as a hierarchical tree of strongly−typed, columned data. In other words, the model can be seen as a tree where every node has different values depending on which column is being queried. The type of data found in a column is determined by using the Python or GObject type system (i.e. gobject.TYPE_INT, gtk.BUTTON, gobject.TYPE_STRING, etc.). The types are homogeneous per column across all nodes. It is important to note that this interface only provides a way of examining a model and observing changes. The implementation of each individual model decides how and if changes are made.

In order to make life simpler for programmers who do not need to write their own specialized model, two generic models are provided: the `gtk.TreeStore` and the `gtk.ListStore`. To use these, the developer simply pushes data into these models as necessary. These models provide the data structure as well as all appropriate tree interfaces. As a result, implementing drag and drop, sorting, and storing data is trivial. For the vast majority of trees and lists, these two models are sufficient.

Models are accessed on a node−column level of granularity. One can query for the value of a model at a certain node and a certain column on that node. A particular node in a model is referenced using a path or a `gtk.TreeIter` object. Most of the interface consists of operations on a `gtk.TreeIter`.

A path is essentially a potential node. It is a location on a model that may or may not actually correspond to a node on a specific model. A path can be converted into either an array of unsigned integers or a string. The string form is a list of numbers separated by a colon. Each number refers to the offset at that level. Thus, the path "0" refers to the root node and the path "2:4" refers to the fifth child of the third node.

By contrast, a `gtk.TreeIter` is a reference to a specific node on a specific model. One can convert a path to a treeiter by calling `get_iter()`. These treeiters are the primary way of accessing a model and are similar to the textiters used by `gtk.TextBuffer`. The model interface defines a set of operations using them for navigating the model.

It is expected that models fill in the treeiter with private data. For example, the `gtk.ListStore` model, which is internally a simple linked list, stores a list node in one of the pointers. The `gtk.TreeModelSort` stores an array and an offset in two of the pointers. Additionally, there is an integer field. This field is generally filled with a unique stamp per model. This stamp is for catching errors resulting from using invalid treeiters with a model.

The lifecycle of a treeiter can be a little confusing at first. treeiters are expected to always be valid for as long as the model is unchanged (and doesn't emit a signal). The model is considered to own all outstanding treeiters and nothing needs to be done to free them from the user's point of view. Additionally, some models guarantee that an treeiter is valid for as long as the node it refers to is valid (most notably the `gtk.TreeStore` and `gtk.ListStore`). Although generally uninteresting, as one always has to allow for the case where treeiters do not persist beyond a signal, some very important performance enhancements were made in the sort model. As a result, the `gtk.TREE_MODEL_ITERS_PERSIST` flag was added to indicate this behavior.

A `gtk.TreeModel` object supports some of the Python Mapping protocol that allows you to retrieve a `gtk.TreeModelRow` object representing a row in the model. You can also set the values in a row using the same protocol. For example, you can retrieve the second row of a gtk.TreeModel using any of:

```
treemodelrow = model[1]
treemodelrow = model[(1,)]
treemodelrow = model['1']
treemodelrow = model["1"]
```

Also if the model has two columns both containing strings then the following will set the values of the third row.

```
model[(2,)] = ('new string value', 'string 2')
```

You can also retrieve the number of top level items in the gtk.TreeModel by using the Python `len()` function:

```
n_rows = len(model)
```

A gtk.TreeModelRowIter object can be retrieved for iterating over the top level rows of a gtk.TreeModel by calling the Python `iter()` function:

```
treemodelrowiter = iter(model)
```

Methods

gtk.TreeModel.get_flags

```
def get_flags()
```

Returns :	the flags supported by this interface.

The `get_flags()` method returns a set of flags supported by this interface. The flags are a bitwise combination of:

`gtk.TREE_MODEL_ITERS_PERSIST`	Treeiters survive all signals emitted by the tree.
`gtk.TREE_MODEL_LIST_ONLY`	The model is a list only, and never has children

The flags supported should not change during the lifecycle of the tree_model.

gtk.TreeModel.get_n_columns

```
def get_n_columns()
```

Returns :	The number of columns.

The `get_n_columns()` method returns the number of columns supported by the treemodel.

gtk.TreeModel.get_column_type

```
def get_column_type(index)
```

`index` :	the column index.
Returns :	the type of the column.

The `get_column_type()` method returns the type of the column.

gtk.TreeModel.get_iter

```
def get_iter(path)
```

path :	a path
Returns :	a new gtk.TreeIter that points at *path*.

The get_iter() method returns a new gtk.TreeIter pointing to *path*. This method raises a ValueError exception if *path* is not a valid tree path.

gtk.TreeModel.get_iter_from_string

```
def get_iter_from_string(path_string)
```

path_string :	a string representation of a path.
Returns :	a new gtk.TreeIter that points at the path represented by *path_string*

The get_iter_from_string() method returns a gtk.TreeIter pointing to the path represented by *path_string*, if it exists. This method raises a ValueError exception if *path_string* does not represent a valid tree path.

gtk.TreeModel.get_string_from_iter

```
def get_string_from_iter(iter)
```

iter :	An gtk.TreeIter.
Returns :	A string representation of iter

Note

This method is available in PyGTK 2.2 and above.

The get_string_from_iter() method returns a string representation of the path pointed to by *iter*. This string is a ':' separated list of numbers. For example, "4:10:0:3" would be an acceptable return value for this string.

gtk.TreeModel.get_iter_root

```
def get_iter_root()
```

Returns :	a new gtk.TreeIter that points at the first path in the treemodel or None

The get_iter_root() method returns a gtk.TreeIter pointing to the path "0" or None if the tree is empty.

gtk.TreeModel.get_iter_first

```
def get_iter_first()
```

Returns :	a new gtk.TreeIter that points at the first path in the treemodel or None

The get_iter_first() method returns a gtk.TreeIter pointing to the path "0" or None if the tree is empty.

gtk.TreeModel.get_path

```
    def get_path(iter)
```

iter :	a gtk.TreeIter.
Returns :	the tree path referenced by *iter*.

The get_path() method returns the tree path referenced by *iter*.

gtk.TreeModel.get_value

```
    def get_value(iter, column)
```

iter :	a gtk.TreeIter.
column :	the column value to retrieve.
Returns :	a value.

The get_value() method returns the value at *column* at the path pointed to by *iter*.

gtk.TreeModel.iter_next

```
    def iter_next(iter)
```

iter :	a gtk.TreeIter.
Returns :	a gtk.TreeIter pointing at the next row or None if there is no next row.

The iter_next() method returns a gtk.TreeIter pointing at the row at the current level after the row referenced by *iter*. If there is no next row, None is returned. *iter* is unchanged.

gtk.TreeModel.iter_children

```
    def iter_children(parent)
```

parent :	the gtk.TreeIter pointing to the parent
Returns :	the new gtk.TreeIter to be set to the first child or None

The iter_children() method returns a new gtk.TreeIter pointing to the first child of *parent*. If *parent* has no children, None is returned. *parent* will remain a valid node after this method has been called.

gtk.TreeModel.iter_has_child

```
    def iter_has_child(iter)
```

iter :	a gtk.TreeIter to test for children.
Returns :	TRUE if *iter* has children.

The iter_has_child() method returns TRUE if *iter* has children, or FALSE otherwise.

gtk.TreeModel.iter_n_children

```
    def iter_n_children(iter)
```

iter :	a gtk.TreeIter, or None.
Returns :	the number of children of *iter*.

The `iter_n_children()` method returns the number of children that *`iter`* has. As a special case, if *`iter`* is `None`, then the number of top level nodes is returned.

gtk.TreeModel.iter_nth_child

```
    def iter_nth_child(parent, n)
```

`parent` :	a `gtk.TreeIter` to get the child from, or `None`.
`n` :	Then index of the desired child.
Returns :	the `gtk.TreeIter` that is set to the nth child or `None`

The `iter_nth_child()` method returns a new `gtk.TreeIter` pointing to the child of *`parent`*, with the index specified by *`n`*. The first index is 0. If *`n`* is too big, or *`parent`* has no children, this method returns `None`. *`parent`* will remain a valid node after this function has been called. As a special case, if *`parent`* is `None`, then the treeiter points to the *`n`*th root node.

gtk.TreeModel.iter_parent

```
    def iter_parent(child)
```

`child` :	The `gtk.TreeIter`.
Returns :	a new `gtk.TreeIter` set to the parent of *`child`* or `None`

The `iter_parent()` method returns a `gtk.TreeIter` pointing to the parent of *`child`*. If *`child`* is at the top level, and doesn't have a parent, then `None` is returned. *`child`* will remain a valid node after this method has been called.

gtk.TreeModel.ref_node

```
    def ref_node(iter)
```

`iter` :	a `gtk.TreeIter`.

The `ref_node()` method lets the treemodel ref the node that *`iter`* points to. This is an optional method for models to implement. To be more specific, models may ignore this call as it exists primarily for performance reasons. This function is primarily meant as a way for views to let the caching model know when nodes are being displayed (and hence, whether or not to cache that node.) For example, a file−system based model would not want to keep the entire file−hierarchy in memory, just the sections that are currently being displayed by every current view. A model should be expected to be able to get a treeiter independent of it's reffed state.

gtk.TreeModel.unref_node

```
    def unref_node(iter)
```

`iter` :	a `gtk.TreeIter`.

The `unref_node()` method lets the treemodel unref the node that *`iter`* points to. This is an optional method for models to implement. To be more specific, models may ignore this call as it exists primarily for performance reasons. For more information on what this means, see the `ref_node()` method. Please note that nodes that are deleted are not unreffed.

gtk.TreeModel.get

```
def get(iter, column, ...)
```

iter :	a gtk.TreeIter pointing at the row to retrieve data value from
column :	a column number
... :	zero or more column numbers
Returns :	a tuple containing the column values

Note

This method is available in PyGTK 2.4 and above.

The `get`() method returns a tuple containing the values of one or more cells in the row referenced by the gtk.TreeIter specified by *iter*. column specifies the first column number to retrieve a value from. The additional arguments should contain integer column numbers for additional column values. For example, to get values from columns 0 and 3, you would write:

```
value0, value3 = treemodel_get(iter, 0, 3)
```

gtk.TreeModel.foreach

```
def foreach(func, user_data)
```

func :	a function to be called on each row
user_data :	the user data to passed to *func*.

The `foreach`() method calls *func* on each node in model in a depth–first fashion. *user_data* is passed to *func* each time it is called. If *func* returns `TRUE`, then the operation ceases, and `foreach`() returns.

The signature of *func* is:

```
def func(model, path, iter, user_data)
```

where *model* is the treemodel, *path* is the current path, and *iter* is a treeiter pointing to *path*.

If *func* is an object method its signature will be:

```
def func(self, model, path, iter, user_data)
```

gtk.TreeModel.row_changed

```
def row_changed(path, iter)
```

path :	a path pointing to the changed row
iter :	a gtk.TreeIter pointing to the changed row

The `row_changed`() method emits the "row–changed" signal on the treemodel with the parameters *path* and *iter* that are the path and a treeiter pointing to the path of the changed row.

gtk.TreeModel.row_inserted

```
def row_inserted(path, iter)
```

path :	a path pointing to the inserted row

iter :	a gtk.TreeIter pointing to the inserted row

The `row_inserted`() method emits the "row−inserted" signal on the treemodel with the parameters *path* and *iter* that are the path and a treeiter pointing to the path of the inserted row.

gtk.TreeModel.row_has_child_toggled

```
def row_has_child_toggled(path, iter)
```

path :	a path pointing to the changed row
iter :	a gtk.TreeIter pointing to the changed row

The `row_has_child_toggled`() method emits the "row−has−child−toggled" signal on the treemodel with the parameters *path* and *iter* that are the path and a treeiter pointing to the path of the changed row. This should be called by models after the child state of a node changes.

gtk.TreeModel.row_deleted

```
def row_deleted(path)
```

path :	a path pointing to the previous location of the deleted row.

The `row_deleted`() method emits the "row−deleted" signal on the treemodel. This should be called by models after a row has been removed. The location pointed to by *path* should be the location that the deleted row was at. It may not be a valid location anymore.

gtk.TreeModel.rows_reordered

```
def rows_reordered(path, iter, new_order)
```

path :	A tree path pointing to the tree node whose children have been reordered, or `None` or () or "" to indicate the top level node.
iter :	A valid gtk.TreeIter pointing to the node whose children have been reordered, or `None` to indicate the top level node.
new_order :	a sequence of integers containing the new indexes of the children, i.e. the former child `n` is now at the position specified by *new_order*`[n]`.

The `rows_reordered`() method emits the "rows_reordered" signal on the tree model. This method should be called by a tree model when its rows have been reordered. If *iter* is `None` to indicate that the top level rows have been reordered, *path* should be `None` or () or "".

gtk.TreeModel.filter_new

```
def filter_new(root=None)
```

root :	a tree path or `None`.
Returns :	A new gtk.TreeModel.

Note

This method is available in PyGTK 2.4 and above.

The `filter_new`() method creates a new gtk.TreeModel, with the tree model as the child_model and the virtual root specified by *root*.

Functions

gtk.tree_row_reference_inserted

```
    def gtk.tree_row_reference_inserted(proxy, path)
```

proxy :	a GObject
path :	a row position that was inserted

The `gtk.tree_row_reference_inserted()` function lets a set of row references know that the model emitted the "row_inserted" signal for the row specified by *path*.

gtk.tree_row_reference_deleted

```
    def gtk.tree_row_reference_deleted(proxy, path)
```

proxy :	a GObject
path :	a row position that was deleted

The `gtk.tree_row_reference_deleted()` function lets a set of row references know that the model emitted the "row_deleted" signal for the row specified by *path*.

Signals

The "row−changed" gtk.TreeModel Signal

```
    def callback(treemodel, path, iter, user_param1, ...)
```

treemodel :	the treemodel that received the signal
path :	a path
iter :	a gtk.TreeIter pointing at *path*
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "row−changed" signal is emitted when the row specified by *path* and pointed to by *iter* has changed in the *treemodel*. Usually, this means that one or more column values have changed.

The "row−deleted" gtk.TreeModel Signal

```
    def callback(treemodel, path, user_param1, ...)
```

treemodel :	the treemodel that received the signal
path :	a path
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "row−deleted" signal is emitted when the row that was specified by *path* is deleted from *treemodel*.

The "row−has−child−toggled" gtk.TreeModel Signal

```
    def callback(treemodel, path, iter, user_param1, ...)
```

treemodel :	the treemodel that received the signal
path :	a path
iter :	a `gtk.TreeIter` pointing at *path*
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "row−has−child−toggled" signal is emitted when the child state of the row specified by *path* and pointed to by *iter* has changed in *treemodel*.

The "row−inserted" gtk.TreeModel Signal

```
    def callback(treemodel, path, iter, user_param1, ...)
```

treemodel :	the treemodel that received the signal
path :	a path
iter :	a `gtk.TreeIter` pointing at *path*
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "row−inserted" signal is emitted when the row specified by *path* and pointed to by *iter* is inserted into *treemodel*. The row referenced by *iter* will be empty so using the `get_value()` method will always return `None`. Connect to the "row−changed" signal if you want to track value changes.

The "rows−reordered" gtk.TreeModel Signal

```
    def callback(treemodel, path, iter, new_order, user_param1, ...)
```

treemodel :	the treemodel that received the signal
path :	a path
iter :	a `gtk.TreeIter` pointing at *path*
new_order :	an array of reordered row numbers
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "rows−reordered" signal is emitted when the *treemodel* child rows of the row specified by *path* and pointed to by *iter* are reordered. *new_order* is an array of node index numbers representing the new order of the rows. The value of *new_order* cannot be retrieved in PyGTK because it is passed as an opaque pointer (`gobject.GPointer`) value. *iter* may be `None` and *path* an empty tuple to indicate that the top level rows were reordered.

Prev	Up	Next
gtk.TreeIter	Home	gtk.TreeModelFilter

gtk.TreeModelFilter

Prev **The gtk Class Reference** Next

gtk.TreeModelFilter

gtk.TreeModelFilter a `gtk.TreeModel` which hides parts of an underlying tree (new in PyGTK 2.4)

Synopsis

```
class gtk.TreeModelFilter(gobject.GObject, gtk.TreeModel, gtk.TreeDragSource):
    def set_visible_func(func, data=None)
    def set_modify_func(types, func, data=None)
    def set_visible_column(column)
    def get_model()
    def convert_child_iter_to_iter(child_iter)
    def convert_iter_to_child_iter(filter_iter)
    def convert_child_path_to_path(child_path)
    def convert_path_to_child_path(filter_path)
    def refilter()
    def clear_cache()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.TreeModelFilter (implements gtk.TreeModel, gtk.TreeDragSource)
```

Properties

"child–model"	Read–Write–Construct Only	The `gtk.TreeModel` for the filtermodel to filter. Available in GTK+ 2.4 and above.
"virtual–root"	Read–Write–Construct Only	The virtual root (relative to the child model) for this filtermodel. Available in GTK+ 2.4 and above.

Description

Note

This object is available in PyGTK 2.4 and above.

A `gtk.TreeModelFilter` is a tree model which wraps another tree model, and can do the following things:

- Filter specific rows, based on data from a "visible column", a column storing booleans indicating whether the row should be filtered or not, or based on the return value of a "visible function", which gets a *`model`*, *`iter`* and *`user_data`* and returns a boolean indicating whether the row should be filtered or not.
- Modify the "appearance" of the model, using a modify function. This is extremely powerful and allows for just changing some values and also for creating a completely different synthetic model based on the child model. For example, you can create a model with columns synthesized from the data in the child model.
- Set a different root node, also known as a "virtual root". You can pass in a tree path indicating the root node for the filter at construction time.

A `gtk.TreeModelFilter` is created using the `gtk.TreeModel.filter_new()` method. For example:

```
  liststore = gtk.ListStore(gobject.TYPE_INT, gobject.TYPE_STRING)
  modelfilter = liststore.filter_new()
```

Methods

gtk.TreeModelFilter.set_visible_func

```
def set_visible_func(func, data=None)
```

func :	a function called to determine the visibility of a row
data :	User data to pass to `func`

Note

This method is available in PyGTK 2.4 and above.

The `set_visible_func()` method sets the visible function used when filtering the rows of the treemodel filter to the value of *`func`*. *`data`* is the user data that is passed to func (see below). This method will fail if the `set_visible_column()` method has already been called. The visible function signature is:

```
def visible_func(model, iter, user_data):
```

where *`model`* is the child `gtk.TreeModel`, *`iter`* is a `gtk.TreeIter` pointing at a row in model and *`user_data`* is the data parameter. The function should return `TRUE` if the row should be visible.

gtk.TreeModelFilter.set_modify_func

```
def set_modify_func(types, func, data=None)
```

types :	a sequence containing the column types
func :	a function that is called to provide the data for a specific row and column
data :	user data to pass to the modify function, or `None`.

Note

This method is available in PyGTK 2.4 and above.

The `set_modify_func()` method uses the list of column types specified by *`types`* and the function specified by *`func`* to provide a synthetic model based on the child model of the `gtk.TreeModelFilter`. *`data`* is passed to *`func`* when it is called. *`func`* is called for each data access to return the data which should be displayed at the location specified using the parameters of the modify function.

The signature of func is:

```
def func(model, iter, column, user_data)
```

where *`model`* is the `gtk.TreeModelFilter`, *`iter`* is a `gtk.TreeIter` pointing at a row in *`model`*, *`column`* is the column number to provide the value for and *`user_data`* is *`data`*. *`func`* should returns the generated value for the specified location in *`model`*.

Note

This method must be called before the `gtk.TreeModelFilter` is associated with a `gtk.TreeView` and before either of the `gtk.TreeModel.get_n_columns()` or `gtk.TreeModel.get_column_type()` methods are called. Also this method can only be called once – there is no way to change the modify function once it is set.

Since *`func`* is called for every access to a value in *`model`*, it will be slow for models with a large number of rows and/or columns.

gtk.TreeModelFilter.set_visible_column

```
    def set_visible_column(column)
```

column :	the number of the column containing the visible information.

Note

This method is available in PyGTK 2.4 and above.

The `set_visible_column()` method sets the visible column setting to the value of *column*. The visible column setting contains the number of the "child−model" column that is used to determine the visibility of the model rows. The specified column should be a column of type `gobject.TYPE_BOOLEAN`, where `True` means that a row is visible, and `False`, not visible. This method will fail if the `set_visible_func()` method has already been called.

gtk.TreeModelFilter.get_model

```
    def get_model()
```

Returns :	the child `gtk.TreeModel`

Note

This method is available in PyGTK 2.4 and above.

The `get_model()` method returns the child `gtk.TreeModel` of the treemodel filter

gtk.TreeModelFilter.convert_child_iter_to_iter

```
    def convert_child_iter_to_iter(child_iter)
```

child_iter :	A valid `gtk.TreeIter` pointing to a row on the child model.
Returns :	a `gtk.TreeIter` pointing to a row in the treemodel filter.

Note

This method is available in PyGTK 2.4 and above.

The `convert_child_iter_to_iter()` method returns a `gtk.TreeIter` pointing to the row in the treemodel filter that corresponds to the child treemodel row pointed to by the `gtk.TreeIter` specified by *child_iter*.

gtk.TreeModelFilter.convert_iter_to_child_iter

```
    def convert_iter_to_child_iter(filter_iter)
```

filter_iter :	A valid `gtk.TreeIter` pointing to a row in the treemodel filter.
Returns :	a `gtk.TreeIter` pointing to a row in the child treemodel.

Note

This method is available in PyGTK 2.4 and above.

The `convert_iter_to_child_iter()` method a `gtk.TreeIter` pointing to the row in the child treemodel that corresponds to the treemodel filter row pointed to by the `gtk.TreeIter` specified by *filter_iter*.

gtk.TreeModelFilter.convert_child_path_to_path

```
    def convert_child_path_to_path(child_path)
```

child_path :	a tree path in the child treemodel to convert.
Returns :	a treemodel filter tree path, or `None`.

Note

This method is available in PyGTK 2.4 and above.

The `convert_child_path_to_path`() method returns a treemodel filter tree path that corresponds to the child treemodel tree path specified by *child_path*. If *child_path* isn't a valid path on the child model, `None` is returned.

gtk.TreeModelFilter.convert_path_to_child_path

```
    def convert_path_to_child_path(filter_path)
```

filter_path :	a treemodel filter tree path to convert.
Returns :	a child treemodel tree path, or `None`.

Note

This method is available in PyGTK 2.4 and above.

The `convert_path_to_child_path`() method returns a child treemodel tree path that corresponds to the treemodel filter tree path specified by *filter_path*. If *filter_path* does not point to a row in the child model, `None` is returned.

gtk.TreeModelFilter.refilter

```
    def refilter()
```

Note

This method is available in PyGTK 2.4 and above.

The `refilter`() method emits the `gtk.TreeModel` "row−changed" signal for each row in the child model, thereby causing the filter to re−evaluate whether a row is visible or not.

gtk.TreeModelFilter.clear_cache

```
    def clear_cache()
```

Note

This method is available in PyGTK 2.4 and above.

The `clear_cache`() method clears the treemodel filter of any cached iterators that haven't been reffed with the `gtk.TreeModel.ref_node()`. This might be useful if the child model being filtered is static (and doesn't change often) and there has been a lot of unreffed access to nodes. As a side effect of this function, all unreffed iters will be invalid. This method should almost never be called by an application.

gtk.TreeModelSort

gtk.TreeModelSort — a tree model that is a sorted version of a child gtk.TreeModel

Synopsis

```
class gtk.TreeModelSort(gobject.GObject, gtk.TreeModel, gtk.TreeSortable):
    gtk.TreeModelSort(child_model)
    def get_model()
    def convert_child_path_to_path(child_path)
    def convert_child_iter_to_iter(sort_iter, child_iter)
    def convert_path_to_child_path(sorted_path)
    def convert_iter_to_child_iter(child_iter, sorted_iter)
    def reset_default_sort_func()
    def clear_cache()
    def iter_is_valid(iter)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.TreeModelSort (implements gtk.TreeModel, gtk.TreeSortable)
```

Properties

"model"	Read–Write–Construct	The child model for the gtk.TreeModelSort to sort.

Description

A gtk.TreeModelSort is a sorted model of its child model. A gtk.TreeModelSort implements the gtk.TreeModel interface and the gtk.TreeSortable interface to manage the sort functions.

Constructor

```
    gtk.TreeModelSort(child_model)
```

child_model :	a child gtk.TreeModel
Returns :	A new gtk.TreeModel.

Creates a new gtk.TreeModel, with *child_model* as the child model.

Methods

gtk.TreeModelSort.get_model

```
def get_model()
```

Returns :	the "child model" being sorted

The `get_model`() method returns the model that the `gtk.TreeModelSort` is sorting.

gtk.TreeModelSort.convert_child_path_to_path

```
def convert_child_path_to_path(child_path)
```

`child_path` :	A child tree path to convert
Returns :	A new tree path in the treemodelsort, or `None`

The `convert_child_path_to_path`() method converts the path in the child model specified by *`child_path`* to a path relative to the treemodelsort. That is, *`child_path`* points to a path in the child model. The returned path will point to the same row in the sorted model. If *`child_path`* isn't a valid path on the child model, then `None` is returned.

gtk.TreeModelSort.convert_child_iter_to_iter

```
def convert_child_iter_to_iter(sort_iter, child_iter)
```

`sort_iter` :	`None` or a `gtk.TreeIter` for backward compatibility.
`child_iter` :	A valid `gtk.TreeIter` pointing to a row on the child model
Returns :	A `gtk.TreeIter` pointing to the same path in the sorted model.

The `convert_child_iter_to_iter`() method returns a `gtk.TreeIter` that points to the row in the treemodelsort that corresponds to the row pointed to by *`child_iter`*. *`sort_iter`* parameter should be `None` but can specify a `gtk.TreeIter` for backward compatibility.

gtk.TreeModelSort.convert_path_to_child_path

```
def convert_path_to_child_path(sorted_path)
```

`sorted_path` :	a path in the sorted model
Returns :	a new path in the child model, or `None`

The `convert_path_to_child_path`() method returns a path in the child model that refers to the same row as the path in the sorted model specified by *`sorted_path`*. That is, *`sorted_path`* points to a location in treemodelsort and the returned path will point to the same location in the child model. If *`sorted_path`* does not point to a location in the child model, `None` is returned.

gtk.TreeModelSort.convert_iter_to_child_iter

```
def convert_iter_to_child_iter(child_iter, sorted_iter)
```

`child_iter` :	`None` or a `gtk.TreeIter` for backward compatibility.
`sorted_iter` :	A valid `gtk.TreeIter` pointing to a row on *`tree_model_sort`*.
Returns :	A `gtk.TreeIter` that points to a row in the child model

The `convert_iter_to_child_iter`() method returns a `gtk.TreeIter` that points to the row in the child model that is the same row pointed to by *sorted_iter* in the treemodelsort. *child_iter* should be None but can specify a gtk.TreeIter for backward compatibility.

gtk.TreeModelSort.reset_default_sort_func

```
    def reset_default_sort_func()
```

The reset_default_sort_func() method resets the default sort function to be in the 'unsorted' state. That is, it is in the same order as the child model. It will re–sort the model to be in the same order as the child model only if the gtk.TreeModelSort is in 'unsorted' state.

gtk.TreeModelSort.clear_cache

```
    def clear_cache()
```

The clear_cache() method clears the treemodelsort of any cached iterators that haven't been reffed with the ref_node() method. This might be useful if the child model being sorted is static (and doesn't change often) and there has been a lot of unreffed access to nodes. As a side effect of this function, all unreffed gtk.TreeIter objects will be invalid.

gtk.TreeModelSort.iter_is_valid

```
    def iter_is_valid(iter)
```

iter :	A gtk.TreeIter.
Returns :	TRUE if *iter* is valid

Note

This method is available in PyGTK 2.2 and above.

The iter_is_valid() method checks if the gtk.TreeIter specified by *iter* is valid for the treemodel sort.

Warning

This method is slow. Only use it for debugging and/or testing purposes.

Prev	Up	Next
gtk.TreeModelFilter	Home	gtk.TreeModelRow

gtk.TreeRowReference

Prev **The gtk Class Reference** Next

gtk.TreeRowReference

gtk.TreeRowReference — an object maintaining a persistent reference to a gtk.TreeModel row (new in PyGTK 2.4)

Synopsis

```
class gtk.TreeRowReference(gobject.GBoxed):
    gtk.TreeRowReference(model, path)
    def get_path()
    def valid()
    def copy()
    def free()
```

Description

Note

This object is available in PyGTK 2.4 and above.

A `gtk.TreeRowReference` is an object that points to a row in a `gtk.TreeModel` similar to a `gtk.TreeIter`. A `gtk.TreeRowReference`, unlike a `gtk.TreeIter`, maintains a persistent reference in spite of changes in the model.

Constructor

```
    gtk.TreeRowReference(model, path)
```

`model` :	a `gtk.TreeModel`
`path` :	a valid tree path to monitor
Returns :	a `gtk.TreeRowReference`, or `None`

Note

This constructor is available in PyGTK 2.4 and above.

Creates a row reference pointing to the treemodel row specified by *`model`* and *`path`*. This reference will continue pointing to the node in spite of changes in *`model`*. It listens to all signals emitted by *`model`*, and updates its path appropriately. If *`path`* isn't a valid path in *`model`*, `None` is returned.

Methods

gtk.TreeRowReference.get_path

```
    def get_path()
```

Returns :	A current path, or `NULL`.

Note

This method is available in PyGTK 2.4 and above.

The `get_path`() method returns the path that the row reference currently points to, or `None` if the path pointed to is no longer valid.

gtk.TreeRowReference.valid

```
  def valid()
```

Returns :	`TRUE` if the row reference points to a valid path.

Note

This method is available in PyGTK 2.4 and above.

The `valid`() method returns `TRUE` if the row reference is not `None` and refers to a current valid path.

gtk.TreeRowReference.copy

```
  def copy()
```

Returns :	a copy of the row reference

Note

This method is available in PyGTK 2.4 and above.

The `copy`() method returns a copy of the tree row reference.

gtk.TreeRowReference.free

```
  def free()
```

Note

This method is available in PyGTK 2.4 and above.

The free() method frees the tree row reference. The row reference may be `None`.

gtk.TreeSelection

gtk.TreeSelection — the selection object for `gtk.TreeView`

Synopsis

```
class gtk.TreeSelection(gtk.Object):
    def set_mode(type)
    def get_mode()
    def set_select_function(func, data=None)
    def get_tree_view()
    def get_selected()
    def get_selected_rows()
    def count_selected_rows()
    def selected_foreach(func, data=None)
    def select_path(path)
    def unselect_path(path)
    def select_iter(iter)
    def unselect_iter(iter)
    def path_is_selected(path)
    def iter_is_selected(iter)
    def select_all()
    def unselect_all()
    def select_range(start_path, end_path)
    def unselect_range(start_path, end_path)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.TreeSelection
```

Signal Prototypes

"changed"	`def callback(treeselection, user_param1, ...)`

Description

The `gtk.TreeSelection` object is a helper object to manage the selection for a `gtk.TreeView` widget. The `gtk.TreeSelection` object is automatically created when a new `gtk.TreeView` widget is created, and cannot exist independent of this widget. The primary reason the `gtk.TreeSelection` object exists is for cleanliness of code and API. That is, there is no conceptual reason all these functions could not be methods on the `gtk.TreeView` widget instead of a separate function. The `gtk.TreeSelection` object is retrieved from a `gtk.TreeView` by calling the `gtk.TreeView.get_selection()` method. It can be manipulated to check the selection status of the tree, as well as select and deselect individual rows. Selection is done completely on the view side. As a result, multiple views of the same model can have completely different selections. Additionally, you cannot change the selection of a row on the model that is not currently displayed by the view without expanding its parents first.

One of the important things to remember when monitoring the selection of a view is that the "changed" signal is mostly a hint. That is, it may only emit one signal when a range of rows is selected. Additionally, it may on occasion emit a "changed" signal when nothing has happened (mostly as a result of programmers calling the `select_path()` or `select_iter()` methods on an already selected row).

Methods

gtk.TreeSelection.set_mode

```
  def set_mode(type)
```

type :	the selection mode

The `set_mode()` method sets the selection mode of the treeselection to the mode specified by *type*. The value of *type* must be one of: `gtk.SELECTION_NONE`, `gtk.SELECTION_SINGLE`, `gtk.SELECTION_BROWSE` or `gtk.SELECTION_MULTIPLE`. See the GTK Selection Mode Constants description for more detail.

If the previous type was `gtk.SELECTION_MULTIPLE`, then the anchor is kept selected, if it was previously selected.

gtk.TreeSelection.get_mode

```
def get_mode()
```

Returns :	the current selection mode

The `get_mode`() method returns the selection mode for treeselection. See the `set_mode()` method for more information.

gtk.TreeSelection.set_select_function

```
def set_select_function(func, data)
```

`func` :	the selection function.
`data` :	the selection function's data.

The `set_selection_function`() method sets the selection function to *`func`* (a function or method). If the selection function is set, it is called before any node is selected or unselected, giving some control over which nodes are selected. The selection function should return `TRUE` if the state of the node may be toggled, and `FALSE` if the state of the node should be left unchanged. The signature of the selection function callback is:

```
def selectfunction(selection, model, path, path_currently_selected, ...)

def selectmethod(self, selection, model, path, is_selected, ...)
```

where *`selection`* is the `gtk.TreeSelection`, *`model`* is the `gtk.TreeModel` used by the gtkTreeView associated with selection, *`path`* is the path of the selected row, *`is_selected`* is `TRUE` if the row is currently selected and `...` is the user data if any (may not be present if *`data`* was `None`). If *`func`* is a method then *`self`* is the object that the method is called upon.

gtk.TreeSelection.get_tree_view

```
def get_tree_view()
```

Returns :	a `gtk.TreeView`

The `get_tree_iter`() method returns the tree view associated with the treeselection.

gtk.TreeSelection.get_selected

```
def get_selected()
```

Returns :	a 2–tuple containing a reference to the `gtk.TreeModel` and a `gtk.TreeIter` pointing to the currently selected node.

The `get_selected`() method returns a 2–tuple containing the treemodel and a treeiter pointing to the selected node in the treemodel if the treeselection is set to `gtk.SELECTION_SINGLE` or `gtk.SELECTION_BROWSE`. The returned `gtk.TreeIter` will be `None` if there is no row selected. This method will not work if you use *`selection`* is `gtk.SELECTION_MULTIPLE`.

gtk.TreeSelection.get_selected_rows

```
def get_selected_rows()
```

Returns :	a 2–tuple containing the tree model and a list of the tree paths of all selected rows.

Note

This method is available in PyGTK 2.2 and above.

The `get_selected_rows()` method returns a 2–tuple containing a `gtk.TreeModel` and a list of the tree paths of all selected rows. Additionally, if you are planning on modifying the tree model after calling this method, you may want to convert the returned list into a list of `gtk.TreeRowReference` objects. To do this, you can use the gtk.TreeRowReference() constructor.

gtk.TreeSelection.count_selected_rows

```
def count_selected_rows()
```

Returns :	The number of rows selected.

Note

This method is available in PyGTK 2.2 and above.

The `count_selected_rows()` method returns the number of rows that have been selected.

gtk.TreeSelection.selected_foreach

```
def selected_foreach(func, data=None)
```

`func` :	the function or method to call for each selected node.
`data` :	the user data to pass to *`func`*.

The `selected_foreach()` method calls the function or method specified by *`func`* for each selected node passing the user data specified by *`data`*. The signature of *`func`* is:

```
  def foreachfunction(treemodel, path, iter, ...)

  def foreachmethod(self, treemodel, path, iter, ...)
```

where *`treemodel`* is the `gtk.TreeModel` being viewed, *`path`* is the path of the selected row, *`iter`* is a `gtk.TreeIter` pointing to the selected row and `...` is the user data if any (may not be present if *`data`* was `None`). If *`func`* is a method then *`self`* is the object that the method is called upon.

Note

You cannot modify the tree or selection in the callback function.

gtk.TreeSelection.select_path

```
def select_path(path)
```

`path` :	the tree path to be selected.

The `select_path()` method selects the row at *`path`*.

gtk.TreeSelection.unselect_path

```
def unselect_path(path)
```

`path` :	the tree path to be unselected.

The `unselect_path()` method unselects the row at *`path`*.

gtk.TreeSelection.select_iter

```
    def select_iter(iter)
```

`iter` :	the `gtk.TreeIter` to be selected.

The `select_iter()` method selects the row pointed to by the `gtk.TreeIter` specified by *`iter`*.

gtk.TreeSelection.unselect_iter

```
    def unselect_iter(iter)
```

`iter` :	the `gtk.TreeIter` to be unselected.

The `unselect_iter()` method unselects the row pointed to by the `gtk.TreeIter` specified by *`iter`*.

gtk.TreeSelection.path_is_selected

```
    def path_is_selected(path)
```

`path` :	A tree path to check if selected.
Returns :	`TRUE` if *`path`* is selected.

The `path_is_selected()` method returns `TRUE` if the row pointed to by *`path`* is currently selected. If *`path`* does not point to a valid location, `FALSE` is returned.

gtk.TreeSelection.iter_is_selected

```
    def iter_is_selected(iter)
```

`iter` :	a `gtk.TreeIter`
Returns :	`TRUE`, if the row pointed to by *`iter`* is selected

The `iter_is_selected()` method returns `TRUE` if the row pointed to by *`iter`* is currently selected.

gtk.TreeSelection.select_all

```
    def select_all()
```

The `select_all()` method selects all the nodes. The treeselection is must be set to `gtk.SELECTION_MULTIPLE` mode.

gtk.TreeSelection.unselect_all

```
    def unselect_all()
```

The `unselect_all()` method unselects all the nodes.

gtk.TreeSelection.select_range

```
    def select_range(start_path, end_path)
```

`start_path` :	the initial node path of the range.

`end_path` :	the final node path of the range.

The `select_range()` method selects a range of nodes specified by the tree paths *`start_path`* and *`end_path`* inclusive.

gtk.TreeSelection.unselect_range

```
    def unselect_range(start_path, end_path)
```

`start_path` :	The initial node of the range.
`end_path` :	The final node of the range.

Note

This method is available in PyGTK 2.2 and above.

The `unselect_range()` method unselects the range of nodes specified by the tree paths *`start_path`* and *`end_path`* inclusive.

Signals

The "changed" gtk.TreeSelection Signal

```
    def callback(treeselection, user_param1, ...)
```

`treeselection` :	the treeselection that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "changed" signal is emitted when the selection has changed.

Prev	Up	Next
gtk.TreeRowReference	Home	gtk.TreeSortable

	gtk.TreeSortable	
Prev	**The gtk Class Reference**	Next

gtk.TreeSortable

gtk.TreeSortable — an interface for sorting a `gtk.TreeModel`

Synopsis

```
class gtk.TreeSortable(gobject.GInterface):
    def sort_column_changed()
    def get_sort_column_id()
    def set_sort_column_id(sort_column_id, order)
    def set_sort_func(sort_column_id, sort_func, user_data=None)
    def set_default_sort_func(sort_func, user_data=None)
    def has_default_sort_func()
```

Signal Prototypes

"sort–column–changed"	`def callback(treesortable, user_param1, ...)`

Description

The `gtk.TreeSortable` interface provide the common methods for a `gtk.TreeModel` to implement a sorted model for a `gtk.TreeView`. See the `gtk.TreeModelSort` for an example of a treemodel that implements the `gtk.TreeSortable` interface. The key element of the interface is a "sort column ID" which is an arbitrary integer value referring to a sort function and associated user data. A sort column ID must be greater than or equal to zero. A usable sort column ID is created by using the `set_sort_func()` method. The sort column ID can then be used for sorting a `gtk.ListStore` or `gtk.TreeStore` using the `set_sort_column_id()` method.

The `gtk.ListStore` and `gtk.TreeStore` objects automatically setup sort column IDs corresponding to the columns in the store. These sort column IDs are associated with an internal comparison function that handles the fundamental types:

- gboolean
- str
- int
- long
- float

Note

Once a sort column ID has been set on a `gtk.TreeModel` implementing the `gtk.TreeSortable` interface it cannot be returned to the original unsorted state. You can change the sort function or use a default sort function but you cannot set the `gtk.TreeModel` to have no sort function.

Methods

gtk.TreeSortable.sort_column_changed

```
    def sort_column_changed()
```

The `sort_column_changed`() method emits the "sort_column_changed" signal on the treesortable object.

gtk.TreeSortable.get_sort_column_id

```
    def get_sort_column_id()
```

Returns :	a tuple containing the sort column id and the sort type: `gtk.SORT_ASCENDING` or `gtk.SORT_DESCENDING`

The `get_sort_column_id`() method returns a tuple containing the current sort column ID and the sort type (either `gtk.SORT_ASCENDING` or `gtk.SORT_DESCENDING`), if applicable. If the sort column ID is not set, then the tuple `(-2, 0)` is returned. If the sort column ID is set to –1 indicating the default sort function is to be used this method returns (`None`, `None`)

gtk.TreeSortable.set_sort_column_id

```
def set_sort_column_id(sort_column_id, order)
```

`sort_column_id` :	the sort column id to set
`order` :	the sort order: `gtk.SORT_ASCENDING` or `gtk.SORT_DESCENDING`

The `set_sort_column_id()` method sets the current sort comparison function to that associated with the sort column ID specified by *`sort_column_id`* with the sort order type specified by *`order`*. The value of order must be either: `gtk.SORT_ASCENDING` or `gtk.SORT_DESCENDING`. The treesortable will resort itself to reflect this change, after emitting a "sort_column_changed" signal.

If *`sort_column_id`* is –1, then the default sort function will be used, if it is set. If a default sort function is not set then the sort column ID is not changed.

gtk.TreeSortable.set_sort_func

```
def set_sort_func(sort_column_id, sort_func, user_data=None)
```

`sort_column_id` :	the sort column id to set the function for
`sort_func` :	The sorting function
`user_data` :	the user data to pass to the sort func, or `None`

The `set_sort_func()` method sets the comparison function (or method) used when sorting on the sort column ID specified by *`column_id`* to the value specified by *`sort_func`*. If the current sort column id of the treesortable is the same as *`sort_column_id`*, the model will be resorted. The signature of the comparison function (or method) is:

```
def comparefunction(treemodel, iter1, iter2, user_data)

def comparemethod(self, treemodel, iter1, iter2, user_data)
```

where *`treemodel`* is the tree model implementing the `gtk.TreeSortable` interface, *`iter1`* and *`iter2`* point at the rows to compare and *`user_data`* is the user data specified in `set_sort_func()` or *`None`*. If *`sort_func`* is a method then *`self`* is the object associated with the method.

The comparison callback should return –1 if the *`iter1`* row should come before the *`iter2`* row, 0 if the rows are equal, or 1 if the *`iter1`* row should come after the *`iter2`* row.

gtk.TreeSortable.set_default_sort_func

```
def set_default_sort_func(sort_func, user_data=None)
```

`sort_func` :	the sorting function
`user_data` :	the user data to pass to *`sort_func`*, or `None`

The `set_default_sort_func()` method sets the default comparison function (or method) to the value of *`sort_func`*. If the current sort column id of the treesortable is –1 (the `get_sort_column_id()` method returns (None, None)), then the model will be resorted using the *`sort_func`*. See the `set_sort_func()` method for more details on the signature of the comparison function.

gtk.TreeSortable.has_default_sort_func

```
def has_default_sort_func()
```

Returns :	`TRUE`, if the model has a default sort function

The `has_default_sort_func()` method returns `TRUE` if the model has a default sort function. This is used primarily by `gtk.TreeViewColumn` to determine if a model can go back to the default state, or not.

Signals

The "sort–column–changed" gtk.TreeSortable Signal

```
def callback(treesortable, user_param1, ...)
```

treesortable :	the treesortable that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "sort–column–changed" signal is emitted when the `sort_column_changed()` method is called or the sort column is changed using the `set_sort_column_id()` method.

gtk.TreeStore

gtk.TreeStore — a model for tree widgets with columns

Synopsis

```
class gtk.TreeStore(gobject.GObject, gtk.TreeModel, gtk.TreeDragSource, gtk.TreeDragDest, gtk.T
    gtk.TreeStore(...)
    def set_value(iter, column, value)
    def set(iter, ...)
    def remove(iter)
    def insert(parent, position, row=None)
    def insert_before(parent, sibling, row=None)
    def insert_after(parent, sibling, row=None)
    def prepend(parent, row=None)
    def append(parent, row=None)
    def is_ancestor(iter, descendant)
    def iter_depth(iter)
    def clear()
    def iter_is_valid(iter)
    def reorder(parent, new_order)
    def swap(a, b)
    def move_after(iter, position)
    def move_before(iter, position)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.TreeStore (implements gtk.TreeModel, gtk.TreeDragSource, gtk.TreeDragDest, gtk.TreeSo
```

Description

A `gtk.TreeStore` is a model for multi–columned tree widgets. A `gtk.TreeStore` is a subclass of gobject.GObject and implements the `gtk.TreeModel`, `gtk.TreeDragSource`, `gtk.TreeDragDest` and `gtk.TreeSortable` interfaces.

Constructor

```
  gtk.TreeStore(...)
```

`...` :	one or more column types
Returns :	a new `gtk.TreeStore`

Creates a new tree store as with one or more columns each of the types passed in. As an example:

```
gtk.TreeStore(gobject.TYPE_INT, gobject.TYPE_STRING, gtk.gdk.Pixbuf)
```

will create a new `gtk.TreeStore` with three columns, of type int, string and `gtk.gdk.Pixbuf` respectively.

Methods

gtk.TreeStore.set_value

```
  def set_value(iter, column, value)
```

`iter` :	a `gtk.TreeIter` for the row being modified
`column` :	the column number to modify
`value` :	a new value for the cell

The `set_value()` method sets the data in the cell specified by *`iter`* and *`column`* to the value specified by *`value`*. The type of *`value`* must be convertible to the type of the column.

gtk.TreeStore.set

```
  def set(iter, ...)
```

`iter` :	a `gtk.TreeIter` for the row being modified
`...` :	one or more column ID–value pairs

The `set()` method sets the value of one or more cells in the row referenced by *`iter`*. The argument list following *`iter`* should contain pairs of integer column numbers followed by the value to be set. For example, to set column 0 with type `gobject.TYPE_STRING` to "Foo", you would write:

```
store.set(iter, 0, "Foo")
```

gtk.TreeStore.remove

```
  def remove(iter)
```

`iter` :	a `gtk.TreeIter`
Returns :	`None` in PyGTK 2.0. Returns `TRUE` in PyGTK 2.2 and above if *`iter`* is still valid.

The remove() method removes the row pointed to by *iter* from the treestore. After being removed, *iter* is set to the next valid row at that level, or invalidated if it previously pointed to the last one.

gtk.TreeStore.insert

```
    def insert(parent, position, row=None)
```

parent :	a gtk.TreeIter, or None
position :	the position to insert the new row
row :	an optional list or tuple containing column values (in order) to set on the row or None
Returns :	a gtk.TreeIterpointing to the new row

The insert() method inserts a new row at *position*. If *parent* is not None, then the row will be made a child of *parent*. Otherwise, the row will be created at the toplevel. If *position* is larger than the number of rows at that level, then the new row will be inserted to the end of the list. This method returns a gtk.TreeIter pointing at the new row. If *row* is not None it must be a tuple or list containing ordered column values that are used to set values in the columns of the row.

gtk.TreeStore.insert_before

```
    def insert_before(parent, sibling, row=None)
```

parent :	a gtk.TreeIter, or None
sibling :	a gtk.TreeIter, or None
row :	an optional list or tuple containing ordered column values to set on the row or None
Returns :	a gtk.TreeIterpointing to the new row

The insert_before() method inserts a new row before the row pointed to by *sibling*. If *sibling* is None, then the row will be appended to the children of the row pointed to by *parent*. If *parent* and *sibling* are None, the row will be appended to the toplevel. If both *sibling* and *parent* are set, then *parent* must be the parent of *sibling*. When *sibling* is set, *parent* is optional. This method returns a gtk.TreeIter pointing at the new row. If *row* is not None it must be a tuple or list containing ordered column values that are used to set values in the columns of the row.

gtk.TreeStore.insert_after

```
    def insert_after(parent, sibling, row=None)
```

parent :	a gtk.TreeIter, or None
sibling :	a gtk.TreeIter, or None
row :	a tuple or list containing ordered column values to be set in the new row
Returns :	a gtk.TreeIterpointing to the new row

The insert_after() method inserts a new row after the row pointed to by *sibling*. If *sibling* is None, then the row will be prepended to the beginning of the children of *parent*. If *parent* and *sibling* are None, then the row will be prepended to the toplevel. If both *sibling* and *parent* are set, *parent* must be the parent of *sibling*. When *sibling* is set, *parent* is optional. This method returns a gtk.TreeIter pointing at the new row. If *row* is not None it must be a tuple or list containing ordered column values that are used to set values in the columns of the row.

gtk.TreeStore.prepend

```
    def prepend(parent, row=None)
```

parent :	a gtk.TreeIter, or None
row :	a tuple or list containing ordered column values to be set in the new row
Returns :	a gtk.TreeIterpointing to the new row

The `prepend`() method prepends a new row to the treestore. If *parent* is not `None`, the new row will be prepended before the first child of *parent*, otherwise it will prepend a row to the top level. This method returns a gtk.TreeIter pointing at the new row. If *row* is not `None` it must be a tuple or list containing ordered column values that are used to set values in the columns of the row.

gtk.TreeStore.append

```
    def append(parent, row=None)
```

parent :	a gtk.TreeIter, or None
row :	a tuple or list containing ordered column values to be set in the new row
Returns :	a gtk.TreeIterpointing to the new row

The `append`() method appends a new row to the treestore. If *parent* is not `None`, the new row will be prepended after the last child of *parent*, otherwise it will append a row to the top level. This method returns a gtk.TreeIter pointing at the new row. If *row* is not `None` it must be a tuple or list containing ordered column values that are used to set values in the columns of the row.

gtk.TreeStore.is_ancestor

```
    def is_ancestor(iter, descendant)
```

iter :	a gtk.TreeIter
descendant :	a gtk.TreeIter
Returns :	`TRUE`, if *iter* is an ancestor of *descendant*

The `is_ancestor`() method returns `TRUE` if the row pointed to by *iter* is an ancestor of the row pointed to by *descendant*. That is, *iter* is the parent (or grandparent or great−grandparent) of *descendant*.

gtk.TreeStore.iter_depth

```
    def iter_depth(iter)
```

iter :	a gtk.TreeIter
Returns :	the depth of *iter*

The `iter_depth()` method returns the depth of the row pointed to by *iter*. This will be 0 for anything on the root level, 1 for anything down a level, etc.

gtk.TreeStore.clear

```
    def clear()
```

The `clear`() method removes all rows from the treestore.

gtk.TreeStore.iter_is_valid

```
    def iter_is_valid(iter)
```

iter :	a gtk.TreeIter.
Returns :	TRUE if *iter* is valid for the tree store,

Note

This method is available in PyGTK 2.2 and above.

The `iter_is_valid()` method returns TRUE if *iter* is a valid gtk.TreeIter for the tree store.

Warning

This function is slow. Only use it for debugging and/or testing purposes.

gtk.TreeStore.reorder

```
    def reorder(parent, new_order)
```

parent :	a gtk.TreeIter.
new_order :	a list of integers mapping the new position of each child to its old position before the re–ordering, i.e. *new_order*[newpos] = oldpos.

Note

This method is available in PyGTK 2.2 and above.

The `reorder()` method reorders the children of the tree store node pointed to by *parent* to match the order of the list of row numbers contained in *new_order*. Note that this method only works with unsorted stores.

gtk.TreeStore.swap

```
    def swap(a, b)
```

a :	a gtk.TreeIter.
b :	another gtk.TreeIter.

Note

This method is available in PyGTK 2.2 and above.

The `swap()` method swaps the tree store nodes pointed to by *a* and *b* in the same level of the tree store. Note that this method only works with unsorted stores.

gtk.TreeStore.move_after

```
    def move_after(iter, position)
```

iter :	a gtk.TreeIter.
position :	a second gtk.TreeIter or None.

Note

This method is available in PyGTK 2.2 and above.

The `move_after()` method moves the tree store node specified by *`iter`* to the position after the node specified by *`position`*. *`iter`* and *`position`* should be in the same level. Note that this method only works with unsorted stores. If *`position`* is `None`, *`iter`* will be moved to the start of the level.

gtk.TreeStore.move_before

```
    def move_before(iter, position)
```

`iter` :	a `gtk.TreeIter`.
`position` :	a `gtk.TreeIter` or `None`.

Note

This method is available in PyGTK 2.2 and above.

The move_before() method moves the tree store node pointed to by *`iter`* to the position before the node specified by *`position`*. *`iter`* and *`position`* should be in the same level. Note that this method only works with unsorted stores. If *`position`* is `None`, *`iter`* will be moved to the end of the level.

gtk.TreeView

gtk.TreeView — a widget for displaying both trees and lists.

Synopsis

```
class gtk.TreeView(gtk.Container):
    gtk.TreeView(model=None)
    def get_model()
    def set_model(model=None)
    def get_selection()
    def get_hadjustment()
    def set_hadjustment(adjustment)
    def get_vadjustment()
    def set_vadjustment(adjustment)
    def get_headers_visible()
    def set_headers_visible(headers_visible)
    def columns_autosize()
    def set_headers_clickable(active)
    def set_rules_hint(setting)
    def get_rules_hint()
    def append_column(column)
    def remove_column(column)
    def insert_column(column, position)
    def insert_column_with_attributes(position, title, cell, ...)
    def insert_column_with_data_func(position, title, cell, func, data=None)
    def get_column(n)
    def get_columns()
    def move_column_after(column, base_column)
    def set_expander_column(column)
    def get_expander_column()
    def set_column_drag_function(func, user_data)
    def scroll_to_point(tree_x, tree_y)
    def scroll_to_cell(path, column, use_align, row_align, col_align)
    def row_activated(path, column)
    def expand_all()
    def collapse_all()
    def expand_to_path(path)
    def expand_row(path, open_all)
    def collapse_row(path)
    def map_expanded_rows(func, data)
    def row_expanded(path)
    def set_reorderable(reorderable)
    def get_reorderable()
    def set_cursor(path, focus_column=None, start_editing=False)
    def set_cursor_on_cell(path, focus_column=None, focus_cell=None, start_editing=False)
    def get_cursor()
    def get_bin_window()
    def get_path_at_pos(x, y)
    def get_cell_area(path, column)
    def get_background_area(path, column)
    def get_visible_rect()
    def widget_to_tree_coords(wx, wy)
    def tree_to_widget_coords(tx, ty)
    def enable_model_drag_source(start_button_mask, targets, actions)
    def enable_model_drag_dest(targets, actions)
    def unset_rows_drag_source()
    def unset_rows_drag_dest()
    def set_drag_dest_row(path, pos)
    def get_drag_dest_row()
    def get_dest_row_at_pos(x, y)
    def create_row_drag_icon(path)
    def set_enable_search(enable_search)
    def get_enable_search()
    def get_search_column()
    def set_search_column(column)
    def set_search_equal_func(func=None, user_data=None)
    def get_fixed_height_mode()
    def set_fixed_height_mode(enable)
    def get_hover_selection()
    def set_hover_selection(hover)
    def get_hover_expand()
    def set_hover_expand(expand)
    def set_row_separator_func(func=None, user_data=None)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.TreeView
```

Properties

"enable–search"	Read–Write	If `TRUE`, the user can search through columns interactively. Default value: `TRUE`
"expander–column"	Read–Write	The column for the expander
"fixed–height–mode"	Read–Write	If `TRUE`, assume all rows have the same height thereby speeding up display. Default value: `FALSE`. Available in GTK+ 2.4 and above.
"hadjustment"	Read–Write	The horizontal Adjustment for the widget
"headers–clickable"	Write	If `TRUE`, the column headers respond to click events. Default value: `FALSE`
"headers–visible"	Read–Write	If `TRUE`, show the column header buttons. Default value: `TRUE`
"hover–expand"	Read–Write	If `TRUE`, rows expand or collapse if the pointer moves over them. This mode is primarily intended for treeviews in popups, e.g. in `gtk.ComboBox` or `gtk.EntryCompletion`. Default value: `FALSE`. Available in GTK+ 2.6 and above.
"hover–selection"	Read–Write	If `TRUE`, the selected row follows the pointer. Currently, this works only for the selection modes `gtk.SELECTION_SINGLE` and `gtk.SELECTION_BROWSE`. This mode is primarily intended for treeviews in popups, e.g. in `gtk.ComboBox` or `gtk.EntryCompletion`. Default value: `FALSE`. Available in GTK+ 2.6 and above.
"model"	Read–Write	The model for the tree view
"reorderable"	Read–Write	If `TRUE`, the view is reorderable. Default value: `FALSE`.
"rules–hint"	Read–Write	If `TRUE`, hint to the theme engine to draw rows in alternating colors. Default value: `FALSE`.
"search–column"	Read–Write	The model column to search when searching through code. Allowed values: >= −1. Default value: −1
"vadjustment"	Read–Write	The vertical Adjustment for the widget

Style Properties

"allow–rules"	Read	If `TRUE`, allow drawing of alternating color rows. Default value: `TRUE`
"even–row–color"	Read	The `gtk.gdk.Color` to use for even rows. Available in GTK+ 2.2 and above.
"expander–size"	Read	The size of the expander arrow. Allowed values: >= 0. Default value: 12
"horizontal–separator"	Read	The horizontal space between cells. Must be an even number. Allowed values: >= 0. Default value: 2
"indent–expanders"	Read	If `TRUE`, the expanders are indented.
"odd–row–color"	Read	The `gtk.gdk.Color` to use for odd rows. Available in GTK+ 2.2 and above.
"vertical–separator"	Read–Write	The vertical space between cells. Must be an even number. Allowed values: >= 0. Default value: 2

Signal Prototypes

"columns–changed"	`def callback(`*`treeview`*`, `*`user_param1`*`, ...)`
"cursor–changed"	`def callback(`*`treeview`*`, `*`user_param1`*`, ...)`
"expand–collapse–cursor–row"	`def callback(`*`treeview`*`, `*`logical`*`, `*`expand`*`, `*`open_all`*`, `*`user_param1`*`, ...)`
"move–cursor"	

	`def callback(treeview, step, count, user_param1, ...)`
"row−activated"	`def callback(treeview, path, view_column, user_param1, ...)`
"row−collapsed"	`def callback(treeview, iter, path, user_param1, ...)`
"row−expanded"	`def callback(treeview, iter, path, user_param1, ...)`
"select−all"	`def callback(treeview, user_param1, ...)`
"select−cursor−parent"	`def callback(treeview, user_param1, ...)`
"select−cursor−row"	`def callback(treeview, start_editing, user_param1, ...)`
"set−scroll−adjustments"	`def callback(treeview, hadjustment, vadjustment, user_param1, ...)`
"start−interactive−search"	`def callback(treeview, user_param1, ...)`
"test−collapse−row"	`def callback(treeview, iter, path, user_param1, ...)`
"test−expand−row"	`def callback(treeview, iter, path, user_param1, ...)`
"toggle−cursor−row"	`def callback(treeview, user_param1, ...)`
"unselect−all"	`def callback(treeview, user_param1, ...)`

Description

A `gtk.TreeView` widget is used to display the contents of any model implementing the `gtk.TreeModel` interface. The tree models provided standard with `GTK+` and `PyGTK` are:

- `gtk.ListStore`
- `gtk.TreeStore`
- `gtk.TreeModelSort`

In addition, `PyGTK` provides `gtk.GenericTreeModel` that allows you to create your own tree model entirely in Python.

The `gtk.TreeView` uses columns and cell renderers to actually display the model information. `GTK+` and `PyGTK` provides the `gtk.TreeViewColumn` to manage the display of a column and the following cell renderers:

- `gtk.CellRendererPixbuf`
- `gtk.CellRendererText`
- `gtk.CellRendererToggle`

In addition, `PyGTK` provides the `gtk.GenericCellRenderer` that allows you to create your own cell renderers entirely in Python.

Constructor

```
    gtk.TreeView(model=None)
```

`model` :	the tree model to display

Returns :	A new `gtk.TreeView` widget.

Creates a new `gtk.TreeView` widget displaying the model specified by *`model`*.

Methods

gtk.TreeView.get_model

```
    def get_model()
```

Returns :	the current `gtk.TreeModel`, or `None` if none is currently being used.

The `get_model`() method returns the value of the "model" property containing the model the `gtk.TreeView` is displaying or `None` there is no the model.

gtk.TreeView.set_model

```
    def set_model(model=None)
```

`model` :	the new tree model to use with the treeview

The `set_model`() method sets the "model" property for the treeview to the value of *`model`*. If the treeview already has a model set, this method will remove it before setting the new model. If *`model`* is `None`, it will unset the old model.

gtk.TreeView.get_selection

```
    def get_selection()
```

Returns :	A `gtk.TreeSelection` object.

The `get_selection`() method returns the current `gtk.TreeSelection` associated with the treeview.

gtk.TreeView.get_hadjustment

```
    def get_hadjustment()
```

Returns :	a `gtk.Adjustment` object, or `None` if none is currently being used.

The `get_hadjustment`() method returns the value of the "hadjustment" property that contains the current horizontal `gtk.Adjustment` object or `None` (if no horizontal adjustment is being used).

gtk.TreeView.set_hadjustment

```
    def set_hadjustment(adjustment)
```

`adjustment` :	the `gtk.Adjustment` to set

The `set_hadjustment`() method sets the "hadjustment" property to the value of *`adjustment`* that must be a `gtk.Adjustment` object.

gtk.TreeView.get_vadjustment

```
    def get_vadjustment()
```

Returns :	a `gtk.Adjustment` object, or `None` if none is currently being used.

The `get_vadjustment()` method returns the value of the "vadjustment" property that contains the horizontal `gtk.Adjustment` or None if there is no vertical adjustment.

gtk.TreeView.set_vadjustment

```
def set_vadjustment(adjustment)
```

`adjustment` :	the `gtk.Adjustment` to set

The `set_vadjustment()` method sets the "vadjustment" property to the value of *`adjustment`*. The new `gtk.Adjustment` replaces the current vertical adjustment.

gtk.TreeView.get_headers_visible

```
def get_headers_visible()
```

Returns :	`TRUE` if the headers are visible.

The `get_headers_visible()` method returns the value of the "headers−visible" property. If "headers−visible" is `TRUE` the headers on the treeview are visible.

gtk.TreeView.set_headers_visible

```
def set_headers_visible(headers_visible)
```

`headers_visible` :	if `TRUE` the headers are visible

The `set_headers_visible()` method sets the "headers−visible" property to the value of *`headers_visible`*. If *`headers_visible`* is `TRUE` the headers will be displayed.

gtk.TreeView.columns_autosize

```
def columns_autosize()
```

The `columns_autosize()` method resizes all columns to their optimal width. Only works after the treeview has been realized.

gtk.TreeView.set_headers_clickable

```
def set_headers_clickable(active)
```

`active` :	if `TRUE` the headers are clickable

The `set_headers_clickable()` method sets the "headers−clickable" property to the value of *`active`*. If *`active`* is `TRUE` the column title buttons can be clicked.

gtk.TreeView.set_rules_hint

```
def set_rules_hint(setting)
```

`setting` :	if `TRUE` the tree requires reading across rows

The `set_rules_hint()` method sets the "rules−hint" property to the value of *`setting`*. If *`setting`* is `TRUE` it indicates that the user interface for your application requires users to read across tree rows and associate cells with one another. By default, the tree will be rendered with alternating row colors. Do *not* use it just because you prefer the appearance of the ruled tree; that's a question for the theme. Some themes will draw tree rows in alternating colors even when rules are turned off, and users who prefer that appearance all the time can choose those themes. You should call this method only as a *semantic* hint to the theme engine that your tree makes alternating colors useful from a functional standpoint (since it has lots of columns, generally).

gtk.TreeView.get_rules_hint

```
def get_rules_hint()
```

Returns :	`TRUE` if rules are useful for the user of this tree

The `get_rules_hint()` returns the value of the "rules−hint" property. See the `set_rules_hint()` method for more information on the use of "rules−hint".

gtk.TreeView.append_column

```
def append_column(column)
```

`column` :	the `gtk.TreeViewColumn` to add.
Returns :	the number of columns in *`tree_view`* after appending.

The `append_column()` method appends the specified *`column`* to the list of columns and returns the new number of columns in the treeview.

gtk.TreeView.remove_column

```
def remove_column(column)
```

`column` :	the `gtk.TreeViewColumn` to remove.
Returns :	the number of columns in the treeview after the column removal.

The `remove_column()` method removes the specified *`column`* from the treeview.

gtk.TreeView.insert_column

```
def insert_column(column, position)
```

`column` :	the `gtk.TreeViewColumn` to be inserted.
`position` :	the position to insert *`column`*.
Returns :	the number of columns in the treeview after the insertion.

The `insert_column()` method inserts the specified *`column`* into the treeview at the location specified by *`position`*. If *`position`* is −1, then the column is inserted at the end.

gtk.TreeView.insert_column_with_attributes

```
def insert_column_with_attributes(position, title, cell, ...)
```

`position` :	the position to insert the new column in.
`title` :	the title to set the header to.
`cell` :	the `gtk.CellRenderer`.
`...` :	optional keyword−value arguments
Returns :	the number of columns in *`tree_view`* after the insertion.

The `insert_columns_with_attributes()` method creates a new `gtk.TreeViewColumn` and inserts it into the treeview at the location specified by *`position`* with the column title specified by *`title`* and using the `gtk.CellRenderer` specified by *`cell`*. If *`position`* is −1, then the newly created column is inserted at the end. The column is initialized with the optional attributes passed as keyword−value pairs (e.g. text=0, foreground=2). See the `gtk.TreeViewColumn.add_attribute()` method for more information.

gtk.TreeView.insert_column_with_data_func

```
def insert_column_with_data_func(position, title, cell, func, data=None)
```

`position` :	the position to insert, −1 for append
`title` :	the column title
`cell` :	a cell renderer for the column
`func` :	the function or method to set attributes of the cell renderer
`data` :	the data to pass with *`func`*
Returns :	the number of columns in the treeview after the insertion

The `insert_column_with_data_func()` method is a convenience function that inserts a new column into the treeview at the location specified by *`position`* with the specified *`title`* and the cell renderer specified by *`cell`* and using the function or method specified by *`func`* to set cell renderer attributes (normally using data from the model). The signature of func is:

```
  def celldatafunction(column, cell, model, iter, user_data)

  def celldatamethod(self, column, cell, model, iter, user_data)
```

where *`column`* is the `gtk.TreeViewColumn` in the treeview, *`cell`* is the `gtk.CellRenderer` for *`column`*, *`model`* is the `gtk.TreeModel` for the treeview and *`iter`* is the `gtk.TreeIter` pointing at the row. See the `gtk.TreeViewColumn.set_cell_data_func()` and `gtk.TreeViewColumn.pack_start()` methods for more detail.

gtk.TreeView.get_column

```
def get_column(n)
```

`n` :	the position of the column, counting from 0.
Returns :	the `gtk.TreeViewColumn`, or `None` if the position is outside the range of columns.

The `get_column()` method returns the `gtk.TreeViewColumn` at the specified *`position`* in the treeview.

gtk.TreeView.get_columns

```
def get_columns()
```

Returns :	a list of `gtk.TreeViewColumn` s

The `get_columns()` method returns a list of all the `gtk.TreeViewColumn` objects currently in the treeview.

gtk.TreeView.move_column_after

```
def move_column_after(column, base_column)
```

`column` :	the `gtk.TreeViewColumn` to be moved.
`base_column` :	the `gtk.TreeViewColumn` *`column`* is to be after, or `None`.

The `move_column_after()` method moves the `gtk.TreeViewColumn` specified by *`column`* to be after the treeview column specified by *`base_column`*. If *`base_column`* is `None`, then *`column`* is placed in the first position.

gtk.TreeView.set_expander_column

```
def set_expander_column(column)
```

`column` :	the column to draw the expander arrow at or`None`.

The `set_expander_column()` method sets the "expander−column" property to the value of *`column`* which must be a `gtk.TreeViewColumn` in the treeview. If *`column`* is `None`, then the expander arrow is always at the first visible column.

gtk.TreeView.get_expander_column

```
def get_expander_column()
```

Returns :	the expander column.

The `get_expander_column()` method returns the value of the "expander−column" property that contains the current expander column i.e. the column that has the expander arrow drawn next to it.

gtk.TreeView.set_column_drag_function

```
def set_column_drag_function(func, user_data)
```

`func` :	A function to determine which columns are reorderable, or `None`.
`user_data` :	User data to be passed to *`func`*, or `None`

Note

This method is available in PyGTK 2.4 and above.

The `set_column_drag_function()` method sets the user function specified by *`func`* for determining where a column may be dropped when dragged. The user function is called on every column pair in turn at the beginning of a column drag to determine where a drop can take place. The signature of *`func`* is:

```
def func(tree_view, column, prev_column, next_column, data)
```

where *`tree_view`* is the `gtk.TreeView`, *`column`* is the `gtk.TreeViewColumn` being dragged, prev_column and next_column are the two `gtk.TreeViewColumn` objects bracketing the drop spot, and *`data`* is *`user_data`*. If *`prev_column`* or *`next_column`* is `None`, then the drop is at an edge. If *`func`* is `None`, the user drag function is removed and the `gtk.TreeView` reverts to the default behavior of allowing any reorderable column to be dropped anywhere.

gtk.TreeView.scroll_to_point

```
def scroll_to_point(tree_x, tree_y)
```

`tree_x` :	the X coordinate of new top−left pixel of visible area, or −1
`tree_y` :	the Y coordinate of new top−left pixel of visible area, or −1

The `scroll_to_point()` method scrolls the treeview so that the top−left corner of the visible area is at the location specified by *`tree_x`* and *`tree_y`*, where *`tree_x`* and *`tree_y`* are specified in tree window coordinates. The treeview must be realized before this method is called. If it isn't, you should use the `scroll_to_cell()` method instead. If either *`tree_x`* or *`tree_y`* are −1, there is no scrolling in that direction.

gtk.TreeView.scroll_to_cell

```
    def scroll_to_cell(path, column=None, use_align=FALSE, row_align=0.0, col_align=0.0)
```

`path` :	the path of the row to move to
`column` :	the `gtk.TreeViewColumn` to move horizontally to, or `None`.
`use_align` :	if `TRUE` use the alignment arguments
`row_align` :	the vertical alignment of the row specified by *`path`*.
`col_align` :	the horizontal alignment of the column specified by *`column`*.

The `scroll_to_cell`() method scrolls the treeview display to the position specified by *`column`* and *`path`*. If *`column`* is `None`, no horizontal scrolling occurs. The alignment parameters specified by *`row_align`* and *`col_align`* determines where *`column`* is placed within the treeview. The values of *`col_align`* and *`row_align`* range from 0.0 to 1.0. The alignment values specify the fraction of display space that is to the left of or above the cell. If *`use_align`* is `FALSE`, the alignment arguments are ignored, and the tree does the minimum amount of work to scroll the cell onto the screen. This means that the cell will be scrolled to the edge closest to it's current position. If the cell is currently visible on the screen, nothing is done. This method only works if the model is set, and *`path`* is a valid row in the model.

gtk.TreeView.row_activated

```
    def row_activated(path, column)
```

`path` :	the tree path of the row of the cell to be activated.
`column` :	the `gtk.TreeViewColumn` of the cell to be activated.

The `row_activated`() method activates the cell determined by *`path`* and *`column`*.

gtk.TreeView.expand_all

```
    def expand_all()
```

The `expand_all`() method recursively expands all nodes in the treeview.

gtk.TreeView.collapse_all

```
    def collapse_all()
```

The `collapse_all`() method recursively collapses all visible, expanded nodes in the treeview.

gtk.TreeView.expand_to_path

```
    def expand_to_path(path)
```

`path` :	a path to a row.

Note

This method is available in PyGTK 2.2 and above.

The `expand_to_row`() method expands the row with the tree path specified by *`path`*. This will also expand all parent rows of *`path`* as necessary.

gtk.TreeView.expand_row

```
def expand_row(path, open_all)
```

path :	the path to a row
open_all :	if `TRUE` recursively expand, otherwise just expand immediate children

The `expand_row()` method opens the row specified by *`path`* so its children are visible. If *`open_all`* is `TRUE` all rows are expanded, otherwise only the immediate children of *`path`* are expanded.

gtk.TreeView.collapse_row

```
def collapse_row(path)
```

path :	the path to a row

The `collapse_row()` method collapses the row specified by *`path`* (hides its child rows, if they exist).

gtk.TreeView.map_expanded_rows

```
def map_expanded_rows(func, data)
```

func :	A function to be called
data :	User data to be passed to the function.

Note

This method is available in PyGTK 2.2 and above.

The `map_expanded_rows()` method calls the function specified by *`func`* on all expanded rows passing *`data`* as an argument.

gtk.TreeView.row_expanded

```
def row_expanded(path)
```

path :	the path to a row to test the expansion state.
Returns :	`TRUE` if `path` is expanded.

The `row_expanded()` method returns `TRUE` if the node pointed to by *`path`* is expanded.

gtk.TreeView.set_reorderable

```
def set_reorderable(reorderable)
```

reorderable :	if `TRUE`, the tree can be reordered.

The `set_reorderable()` method sets the "reorderable" property to the value of *`reorderable`*. This method is a convenience method to allow you to reorder models that support the `gtk.TreeDragSource` and the `gtk.TreeDragDest` interfaces. Both `gtk.TreeStore` and `gtk.ListStore` support these. If *`reorderable`* is `TRUE`, then the user can reorder the model by dragging and dropping rows. The application can listen to these changes by connecting to the model's signals.

Note

This function does not give you any degree of control over the order –– any reordering is allowed. If more control is needed, you should probably handle drag and drop manually.

gtk.TreeView.get_reorderable

```
    def get_reorderable()
```

Returns :	`TRUE` if the tree can be reordered.

The `get_reorderable()` method returns the value of the "reorderable" property that determines if the user can reorder the tree via drag–and–drop. See the `set_reorderable()` method for more information.

gtk.TreeView.set_cursor

```
    def set_cursor(path, focus_column=None, start_editing=False)
```

`path` :	a tree path
`focus_column` :	a `gtk.TreeViewColumn`, or `None`
`start_editing` :	`if TRUE` the specified cell should start being edited.

The `set_cursor()` method sets the current keyboard focus to be at the row specified by *`path`*, and selects it. This is useful when you want to focus the user's attention on a particular row. If *`column`* is not `None`, then focus is given to the specified column. Additionally, if *`column`* is specified, and *`start_editing`* is `TRUE`, then editing should be started in the specified cell. This method is often followed by the `gtk.Widget.grab_focus()` method to give keyboard focus to the treeview. Please note that editing can only happen when the widget is realized.

gtk.TreeView.set_cursor_on_cell

```
    def set_cursor_on_cell(path, focus_column=None, focus_cell=None, start_editing=False)
```

`path` :	a tree path
`focus_column` :	a `gtk.TreeViewColumn`, or `None`
`focus_cell` :	a `gtk.CellRenderer`, or `None`
`start_editing` :	`TRUE` if the specified cell should start being edited.

Note

This method is available in PyGTK 2.2 and above.

The `set_cursor_on_cell()` method sets the current keyboard focus to be at the node specified by *`path`*, and selects it. This is useful when you want to focus the user's attention on a particular row. If *`focus_column`* is specified, focus is given to that column. If *`focus_column`* and *`focus_cell`* are specified, and *`focus_column`* contains 2 or more editable or activatable cells, then focus is given to the cell specified by *`focus_cell`*. Additionally, if *`focus_column`* is specified, and *`start_editing`* is `True`, editing should be started in the specified cell. This method is often followed by the `gtk.Widget.grab_focus()` method in order to give keyboard focus to the widget. Please note that editing can only happen when the widget is realized.

gtk.TreeView.get_cursor

```
    def get_cursor()
```

Returns :	a tuple containing the current cursor path and focus column.

The `get_cursor()` method returns a tuple containing the current path and focus column. If the cursor isn't currently set, the current path will be `None`. If no column currently has focus, the current focus column will be `None`.

gtk.TreeView.get_bin_window

```
def get_bin_window()
```

Returns :	a `gtk.gdk.Window`, or `None`

The `get_bin_window`() method returns the window that the treeview renders to or `None` if the treeview is not realized yet. This is used primarily to compare to the `event.window` attribute to confirm that the event on the treeview is on the right window.

gtk.TreeView.get_path_at_pos

```
def get_path_at_pos(x, y)
```

`x` :	The x position to be identified.
`y` :	The y position to be identified.
Returns :	a tuple containing: a tree path; a `gtk.TreeViewColumn` object; the X coordinate relative to the cell; and, the Y coordinate relative to the cell. If there is no path at the position `None` is returned.

The `get_path_at_pos`() method returns a tuple containing:

- the path at the specified point (*`x`*, *`y`*), relative to widget coordinates
- the `gtk.TreeViewColumn` at that point
- the X coordinate relative to the cell background
- the Y coordinate relative to the cell background

`x` and *`y`* are relative to the coordinates of an event on the treeview only when `event.window==treeview.get_bin_window()`. It is primarily used for popup menus. This method is only meaningful if the treeview is realized. This method returns `None` if there is no path at the position.

gtk.TreeView.get_cell_area

```
def get_cell_area(path, column)
```

`path` :	a tree path for the row
`column` :	a `gtk.TreeViewColumn` for the column
Returns :	rectangle

The `get_cell_area`() method returns the bounding `gtk.gdk.Rectangle` in tree window coordinates for the cell at the row specified by *`path`* and the column specified by *`column`*. If *`path`* points to a path not currently displayed, the *`y`* and *`height`* attributes of the rectangle will be 0. The sum of all cell rects does not cover the entire tree; there are extra pixels in between rows, for example. The returned rectangle is equivalent to the *`cell_area`* passed to the `gtk.CellRenderer.render()` method. This method is only valid if the treeview is realized.

gtk.TreeView.get_background_area

```
def get_background_area(path, column)
```

`path` :	a tree path for the row,
`column` :	a `gtk.TreeViewColumn` for the column
Returns :	a rectangle

The `get_background_area`() method returns the bounding `gtk.gdk.Rectangle` in tree window coordinates for the cell at the row specified by *`path`* and the column specified by *`column`*. If *`path`* points to a node not found in the tree, the *`y`* and *`height`* attributes of the rectangle will be 0. The returned rectangle is equivalent to the *background_area* passed to the `gtk.CellRenderer.render()`. These background areas tile to cover the entire tree window (except for the area used for header buttons). Contrast with the *cell_area*, returned by the `get_cell_area()` method, that returns only the cell itself, excluding the surrounding borders and the tree expander area.

gtk.TreeView.get_visible_rect

```
    def get_visible_rect()
```

Returns :	a rectangle

The `get_visible_rect()` method returns the bounding `gtk.gdk.Rectangle` for the currently visible region of the treeview widget, in tree coordinates. Convert to widget coordinates with the `tree_to_widget_coords()`. Tree coordinates start at 0,0 for row 0 of the tree, and cover the entire scrollable area of the tree.

gtk.TreeView.widget_to_tree_coords

```
    def widget_to_tree_coords(wx, wy)
```

wx :	the widget X coordinate
wy :	the widget Y coordinate
Returns :	a tuple containing the tree X and Y coordinates

The `widget_to_tree_coords()` method returns a tuple containing the tree X and Y coordinates for the widget coordinates specified by *wx* and *wy*. The tree coordinates cover the full scrollable area of the tree.

gtk.TreeView.tree_to_widget_coords

```
    def tree_to_widget_coords(tx, ty)
```

tx :	tree X coordinate
ty :	tree Y coordinate
Returns :	a tuple containing the widget X and Y coordinates

The `tree_to_widget_coords()` method returns a tuple containing the widget coordinates for the tree coordinates specified by *tx* and *ty*.

gtk.TreeView.enable_model_drag_source

```
    def enable_model_drag_source(start_button_mask, targets, actions)
```

start_button_mask :	the bitmask of buttons that can start the drag
targets :	a sequence of tuples containing target data
actions :	the possible actions for a drag

The `enable_model_drag_source()` method sets the treeview to start a drag operation when the user click and drags on a row. The value of *start_button_mask* is a combination of:

`gtk.gdk.SHIFT_MASK`	The Shift key.
`gtk.gdk.LOCK_MASK`	A Lock key (depending on the modifier mapping of the X server this may either be Caps Lock or Shift Lock).
`gtk.gdk.CONTROL_MASK`	The Control key.
`gtk.gdk.MOD1_MASK`	The fourth modifier key (it depends on the modifier mapping of the X server

	which key is interpreted as this modifier, but normally it is the Alt key).
`gtk.gdk.MOD2_MASK`	The fifth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
`gtk.gdk.MOD3_MASK`	The sixth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
`gtk.gdk.MOD4_MASK`	The seventh modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
`gtk.gdk.MOD5_MASK`	The eighth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
`gtk.gdk.BUTTON1_MASK`	The first mouse button.
`gtk.gdk.BUTTON2_MASK`	The second mouse button.
`gtk.gdk.BUTTON3_MASK`	The third mouse button.
`gtk.gdk.BUTTON4_MASK`	The fourth mouse button.
`gtk.gdk.BUTTON5_MASK`	The fifth mouse button.
`gtk.gdk.RELEASE_MASK`	Differentiates between (keyval, modifiers) pairs from key press and release events.
`gtk.gdk.MODIFIER_MASK`	all of the above

`targets` is a sequence (list or tuple) of tuples that contain information about the targets. The target data contains a string representing the drag type, target flags (a combination of `gtk.TARGET_SAME_APP` and `gtk.TARGET_SAME_WIDGET` or neither) and an application assigned integer ID used for identification purposes.

The value of *`actions`* is one of:

`gtk.gdk.ACTION_DEFAULT`, `gtk.gdk.ACTION_COPY`,	Copy the data.
`gtk.gdk.ACTION_MOVE`	Move the data, i.e. first copy it, then delete it from the source using the `DELETE` target of the X selection protocol.
`gtk.gdk.ACTION_LINK`	Add a link to the data. Note that this is only useful if source and destination agree on what it means.
`gtk.gdk.ACTION_PRIVATE`	Special action which tells the source that the destination will do something that the source doesn't understand.
`gtk.gdk.ACTION_ASK`	Ask the user what to do with the data.

gtk.TreeView.enable_model_drag_dest

```
    def enable_model_drag_dest(targets, actions)
```

`targets` :	a sequence of tuples containing target data
`actions` :	the possible actions for a drag

The `enable_model_drag_dest()` method sets the treeview to receive a drag drop.

`targets` is a sequence (list or tuple) of tuples that contain information about the targets. The target data contains a string representing the drag type, target flags (a combination of `gtk.TARGET_SAME_APP` and `gtk.TARGET_SAME_WIDGET` or neither) and an application assigned integer ID used for identification purposes.

The value of *`actions`* is one of:

	Copy the data.

`gtk.gdk.ACTION_DEFAULT`,	
`gtk.gdk.ACTION_COPY`,	
`gtk.gdk.ACTION_MOVE`	Move the data, i.e. first copy it, then delete it from the source using the `DELETE` target of the X selection protocol.
`gtk.gdk.ACTION_LINK`	Add a link to the data. Note that this is only useful if source and destination agree on what it means.
`gtk.gdk.ACTION_PRIVATE`	Special action which tells the source that the destination will do something that the source doesn't understand.
`gtk.gdk.ACTION_ASK`	Ask the user what to do with the data.

gtk.TreeView.unset_rows_drag_source

```
    def unset_rows_drag_source()
```

The `unset_rows_drag_source()` method unsets the drag source information.

gtk.TreeView.unset_rows_drag_dest

```
    def unset_rows_drag_dest()
```

The `unset_rows_drag_dest()` method unsets the drag destination information.

gtk.TreeView.set_drag_dest_row

```
    def set_drag_dest_row(path, pos)
```

`path` :	a tree path
`pos` :	a drop position

The `set_drag_dest_row()` method sets the treeview drag destination row to the value specified by *`path`* with the drop position specified by *`pos`*. The value of *`pos`* must be one of: `gtk.TREE_VIEW_DROP_BEFORE`, `gtk.TREE_VIEW_DROP_AFTER`, `gtk.TREE_VIEW_DROP_INTO_OR_BEFORE` or `gtk.TREE_VIEW_DROP_INTO_OR_AFTER`.

gtk.TreeView.get_drag_dest_row

```
    def get_drag_dest_row()
```

Returns :	a 2–tuple containing the tree path and the drop position relative to the tree path or `None`

The `get_drag_dest_row()` method returns a 2–tuple containing the path of the drag destination row and a drop position relative to the destination row. The drop position is one of: `gtk.TREE_VIEW_DROP_BEFORE`, `gtk.TREE_VIEW_DROP_AFTER`, `gtk.TREE_VIEW_DROP_INTO_OR_BEFORE` or `gtk.TREE_VIEW_DROP_INTO_OR_AFTER`. This method returns `None` if no drag destination row is set.

gtk.TreeView.get_dest_row_at_pos

```
    def get_dest_row_at_pos(x, y)
```

`x` :	the x coordinate of the position
`y` :	the y coordinate of the position
Returns :	

	a 2–tuple containing the path of the row and the drop position at the position specified by *x* and *y* or `None`

The `get_dest_row_at_pos()` method returns a 2–tuple containing the path of the row and the drop position relative to the row of the position specified by *x* and *y*. The drop position is one of: `gtk.TREE_VIEW_DROP_BEFORE`, `gtk.TREE_VIEW_DROP_AFTER`, `gtk.TREE_VIEW_DROP_INTO_OR_BEFORE` or `gtk.TREE_VIEW_DROP_INTO_OR_AFTER`.

gtk.TreeView.create_row_drag_icon

```
def create_row_drag_icon(path)
```

path :	a tree path
Returns :	a new pixmap of the drag icon.

The `create_row_drag_icon()` method creates a `gtk.gdk.Pixmap` representation of the row specified by *path*. This image is used for a drag icon.

gtk.TreeView.set_enable_search

```
def set_enable_search(enable_search)
```

enable_search :	if `TRUE`, the user can search interactively

The `set_enable_search()` method sets the "enable–search" property to the value of *enable_search*. If *enable_search* is `TRUE` the user can type in text to search through the tree interactively.

gtk.TreeView.get_enable_search

```
def get_enable_search()
```

Returns :	`TRUE` if the user can search interactively

The `get_enable_search()` method returns the value of the "enable–search" property. If "enable–search" is `TRUE` the tree allows interactive searching.

gtk.TreeView.get_search_column

```
def get_search_column()
```

Returns :	the column the interactive search code searches.

The `get_search_column()` method returns the value of the "search–column" property that is the column searched by the interactive search code.

gtk.TreeView.set_search_column

```
def set_search_column(column)
```

column :	the column to search

The `set_search_column()` method sets the "search–column" property to the value of *column*. The value of *column* is the column where the interactive search code should search. Additionally, this method turns on interactive searching (see the `set_enable_search()` method).

gtk.TreeView.set_search_equal_func

```
def set_search_equal_func(func=None, user_data=None)
```

func :	the compare function to use during the search or None
user_data :	user data to pass to *func*, or None

Note

This method is available in PyGTK 2.4 and above.

The `set_search_equal_func()` method sets the compare function for the interactive search capabilities to the function specified by *func*. If *user_data* is specified and not `None`, it is passed to *func*. If *func* is `None`, the default `gtk.TreeView` search equal function will be used. The signature of *func* is:

```
def func(model, column, key, iter, data)
```

where *model* is the `gtk.TreeModel` of the `gtk.TreeView`, *column* is the number of the column being searched (see the `set_search_column()` method for more information), *key* is the string being searched for, *iter* is a `gtk.TreeIter` pointing to the current candidate row and *data* is the context data *user_data*. *func* should return `FALSE` to indicate that the row matches the search criteria.

gtk.TreeView.get_fixed_height_mode

```
def get_fixed_height_mode()
```

Returns :	`TRUE` if fixed height mode is enabled.

Note

This method is available in PyGTK 2.6 and above.

The `get_fixed_height_mode()` method returns the value of the "fixed–height–mode" property. If the "fixed–height–mode" property is `TRUE`, all rows are assumed to be the same height.

gtk.TreeView.set_fixed_height_mode

```
def set_fixed_height_mode(enable)
```

enable :	if `TRUE` enable fixed height mode.

Note

This method is available in PyGTK 2.6 and above.

The `set_fixed_height_mode()` method sets the "fixed–height–mode" property to the value of *enable*. If *enable* is `TRUE` all rows are assumed to have the same height which speeds up `gtk.TreeView` displays. Only enable this option if all rows are the same height and all columns are of type `gtk.TREE_VIEW_COLUMN_FIXED` (see the GTK TreeViewColumn Sizing Constants).

gtk.TreeView.get_hover_selection

```
def get_hover_selection()
```

Returns :	`TRUE` if hover selection mode is enabled.

Note

This method is available in PyGTK 2.6 and above.

The `get_hover_selection()` method returns the value of the "hover−selection" property. If the "hover−selection" property is `TRUE` the selected row follows the pointer. See the `set_hover_selection()` method for more detail.

gtk.TreeView.set_hover_selection

```
def set_hover_selection(hover)
```

hover :	if `TRUE` enable hover selection mode.

Note

This method is available in PyGTK 2.6 and above.

The () method sets the "hover−selection" property to the value of *hover*. If *hover* is `TRUE` the hover selection mode is enables and the selected row follows the pointer. Currently, this works only for the selection modes `gtk.SELECTION_SINGLE` and `gtk.SELECTION_BROWSE` (see the GTK Selection Mode Constants).

gtk.TreeView.get_hover_expand

```
def get_hover_expand()
```

Returns :	`TRUE` if hover expand mode is enabled

Note

This method is available in PyGTK 2.6 and above.

The `get_hover_expand()` method returns the value of the "hover−expand" property. If the "hover−expand" property is `TRUE` rows expand or collapse if the pointer moves over them.

gtk.TreeView.set_hover_expand

```
def set_hover_expand(expand)
```

expand :	if `TRUE` enable hover expand mode.

Note

This method is available in PyGTK 2.6 and above.

The `set_hover_expand()` method sets the "hover−expand" property to the value of *expand*. If *expand* is `TRUE`, rows expand or collapse if the pointer moves over them.

gtk.TreeView.set_row_separator_func

```
def set_row_separator_func(func=None, user_data=None)
```

func :	the row separator function or `None`
user_data :	user data to pass to *func*, or `None`

Note

This method is available in PyGTK 2.6 and above.

The `set_row_separator_func()` method sets the row separator function to the function specified by *`func`*. The row separator function is used to determine if a row should be displayed as a separator. If *`user_data`* is specified and not `None`, it is passed to *`func`*. If *`func`* is `None`, no separators will be drawn. The signature of *`func`* is:

```
  def func(model, iter, data)
```

where *`model`* is the `gtk.TreeModel` of the `gtk.TreeView`, *`iter`* is a `gtk.TreeIter` pointing to the current candidate row and *`data`* is the context data *`user_data`*. *`func`* should return `TRUE` to indicate that the row is a separator.

A common way to implement this is to have a boolean column in *`model`*, whose values *`func`* returns.

Signals

The "columns–changed" gtk.TreeView Signal

```
    def callback(treeview, user_param1, ...)
```

`treeview` :	the treeview that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "columns–changed" signal is emitted when a column has been added to, removed from or moved in *`treeview`*.

The "cursor–changed" gtk.TreeView Signal

```
    def callback(treeview, user_param1, ...)
```

`treeview` :	the treeview that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "cursor–changed" signal is emitted when the cursor moves or is set.

The "expand–collapse–cursor–row" gtk.TreeView Signal

```
    def callback(treeview, logical, expand, open_all, user_param1, ...)
```

`treeview` :	the treeview that received the signal
`logical` :	if `TRUE`
`expand` :	if `TRUE` the row should be expanded
`open_all` :	if `TRUE` recursively expand all children
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled.

The "expand–collapse–cursor–row" signal is emitted when the row at the cursor needs to be expanded or collapsed.

The "move–cursor" gtk.TreeView Signal

```
  def callback(treeview, step, count, user_param1, ...)
```

treeview :	the treeview that received the signal
step :	the movement step size
count :	the number of steps to take
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled.

The "move–cursor" signal is emitted when the user moves the cursor using the **Right**, **Left**, **Up** or **Down** arrow keys or the **Page Up**, **Page Down**, **Home** and **End** keys.

The "row–activated" gtk.TreeView Signal

```
  def callback(treeview, path, view_column, user_param1, ...)
```

treeview :	the treeview that received the signal
path :	the path of the activated row
view_column :	the column in the activated row
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "row–activated" signal is emitted when the `row_activated()` method is called or the user double clicks a *treeview* row. "row–activated" is also emitted when a non–editable row is selected and one of the keys: **Space**, **Shift+Space**, **Return** or **Enter** is pressed.

The "row–collapsed" gtk.TreeView Signal

```
  def callback(treeview, iter, path, user_param1, ...)
```

treeview :	the treeview that received the signal
iter :	a `gtk.TreeIter` pointing to the row that collapsed
path :	the path of the row that collapsed
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "row–collapsed" signal is emitted when a row is collapsed by user of programmatic action.

The "row–expanded" gtk.TreeView Signal

```
  def callback(treeview, iter, path, user_param1, ...)
```

treeview :	the treeview that received the signal
iter :	a `gtk.TreeIter` pointing to the row that expanded
path :	the path of the row that expanded
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "row−expanded" signal is emitted when a row is expanded via user or programmatic action.

The "select−all" gtk.TreeView Signal

```
def callback(treeview, user_param1, ...)
```

treeview :	the treeview that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled.

The "select−all" signal is emitted when the user presses **Control+a** or **Control+/**.

The "select−cursor−parent" gtk.TreeView Signal

```
def callback(treeview, user_param1, ...)
```

treeview :	the treeview that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled.

The "select−cursor−parent" signal is emitted when the user presses **Backspace** while a row has the cursor.

The "select−cursor−row" gtk.TreeView Signal

```
def callback(treeview, start_editing, user_param1, ...)
```

treeview :	the treeview that received the signal
start_editing :	if `TRUE` the cell editing is started
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled.

The "select−cursor−row" signal is emitted when a non−editable row is selected and one of the keys: **Space**, **Shift+Space**, **Return** or **Enter** is pressed.

The "set−scroll−adjustments" gtk.TreeView Signal

```
def callback(treeview, hadjustment, vadjustment, user_param1, ...)
```

treeview :	the treeview that received the signal
hadjustment :	a horizontal `gtk.Adjustment`
vadjustment :	a vertical `gtk.Adjustment`
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled.

The "set−scroll−adjustments" signal is emitted when new horizontal or vertical adjustments are set on the *treeview*.

The "start−interactive−search" gtk.TreeView Signal

```
    def callback(treeview, user_param1, ...)
```

treeview :	the treeview that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled.

The "start−interactive−search" signal is emitted when the user presses **Control+f**.

The "test−collapse−row" gtk.TreeView Signal

```
    def callback(treeview, iter, path, user_param1, ...)
```

treeview :	the treeview that received the signal
iter :	the gtk.TreeIter pointing at the row to test.
path :	the path of the row to be tested
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	`TRUE` if the row should be collapsed.

The "test−collapse−row" signal is emitted when the row pointed to by *iter* and *path* is to be collapsed.

The "test−expand−row" gtk.TreeView Signal

```
    def callback(treeview, iter, path, user_param1, ...)
```

treeview :	the treeview that received the signal
iter :	the gtk.TreeIter pointing at the row to test.
path :	the path of the row to test
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	`TRUE` if the row should be expanded.

The "test−expand−row" signal is emitted when the row pointed to by *iter* and *path* is to be expanded.

The "toggle−cursor−row" gtk.TreeView Signal

```
    def callback(treeview, user_param1, ...)
```

treeview :	the treeview that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	`TRUE` if the signal was handled.

The "toggle−cursor−row" signal is emitted when the user presses **Control+Space**.

The "unselect−all" gtk.TreeView Signal

```
    def callback(treeview, user_param1, ...)
```

treeview :	the treeview that received the signal

user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	TRUE if the signal was handled.

The "unselect−all" signal is emitted when the user presses **Shift+Control+a** or **Shift+Control+/**.

gtk.TreeViewColumn

gtk.TreeViewColumn a visible column in a gtk.TreeView widget

Synopsis

```
class gtk.TreeViewColumn(gtk.Object):
    gtk.TreeViewColumn(title=None, cell_renderer=None, ...)
    def pack_start(cell, expand=TRUE)
    def pack_end(cell, expand=TRUE)
    def clear()
    def get_cell_renderers()
    def add_attribute(cell_renderer, attribute, column)
    def set_attributes(cell_renderer, ...)
    def set_cell_data_func(cell_renderer, func, func_data=None)
    def clear_attributes(cell_renderer)
    def set_spacing(spacing)
    def get_spacing()
    def set_visible(visible)
    def get_visible()
    def set_resizable(resizable)
    def get_resizable()
    def set_sizing(type)
    def get_sizing()
    def get_width()
    def get_fixed_width()
    def set_fixed_width(fixed_width)
    def set_min_width(min_width)
    def get_min_width()
    def set_max_width(max_width)
    def get_max_width()
    def clicked()
    def set_title(title)
    def get_title()
    def set_expand(expand)
    def get_expand()
    def set_clickable(active)
    def get_clickable()
    def set_widget(widget)
    def get_widget()
    def set_alignment(xalign)
    def get_alignment()
    def set_reorderable(reorderable)
    def get_reorderable()
    def set_sort_column_id(sort_column_id)
    def get_sort_column_id()
    def set_sort_indicator(setting)
    def get_sort_indicator()
    def set_sort_order(order)
    def get_sort_order()
    def cell_set_cell_data(tree_model, iter, is_expander, is_expanded)
    def cell_get_size()
    def cell_is_visible()
    def focus_cell(cell)
    def cell_get_position(cell_renderer)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.TreeViewColumn
```

Properties

"alignment"	Read–Write	The alignment of the column header text or widget
"clickable"	Read–Write	If `TRUE`, the header can be clicked
"expand"	Read–Write	If `TRUE`, the column can expand to take extra available space. Available in GTK+ 2.4 and above
"fixed–width"	Read–Write	The current fixed width of the column
"max–width"	Read–Write	The maximum allowed width of the column
"min–width"	Read–Write	The minimum allowed width of the column
"reorderable"	Read–Write	If `TRUE`, the column can be reordered around the headers
"resizable"	Read–Write	If `TRUE`, the user can resize the column
"sizing"	Read–Write	The resize mode of the column: `gtk.TREE_VIEW_COLUMN_GROW_ONLY`, `gtk.TREE_VIEW_COLUMN_AUTOSIZE` or `gtk.TREE_VIEW_COLUMN_FIXED`
"sort–indicator"	Read–Write	If `TRUE`, how a sort indicator
"sort–order"	Read–Write	The sort direction the sort indicator should indicate: `gtk.SORT_ASCENDING` or `gtk.SORT_DESCENDING`
"title"	Read–Write	The title to appear in the column header
"visible"	Read–Write	If `TRUE`, display the column
"widget"	Read–Write	The widget to put in the column header button instead of the column title
"width"	Read	The current width of the column

Signal Prototypes

"clicked"	`def callback(`*`treeviewcolumn`*`, `*`user_param1`*`, ...)`

Description

The `gtk.TreeViewColumn` object is a visible column in a `gtk.TreeView` widget. A `gtk.TreeViewColumn` manages the display of the header and the cells using a `gtk.CellRenderer`.

In PyGTK 2.4 `gtk.TreeViewColumn` implementes the `gtk.CellLayout` interface.

Constructor

```
gtk.TreeViewColumn(title=None, cell_renderer=None, ...)
```

`title` :	the header title string
`cell_renderer` :	a `gtk.CellRenderer`
`...` :	zero or more `attribute=column` pairs
Returns :	A newly created `gtk.TreeViewColumn`.

Creates a new `gtk.TreeViewColumn` with the header string specified by *`title`* and using the `gtk.CellRenderer` specified by *`cell_renderer`*. Zero or more `attribute=column` pairs may be added to specify from which tree model column to retrieve the attribute value. For example:

```
column = gtk.TreeViewColumn('Column Title', cell_renderer, text=0, foreground=1)
```

will retrieve the 'text' attribute values for the cell renderer from column 0 in the treemodel and the 'foreground' attribute values from column 1. See the `add_attribute()` method for more detail.

Methods

gtk.TreeViewColumn.pack_start

```
def pack_start(cell, expand=TRUE)
```

cell :	a `gtk.CellRenderer`.
expand :	if `TRUE` *`cell`* is to be given the extra space allocated to the cell box.

The `pack_start()` method packs the `gtk.CellRenderer` specified by *`cell`* into the beginning column. If *`expand`* is `TRUE`, then the *`cell`* is allocated a share of all available space that the column is allocated. Note *`expand`* defaults to `TRUE` in PyGTK 2.4.

gtk.TreeViewColumn.pack_end

```
def pack_end(cell, expand=TRUE)
```

cell :	a `gtk.CellRenderer`.
expand :	if `TRUE` *`cell`* is to be given the extra space allocated to the cell box.

The `pack_end()` method packs the `gtk.CellRenderer` specified by *`cell`* into the column. If *`expand`* is `TRUE`, then the *`cell`* is allocated a share of all available space that the column is allocated. Note *`expand`* defaults to `TRUE` in PyGTK 2.4.

gtk.TreeViewColumn.clear

```
def clear()
```

The `clear()` method unsets all the mappings on all renderers on the column.

gtk.TreeViewColumn.get_cell_renderers

```
def get_cell_renderers()
```

Returns :	a list of `gtk.CellRenderer` objects

The `get_cell_renderers()` method returns a list containing all the cell renderers in the column, in no particular order.

gtk.TreeViewColumn.add_attribute

```
    def add_attribute(cell_renderer, attribute, column)
```

`cell_renderer` :	the `gtk.CellRenderer` to set attributes on
`attribute` :	an attribute on the renderer
`column` :	the column position on the model to get the attribute from.

The `add_attribute`() method adds an attribute mapping to the list in the tree column. The *`column`* is the column of the tree model to get a value from, and the *`attribute`* is the parameter on *`cell_renderer`* to be set from the value. So for example if column 2 of the model contains strings, you could have the "text" attribute of a `gtk.CellRendererText` get its values from column 2.

gtk.TreeViewColumn.set_attributes

```
    def set_attributes(cell_renderer, ...)
```

`cell_renderer` :	the `gtk.CellRenderer` we're setting the attributes of
`...` :	zero or more `attribute=column` pairs

The `set_attributes`() method sets the attribute locations of the `gtk.CellRenderer` specified by *`cell_renderer`* using the `attribute=column` pairs (e.g. text=0, foreground=1). See the `add_attribute()` method for more detail. All existing cell attributes are removed, and replaced with the new attributes.

gtk.TreeViewColumn.set_cell_data_func

```
    def set_cell_data_func(cell_renderer, func, func_data=None)
```

`cell_renderer` :	a `gtk.CellRenderer`
`func` :	the function or method to use.
`func_data` :	the user data to pass when calling *`func`*.

The `set_cell_data_func`() method sets the data function (or method) to use for the column `gtk.CellRenderer` specified by *`cell_renderer`*. This function (or method) is used instead of the standard attribute mappings for setting the column values, and should set the attributes of the cell renderer as appropriate. *`func`* may be `None` to remove the current data function. The signature of *`func`* is:

```
     def celldatafunction(column, cell, model, iter, user_data)

     def celldatamethod(self, column, cell, model, iter, user_data)
```

where *`column`* is the `gtk.TreeViewColumn` in the treeview, *`cell`* is the `gtk.CellRenderer` for *`column`*, *`model`* is the `gtk.TreeModel` for the treeview and *`iter`* is the `gtk.TreeIter` pointing at the row.

gtk.TreeViewColumn.clear_attributes

```
    def clear_attributes(cell_renderer)
```

`cell_renderer` :	a `gtk.CellRenderer` to clear the attribute mapping on.

The `clear_attributes`() method clears all existing attributes previously set with the `set_attributes()` method.

gtk.TreeViewColumn.set_spacing

```
    def set_spacing(spacing)
```

`spacing` :	the distance between cell renderers in pixels.

The `set_spacing()` method sets the spacing field of the treeview column, which is the number of pixels to place between cell renderers packed into it.

gtk.TreeViewColumn.get_spacing

```
    def get_spacing()
```

Returns :	the spacing of the treeview column.

The `get_spacing()` method returns the spacing of the treeview column.

gtk.TreeViewColumn.set_visible

```
    def set_visible(visible)
```

`visible` :	if `TRUE` the treeview column is visible.

The `set_visible()` method sets the "visible" property to the value of *`visible`*. If *`visible`* is `TRUE` the treeview column is visible

gtk.TreeViewColumn.get_visible

```
    def get_visible()
```

Returns :	`TRUE` if the column is visible

The `get_visible()` method returns the value of the "visible" property. If "visible" is `TRUE` the treeview column is visible.

gtk.TreeViewColumn.set_resizable

```
    def set_resizable(resizable)
```

`resizable` :	if `TRUE`, the column can be resized

The `set_resizable()` method sets the "resizable" property to the value of *`resizable`*. If *`resizable`* is `TRUE` the user can explicitly resize the column by grabbing the outer edge of the column button. If *`resizable`* is `TRUE` and the sizing mode of the column is `gtk.TREE_VIEW_COLUMN_AUTOSIZE`, the sizing mode is changed to `gtk.TREE_VIEW_COLUMN_GROW_ONLY`.

gtk.TreeViewColumn.get_resizable

```
    def get_resizable()
```

Returns :	`TRUE`, if the treeview column can be resized.

The `get_resizable()` method returns the value of the "resizable" property. If "resizable" is `TRUE`, the treeview column can be resized by the user.

gtk.TreeViewColumn.set_sizing

```
    def set_sizing(type)
```

type :	The treeview column sizing: `gtk.TREE_VIEW_COLUMN_GROW_ONLY`, `gtk.TREE_VIEW_COLUMN_AUTOSIZE` or `gtk.TREE_VIEW_COLUMN_FIXED`.

The `set_sizing`() method sets the "sizing" property to the value of *`type`*. The value of type must be one of:

`gtk.TREE_VIEW_COLUMN_GROW_ONLY`	Columns only get bigger in reaction to changes in the model
`gtk.TREE_VIEW_COLUMN_AUTOSIZE`	Columns resize to be the optimal size every time the model changes.
`gtk.TREE_VIEW_COLUMN_FIXED`	Columns are a fixed numbers of pixels wide.

gtk.TreeViewColumn.get_sizing

```
    def get_sizing()
```

Returns :	the treeview column sizing type.

The `get_sizing`() method returns the value of the "sizing" property that contains the current type of the treeview column sizing mode. See the `set_sizing()` method for more detail.

gtk.TreeViewColumn.get_width

```
    def get_width()
```

Returns :	the current width of the treeview column.

The `get_width`() method returns the value of the "width" property that contains the current size of the treeview column in pixels.

gtk.TreeViewColumn.get_fixed_width

```
    def get_fixed_width()
```

Returns :	the fixed width of the treeview column

The `get_fixed_width`() method returns the value of the "fixed−width" property that contains the fixed width of the treeview column i.e. the width of the treeview column only if the sizing type is `gtk.TREE_VIEW_COLUMN_FIXED`.

gtk.TreeViewColumn.set_fixed_width

```
    def set_fixed_width(fixed_width)
```

fixed_width :	the size to set the treeview column to. Must be greater than 0.

The `set_fixed_width`() method sets the "fixed−width" property to the value of *`fixed_width`*. The value of *`fixed_width`* is the size of the column in pixels. This is meaningful only if the sizing type is `gtk.TREE_VIEW_COLUMN_FIXED`. The size of the column is clamped to the min and max width for the column. Please note that the min and max width of the column doesn't actually affect the "fixed_width" property of the widget, just the actual size when displayed.

gtk.TreeViewColumn.set_min_width

```
    def set_min_width(min_width)
```

`min_width` :	the minimum width of the treeview column in pixels, or −1.

The `set_min_width`() method sets the "min−width" property to the value of *`min_width`*. The value of *`min_width`* is the minimum width of the treeview column. If *`min_width`* is −1, then the minimum width is unset.

gtk.TreeViewColumn.get_min_width

```
    def get_min_width()
```

Returns :	the minimum width of the treeview column.

The `get_min_width`() method returns the value of the "min−width" property that contains the minimum width in pixels of the treeview column, or −1 if no minimum width is set.

gtk.TreeViewColumn.set_max_width

```
    def set_max_width(max_width)
```

`max_width` :	the maximum width of the column in pixels, or −1.

The `set_max_width`() method sets the "max−width" property to the value of *`max_width`*. The value of *`max_width`* becomes the maximum width of the treeview column. If *`max_width`* is −1, then the maximum width is unset. Note, the column can actually be wider than max width if it's the last column in a view. In this case, the column expands to fill any extra space.

gtk.TreeViewColumn.get_max_width

```
    def get_max_width()
```

Returns :	the maximum width of the treeview column.

The `get_max_width`() method returns the value of the "max−width" property that contains the maximum width in pixels of the treeview column, or −1 if no maximum width is set.

gtk.TreeViewColumn.clicked

```
    def clicked()
```

The `clicked`() method emits the "clicked" signal on the treeview column. The treeview column must be clickable.

gtk.TreeViewColumn.set_title

```
    def set_title(title)
```

`title` :	the title string of the treeview column.

The `set_title`() method sets the "title" property to the value of *`title`*. The "title" property contains the string that is used to set the treeview column title. If a custom widget has been set (see the `set_widget()` method), this value is ignored.

gtk.TreeViewColumn.get_title

```
    def get_title()
```

Returns :	the title of the column.

The `get_title()` method returns the value of the "title" property that contains the treeview column title.

gtk.TreeViewColumn.set_expand

```
    def set_expand(expand)
```

`expand` :	if `TRUE` the column expands to take extra space if available.

Note

This method is available in PyGTK 2.4 and above.

The `set_expand()` method sets the "expand" property to the value of *`expand`*. If expand is `TRUE` the column expands to take available extra space. This space is shared equally among all columns that have their "expand" property set to `TRUE`. If no column has this option set, then the last column gets all extra space. By default, every column is created with this `FALSE`.

gtk.TreeViewColumn.get_expand

```
    def get_expand()
```

Returns :	`TRUE`, if the column expands

Note

This method is available in PyGTK 2.4 and above.

The `get_expand()` method returns the value of the "expand" property. If "expand is `TRUE` if the column expands to take any available space.

gtk.TreeViewColumn.set_clickable

```
    def set_clickable(active)
```

`active` :	if `TRUE` the treeview column header can be clicked

The `set_clickable()` method sets the "clickable" property to the value of *`active`*. If *`active`* is `TRUE` the header can take keyboard focus, and be clicked.

gtk.TreeViewColumn.get_clickable

```
    def get_clickable()
```

Returns :	`TRUE` if the user can click the column header.

The `get_clickable()` method returns the value of the "clickable" property. If "clickable" is `TRUE` the user can click on the header for the treeview column.

gtk.TreeViewColumn.set_widget

```
    def set_widget(widget)
```

`widget` :	a child `gtk.Widget`.

The `set_widget()` method sets the widget in the header to be *`widget`*.

gtk.TreeViewColumn.get_widget

```
    def get_widget()
```

Returns :	the `gtk.Widget` in the column header, or `None`

The `get_widget()` method returns the `gtk.Widget` in the button on the column header. If a custom widget has not been set using the `set_widget()` method `None` is returned.

gtk.TreeViewColumn.set_alignment

```
    def set_alignment(xalign)
```

`xalign` :	the horizontal alignment, in the range 0.0 to 1.0 inclusive.

The `set_alignment()` method sets the "alignment" property to the value of *`xalign`*. *`xalign`* specifies the alignment of the title or custom widget inside the column header. The alignment value specifies the fraction of free space to the left of the widget.

gtk.TreeViewColumn.get_alignment

```
    def get_alignment()
```

Returns :	the current alignment of the treeview column.

The `get_alignment()` method returns the value of the "alignment" property that contains the current horizontal alignment of the treeview column. See the `set_alignment()` method for more detail.

gtk.TreeViewColumn.set_reorderable

```
    def set_reorderable(reorderable)
```

`reorderable` :	if `TRUE`, the column can be reordered.

The `set_reorderable()` method sets the "reorderable" property to the value of *`reorderable`*. If *`reorderable`* is `TRUE`, the column can be reordered by the end user dragging the header.

gtk.TreeViewColumn.get_reorderable

```
    def get_reorderable()
```

Returns :	`TRUE` if the treeview column can be reordered by the user.

The `get_reorderable()` method returns the value of the "reorderable" property. If "reorderable" is `TRUE` the treeview column can be reordered by the user.

gtk.TreeViewColumn.set_sort_column_id

```
    def set_sort_column_id(sort_column_id)
```

`sort_column_id` :	the logical column ID of the model to sort on or −1.

The `set_sort_column_id()` method is a convenience method that sets the column's sort column ID to the value specified by *`sort_column_id`* (an integer value). The treeview model sorts on the *`sort_column_id`* when this treeview column is selected for sorting. This method also makes the treeview column header clickable. If *`sort_column_id`* is −1 sorting using the treeview column is disabled.

This method sets up a number of callbacks that manage the sorting of the tree model when the column header is clicked. These callbacks provide toggling of the sort order, enabling the sort indicator and so on.

gtk.TreeViewColumn.get_sort_column_id

```
def get_sort_column_id()
```

Returns :	the current column ID for this column, or −1 if this column can't be used for sorting.

The `get_sort_column_id()` method returns the logical column ID that the model sorts on when this column is selected for sorting. See the `set_sort_column_id()` method.

gtk.TreeViewColumn.set_sort_indicator

```
def set_sort_indicator(setting)
```

`setting` :	if `TRUE` display an indicator that the column is sorted

The `set_sort_indicator()` method sets the "sort−indicator" property to the value of *`setting`*. If *`setting`* is `TRUE` an arrow is displayed in the header button when the column is sorted. Call the `set_sort_order()` to change the direction of the arrow.

Note

If the `set_sort_column_id()` convenience method has been called the visibility of the sort indicator will be managed automatically. See the `set_sort_order()` method for more information.

gtk.TreeViewColumn.get_sort_indicator

```
def get_sort_indicator()
```

Returns :	`TRUE` if the sort indicator arrow is displayed

The `get_sort_indicator()` method returns the value of the "sort−indicator" property. If "sort−indicator" is `TRUE` an arrow is displayed in the header button when the column is sorted.

gtk.TreeViewColumn.set_sort_order

```
def set_sort_order(order)
```

`order` :	the sort order that the sort indicator should indicate

The `set_sort_order()` method set the "sort−order" property to the value of order. The value of order must be either: `gtk.SORT_ASCENDING` or `gtk.SORT_DESCENDING`. This method changes the appearance of the sort indicator.

Note

This method *does not* actually sort the model. Use the `set_sort_column_id()` method if you want automatic sorting support. This method is primarily for custom sorting behavior, and should be used in conjunction with the set_sort_column_id() method to do that. For custom models, the mechanism will vary.

The sort indicator changes direction to indicate normal sort or reverse sort. Of course, you must have the sort indicator enabled to see anything when calling this method; see the set_sort_indicator() method.

gtk.TreeViewColumn.get_sort_order

```
def get_sort_order()
```

Returns :	the sort order the sort indicator is indicating

The `get_sort_order`() method returns the value of the "sort–order" property that indicates in which direction the treeview column is sorted. See the set_sort_order() method for more detail.

gtk.TreeViewColumn.cell_set_cell_data

```
def cell_set_cell_data(tree_model, iter, is_expander, is_expanded)
```

`tree_model` :	the `gtk.TreeModel` to get the cell renderer's attributes from.
`iter` :	the `gtk.TreeIter` to get the cell renderer's attributes from.
`is_expander` :	if `TRUE`, the row has children
`is_expanded` :	if `TRUE`, the row has visible children

The `cell_set_cell_data`() method sets the cell renderer attributes based on the specified *`tree_model`* and *`iter`*. That is, for every attribute mapping in the treeview column, it will get a value from the set column in the *`iter`*, and use that value to set the attribute on the cell renderer. If *`is_expander`* is `TRUE` the tree model row has children that may or may not be displayed. If *`is_expanded`* is `TRUE` the tree model row has children that are displayed.

gtk.TreeViewColumn.cell_get_size

```
def cell_get_size()
```

Returns :	a tuple containing five values: a `gtk.gdk.Rectangle` holding the area a the column will be allocated; the x offset of the cell; the y offset of the cell; the cell width; and, the cell height

The `cell_get_size`() method returns a tuple containing:

- a `gtk.gdk.Rectangle` holding the area a cell in the treeview column will be allocated.
- the x offset of the cell relative to *`cell_area`*.
- the y offset of the cell relative to *`cell_area`*.
- the width of the cell.
- the height of the cell.

This method is used primarily by the `gtk.TreeView`.

gtk.TreeViewColumn.cell_is_visible

```
def cell_is_visible()
```

Returns :	`TRUE`, if any of the cells packed into the treeview column are currently visible

The `cell_is_visible`() method returns `TRUE` if any of the cells packed into the treeview column are visible. For this to be meaningful, you must first initialize the cells with the cell_set_cell_data() method.

gtk.TreeViewColumn.focus_cell

```
    def focus_cell(cell)
```

`cell` :	a `gtk.CellRenderer`

Note

This method is available in PyGTK 2.2 and above.

The `focus_cell`() method sets the current keyboard focus to be at *`cell`*, if the column contains 2 or more editable and activatable cells.

gtk.TreeViewColumn.cell_get_position

```
    def cell_get_position(cell_renderer)
```

`cell_renderer` :	a `gtk.CellRenderer`
Returns :	a 2–tuple containing the horizontal position and size of a cell or `None`

Note

This method is available in PyGTK 2.4 and above.

The `cell_get_position`() method returns the horizontal position and size of the cell specified by *`cell_renderer`*. If the cell is not found in the column, `None` is returned.

Signals

The "clicked" gtk.TreeViewColumn Signal

```
    def callback(treeviewcolumn, user_param1, ...)
```

`treeviewcolumn` :	the treeviewcolumn that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "clicked" signal is emitted when the user clicks on the *`treeviewcolumn`* header button.

Prev Up Next
gtk.TreeView Home gtk.UIManager

gtk.UIManager
Prev **The gtk Class Reference** Next

gtk.UIManager

gtk.UIManager construct menus and toolbars from an XML description (new in PyGTK 2.4)

Synopsis

```
class gtk.UIManager(gobject.GObject):
    gtk.UIManager()
    def set_add_tearoffs(add_tearoffs)
    def get_add_tearoffs()
    def insert_action_group(action_group, pos)
    def remove_action_group(action_group)
    def get_action_groups()
    def get_accel_group()
    def get_widget(path)
    def get_toplevels(types)
    def get_action(path)
    def add_ui_from_string(buffer)
    def add_ui_from_file(filename)
    def add_ui(merge_id, path, name, action, type, top)
    def remove_ui(merge_id)
    def get_ui()
    def ensure_update()
    def new_merge_id()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.UIManager
```

Properties

"add–tearoffs"	Read–Write	If `TRUE`, regular menus have tearoff menu items. Note that this only affects regular menus. Generated popup menus never have tearoff menu items. Available in `GTK+` 2.4 and above.
"ui"	Read–Write	An XML string describing the merged UI.

Signal Prototypes

"actions–changed"	`def callback(`*`uimanager, user_param1, ...`*`)`
"add–widget"	`def callback(`*`uimanager, widget, user_param1, ...`*`)`
"connect–proxy"	`def callback(`*`uimanager, action, widget, user_param1, ...`*`)`
"disconnect–proxy"	`def callback(`*`uimanager, action, widget, user_param1, ...`*`)`
"post–activate"	`def callback(`*`uimanager, action, user_param1, ...`*`)`
"pre–activate"	`def callback(`*`uimanager, action, user_param1, ...`*`)`

Description

Note

This object is available in PyGTK 2.4 and above.

A `gtk.UIManager` constructs a user interface (menus and toolbars) from one or more UI definitions, which reference actions from one or more action groups.

UI Definitions

The UI definitions are specified in an XML format which can be roughly described by the following DTD.

```
<!ELEMENT ui          (menubar|toolbar|popup|accelerator)* >
```

```
<!ELEMENT menubar     (menuitem|separator|placeholder|menu)* >
<!ELEMENT menu        (menuitem|separator|placeholder|menu)* >
<!ELEMENT popup       (menuitem|separator|placeholder|menu)* >
<!ELEMENT toolbar     (toolitem|separator|placeholder)* >
<!ELEMENT placeholder (menuitem|toolitem|separator|placeholder|menu)* >
<!ELEMENT menuitem     EMPTY >
<!ELEMENT toolitem     EMPTY >
<!ELEMENT separator    EMPTY >
<!ELEMENT accelerator  EMPTY >
<!ATTLIST menubar      name                  #IMPLIED >
<!ATTLIST toolbar      name                  #IMPLIED >
<!ATTLIST popup        name                  #IMPLIED >
<!ATTLIST placeholder  name                  #IMPLIED >
<!ATTLIST menu         name                  #IMPLIED
                       action                #REQUIRED
                       position (top|bot)    #IMPLIED >
<!ATTLIST menuitem     name                  #IMPLIED
                       action                #REQUIRED
                       position (top|bot)    #IMPLIED >
<!ATTLIST toolitem     name                  #IMPLIED
                       action                #REQUIRED
                       position (top|bot)    #IMPLIED >
<!ATTLIST accelerator  name                  #IMPLIED
                       action                #REQUIRED >
```

There are some additional restrictions beyond those specified in the DTD, e.g. every toolitem must have a toolbar in its ancestry and every menuitem must have a menubar or popup in its ancestry. Since a GMarkup parser is used to parse the UI description, it must not only be valid XML, but valid GMarkup. If a name is not specified, it defaults to the action. If an action is not specified either, the element name is used.

Example 1. A UI definition

```
<ui>
  <menubar>
    <menu name="FileMenu" action="FileMenuAction">
      <menuitem name="New" action="New2Action" />
      <placeholder name="FileMenuAdditions" />
    </menu>
    <menu name="JustifyMenu" action="JustifyMenuAction">
      <menuitem name="Left" action="justify-left"/>
      <menuitem name="Centre" action="justify-center"/>
      <menuitem name="Right" action="justify-right"/>
      <menuitem name="Fill" action="justify-fill"/>
    </menu>
  </menubar>
  <toolbar action="toolbar1">
    <placeholder name="JustifyToolItems">
      <separator/>
      <toolitem name="Left" action="justify-left"/>
      <toolitem name="Centre" action="justify-center"/>
      <toolitem name="Right" action="justify-right"/>
      <toolitem name="Fill" action="justify-fill"/>
      <separator/>
    </placeholder>
  </toolbar>
</ui>
```

The constructed widget hierarchy is very similar to the element tree of the XML, with the exception that placeholders are merged into their parents. The correspondence of XML elements to widgets should be almost obvious:

menubar	a `gtk.MenuBar`
toolbar	a `gtk.Toolbar`

popup	a toplevel `gtk.Menu`
menu	a `gtk.Menu` attached to a menuitem
menuitem	a `gtk.MenuItem` subclass, the exact type depends on the action
toolitem	a `gtk.ToolItem` subclass, the exact type depends on the action
separator	a `gtk.SeparatorMenuItem` or `gtk.SeparatorToolItem`
accelerator	a keyboard accelerator

The "position" attribute determines where a constructed widget is positioned with respect to its siblings in the partially constructed tree. If it is "top", the widget is prepended, otherwise it is appended.

UI Merging

The most remarkable feature of `gtk.UIManager` is that it can overlay a set of menuitems and toolitems over another one, and demerge them later.

Merging is done based on the name of the XML elements. Each element is identified by a path which consists of the names of its ancestors, separated by slashes. For example, the menuitem named "Left" in the example above has the path `/ui/menubar/JustifyMenu/Left` and the toolitem with the same name has path `/ui/toolbar1/JustifyToolItems/Left`.

Accelerators

Every action has an accelerator path. Accelerators are installed together with menuitem proxies, but they can also be explicitly added with <accelerator> elements in the UI definition. This makes it possible to have accelerators for actions even if they have no visible proxies.

Smart Separators

The separators created by `gtk.UIManager` are "smart", i.e. they do not show up in the UI unless they end up between two visible menu or tool items. Separators which are located at the very beginning or end of the menu or toolbar containing them, or multiple separators next to each other, are hidden. This is a useful feature, since the merging of UI elements from multiple sources can make it hard or impossible to determine in advance whether a separator will end up in such an unfortunate position.

Empty Menus

Submenus pose similar problems to separators in connection with merging. It is impossible to know in advance whether they will end up empty after merging. `gtk.UIManager` offers two ways to treat empty submenus:

- make them disappear by hiding the menu item they're attached to
- add an insensitive "Empty" item

The behavior is chosen based on the "is_important" property of the action to which the submenu is associated.

Constructor

```
gtk.UIManager()
```

Returns :	a new ui manager object.

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new `gtk.UIManager` object.

Methods

gtk.UIManager.set_add_tearoffs

```
def set_add_tearoffs(add_tearoffs)
```

`add_tearoffs` :	`TRUE` if tearoff menu items are added to regular menus

Note

This method is available in PyGTK 2.4 and above.

The `add_tearoffs()` method sets the "add_tearoffs" property to the value of *`add_tearoffs`*. If *`add_tearoffs`* is `TRUE` regular menus generated by this `gtk.UIManager` will have tearoff menu items.

Note that this only affects regular menus. Generated popup menus never have tearoff menu items.

gtk.UIManager.get_add_tearoffs

```
def get_add_tearoffs()
```

Returns :	`TRUE` if tearoff menu items are added

Note

This method is available in PyGTK 2.4 and above.

The `get_add_tearoffs()` method returns the value of the "add–tearoffs" property. If "add–tearoffs" is `TRUE` regular menus generated will have tearoff menu items.

gtk.UIManager.insert_action_group

```
def insert_action_group(action_group, pos)
```

`action_group` :	the action group to be inserted
`pos` :	the position at which the group will be inserted. If *`pos`* is negative *`action_group`* is inserted at the end of the list.

Note

This method is available in PyGTK 2.4 and above.

The `insert_action_group()` method inserts the `gtk.ActionGroup` specified by *`action_group`* into the list of associated action groups at the position specified by *`pos`*. Actions in earlier groups hide actions with the same name in later groups.

gtk.UIManager.remove_action_group

```
    def remove_action_group(action_group)
```

`action_group` :	the action group to be removed

Note

This method is available in PyGTK 2.4 and above.

The `remove_action_group()` method removes the `gtk.ActionGroup` specified by *`action_group`* from the list of associated action groups.

gtk.UIManager.get_action_groups

```
    def get_action_groups()
```

Returns :	a list of associated action groups.

Note

This method is available in PyGTK 2.4 and above.

The `get_action_groups()` method returns the list of associated `gtk.ActionGroup` objects.

gtk.UIManager.get_accel_group

```
    def get_accel_group()
```

Returns :	the `gtk.AccelGroup`.

Note

This method is available in PyGTK 2.4 and above.

The `get_accel_group()` method returns the associated `gtk.AccelGroup`.

gtk.UIManager.get_widget

```
    def get_widget(path)
```

`path` :	a path
Returns :	the widget found by following the path, or `None` if no widget was found.

Note

This method is available in PyGTK 2.4 and above.

The `get_widget()` method looks up a widget by following the path specified by *`path`*. The path consists of the names specified in the XML description of the UI. separated by '/'. Elements that don't have a name or action attribute in the XML (e.g. <popup>) can be addressed by their XML element name (e.g. "popup"). The root element ("/ui") can be omitted in the path.

gtk.UIManager.get_toplevels

```
def get_toplevels(types)
```

types :	specifies the types of toplevel widgets to include. Allowed types are `gtk.UI_MANAGER_MENUBAR`, `gtk.UI_MANAGER_TOOLBAR` and `gtk.UI_MANAGER_POPUP`.
Returns :	a list of all toplevel widgets of the requested types.

Note

This method is available in PyGTK 2.4 and above.

The `get_toplevels()` method returns a list of all toplevel widgets of the types specified by *types*.

gtk.UIManager.get_action

```
def get_action(path)
```

path :	a path
Returns :	the action whose proxy widget is found by following the path, or `None` if no widget was found.

Note

This method is available in PyGTK 2.4 and above.

The `get_action()` method looks up a `gtk.Action` by following a path. See the `get_widget()` method for more information about paths.

gtk.UIManager.add_ui_from_string

```
def add_ui_from_string(buffer)
```

buffer :	the string to parse
Returns :	The merge id for the merged UI. The merge id can be used to unmerge the UI with the `remove_ui()` method.

Note

This method is available in PyGTK 2.4 and above.

The `add_ui_from_string()` method parses the string specified by *buffer* that contains a UI definition and merges it with the current contents of the ui manager. An enclosing <ui> element is added if it is missing.

This method raise the GError exception if an error occurs during the parsing of the string.

gtk.UIManager.add_ui_from_file

```
def add_ui_from_file(filename)
```

filename :	the name of the file to parse
Returns :	The merge id for the merged UI. The merge id can be used to unmerge the UI with the `gtk.UIManager.remove_ui()` method.

Note

This method is available in PyGTK 2.4 and above.

The `add_ui_from_file()` method parses the file specified by *`filename`* that contains a UI definition and merges it with the current contents of the ui manager.

This method raise the GError exception if an error occurs during the parsing of the file.

gtk.UIManager.add_ui

```
    def add_ui(merge_id, path, name, action, type, top)
```

`merge_id` :	the merge id for the merged UI, see `new_merge_id()`
`path` :	a path where the element should be added
`name` :	the name for the added UI element
`action` :	the name of the action to be proxied, or `None` to add a separator
`type` :	the type of UI element to add.
`top` :	if `TRUE`, the UI element is added before its siblings, otherwise it is added after its siblings.

Note

This method is available in PyGTK 2.4 and above.

The `add_ui()` method adds a UI element of the type specified by *`type`* to the current contents of the ui manager at the location specified by *`path`*. Note that path must not start with "/ui" though "ui" is acceptable. For example "/menubar" or "ui/menubar" is acceptable but "/ui/menubar" is not. *`type`* must be one of:

`gtk.UI_MANAGER_AUTO`	The type of the UI element (menuitem, toolitem or separator) is set according to the context.
`gtk.UI_MANAGER_MENUBAR`	A menubar.
`gtk.UI_MANAGER_MENU`	A menu.
`gtk.UI_MANAGER_TOOLBAR`	A toolbar.
`gtk.UI_MANAGER_PLACEHOLDER`	A placeholder.
`gtk.UI_MANAGER_POPUP`	A popup menu.
`gtk.UI_MANAGER_MENUITEM`	A menuitem.
`gtk.UI_MANAGER_TOOLITEM`	A toolitem.
`gtk.UI_MANAGER_SEPARATOR`	A separator.
`gtk.UI_MANAGER_ACCELERATOR`	An accelerator.

gtk.UIManager.remove_ui

```
    def remove_ui(merge_id)
```

`merge_id` :	a merge id

Note

This method is available in PyGTK 2.4 and above.

The `remove_ui()` method unmerges the part of the ui manager content identified by *`merge_id`*.

gtk.UIManager.get_ui

```
def get_ui()
```

Returns :	A string containing an XML representation of the merged UI.

Note

This method is available in PyGTK 2.4 and above.

The `get_ui`() method creates a UI definition of the merged UI.

gtk.UIManager.ensure_update

```
def ensure_update()
```

Note

This method is available in PyGTK 2.4 and above.

The `ensure_update`() method makes sure that all pending updates to the UI have been completed. This may occasionally be necessary, since `gtk.UIManager` updates the UI in an idle function. A typical example where this method is useful is to enforce that the menubar and toolbar have been added to the main window before showing it:

```
window.add(vbox)
merge.connect("add_widget", add_widget, vbox)
merge.add_ui_from_file("my-menus")
merge.add_ui_from_file("my-toolbars")
merge.ensure_update()
window.show()
```

gtk.UIManager.new_merge_id

```
def new_merge_id()
```

Returns :	an unused merge id.

Note

This method is available in PyGTK 2.4 and above.

The `new_merge_id`() method returns an unused merge id, suitable for use with the `add_ui()` method. The returned merge ids are monotonically increasing integer values.

Signals

The "actions-changed" gtk.UIManager Signal

```
def callback(uimanager, user_param1, ...)
```

`uimanager` :	the uimanager that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.4 and above.

The "actions−changed" signal is emitted when the set of actions changes.

The "add−widget" gtk.UIManager Signal

```
def callback(uimanager, widget, user_param1, ...)
```

uimanager :	the uimanager that received the signal
widget :	the added widget
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.4 and above.

The "add−widget" signal is emitted for each generated menubar and toolbar. The added widget is specified by *widget*. It is not emitted for generated popup menus, which can be retrieved by the `get_widget()` method.

The "connect−proxy" gtk.UIManager Signal

```
def callback(uimanager, action, widget, user_param1, ...)
```

uimanager :	the uimanager that received the signal
action :	the action
widget :	the proxy widget
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.4 and above.

The "connect−proxy" signal is emitted after connecting the proxy widget specified by *widget* to the `gtk.Action` specified by *action* in the group. This is intended for simple customizations for which a custom action class would be too clumsy, e.g. showing tooltips for menuitems in the statusbar.

The "disconnect−proxy" gtk.UIManager Signal

```
def callback(uimanager, action, widget, user_param1, ...)
```

uimanager :	the uimanager that received the signal
action :	the action
widget :	the proxy widget
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.4 and above.

The "disconnect−proxy" signal is emitted when `widget` is disconnected from `action`.

The "post−activate" gtk.UIManager Signal

```
    def callback(uimanager, action, user_param1, ...)
```

uimanager :	the uimanager that received the signal
action :	the action
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.4 and above.

The "post−activate" signal is emitted after `action` is activated. This signal is intended for applications to get notification after any action is activated.

The "pre−activate" gtk.UIManager Signal

```
    def callback(uimanager, action, user_param1, ...)
```

uimanager :	the uimanager that received the signal
action :	the action
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.4 and above.

The "pre−activate" signal is emitted before `action` is activated. This signal is intended for applications to get notification before any action is activated.

Prev Up Next
gtk.TreeViewColumn Home gtk.VBox

gtk.VBox
Prev **The gtk Class Reference** Next

gtk.VBox

gtk.VBox — a vertical container box

Synopsis

```
class gtk.VBox(gtk.Box):
    gtk.VBox(homogeneous=FALSE, spacing=0)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Box
          +-- gtk.VBox
```

Description

A `gtk.VBox` is a container subclassed from `gtk.Box` that organizes its child widgets into a single column. The `gtk.Box` methods are used to manage the arrangement, spacing, height, and alignment of the `gtk.VBox` child widgets though all are allocated the same width.

Constructor

```
    gtk.VBox(homogeneous=FALSE, spacing=0)
```

homogeneous :	if TRUE all child widgets are given equal space allocations.
spacing :	the additional vertical space between child widgets in pixels.
Returns :	a new `gtk.VBox` widget

Creates a new `gtk.VBox` widget with the spacing between child widgets specified by *spacing*. If *homogeneous* is TRUE all child widgets are allocated the same space.

gtk.VButtonBox

gtk.VButtonBox — a container for arranging buttons vertically.

Synopsis

```
class gtk.VButtonBox(gtk.ButtonBox):
    gtk.VButtonBox()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Box
          +-- gtk.ButtonBox
            +-- gtk.VButtonBox
```

Description

A gtk.VButtonBox is a container subclassed from gtk.ButtonBox that is optimized for the vertical layout of buttons. The gtk.VButtonBox helps provide a consistent layout of buttons in an application by supplying default values of spacing, padding and layout style (see the gtk.ButtonBox reference for more detail). Buttons are packed into a gtk.VButtonBox using the gtk.Container.add()) method. The pack_start() and pack_end() methods can also be used but they work just like the gtk.Container.add()) method i.e. they pack the button in a way that depends on the current layout style and on whether the button has had the gtk.ButtonBox.set_child_secondary() method called on it. The spacing between buttons can be set with the gtk.Box.set_spacing() method. The arrangement and layout of the buttons can be changed with the gtk.ButtonBox.set_layout() method.

Constructor

```
    gtk.VButtonBox()
```

Returns :	a new gtk.VButtonBox widget

Creates a new gtk.VButtonBox widget

gtk.Viewport

gtk.Viewport a widget displaying a portion of a larger widget.

Synopsis

```
class gtk.Viewport(gtk.Bin):
    gtk.Viewport(hadjustment=None, vadjustment=None)
    def get_hadjustment()
    def get_vadjustment()
    def set_hadjustment(adjustment)
    def set_vadjustment(adjustment)
    def set_shadow_type(type)
    def get_shadow_type()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Viewport
```

Properties

"hadjustment"	Read–Write	The gtk.Adjustment that determines the values of the horizontal position for this viewport.
"vadjustment"	Read–Write	The gtk.Adjustment that determines the values of the vertical position for this viewport.
"shadow–type"	Read–Write	The type of shadowed box around the viewport: `gtk.SHADOW_NONE`, `gtk.SHADOW_IN`, `gtk.SHADOW_OUT`, `gtk.SHADOW_ETCHED_IN` or `gtk.SHADOW_ETCHED_OUT`

Signal Prototypes

"set–scroll–adjustments"	`def callback(`*`viewport, hadjustment, vadjustment, user_param1, ...`*`)`

Description

A `gtk.Viewport` widget provides a view into a portion of a larger widget. A `gtk.Viewport` is usually used to provide adjustment capability for a widget to be used in a `gtk.ScrolledWindow`. For example, a label widget doesn't have any adjustments (does not support the "set–scroll–adjustments" signal) and would need a viewport when added to a `gtk.ScrolledWindow` to make the scrollbars functional. See the `gtk.ScrolledWindow.add_with_viewport()` method for more information.

Constructor

```
    gtk.Viewport(hadjustment=None, vadjustment=None)
```

`hadjustment` :	a horizontal adjustment.
`vadjustment` :	a vertical adjustment.
Returns :	a new `gtk.Viewport` widget

Creates a new `gtk.Viewport` with the horizontal and vertical adjustments specified by *`hadjustment`* and *`vadjustment`* respectively. If *`hadjustment`* or *`vadjustment`* is `None` a new adjustment will be created.

Methods

gtk.Viewport.get_hadjustment

```
    def get_hadjustment()
```

Returns :	the horizontal adjustment

The `get_hadjustment()` method returns the value of the "hadjustment" property that contains the horizontal `gtk.Adjustment` of the viewport.

gtk.Viewport.get_vadjustment

```
    def get_vadjustment()
```

Returns :	the vertical adjustment

The `get_vadjustment()` method returns the value of the "vadjustment" property that contains the vertical `gtk.Adjustment` of the viewport.

gtk.Viewport.set_hadjustment

```
def set_hadjustment(adjustment)
```

`adjustment` :	a `gtk.Adjustment`.

The `set_hadjustment()` method sets the "hadjustment" property to the value of *`adjustment`* that becomes the horizontal adjustment of the viewport.

gtk.Viewport.set_vadjustment

```
def set_vadjustment(adjustment)
```

`adjustment` :	a `gtk.Adjustment`.

The `set_vadjustment()` method sets the "vadjustment" property to the value of *`adjustment`* that becomes the vertical adjustment of the viewport.

gtk.Viewport.set_shadow_type

```
def set_shadow_type(type)
```

`type` :	the new shadow type.

The `set_shadow_type()` method sets the "shadow−type" property to the value of *`type`*. The value of *`type`* must be one of:

- `gtk.SHADOW_NONE`
- `gtk.SHADOW_IN`
- `gtk.SHADOW_OUT`
- `gtk.SHADOW_ETCHED_IN`
- `gtk.SHADOW_ETCHED_OUT`

gtk.Viewport.get_shadow_type

```
def get_shadow_type()
```

Returns :	the shadow type

The `get_shadow_type()` method returns the value of the "shadow−type" property that contains the shadow type of the `gtk.Viewport`. See the `set_shadow_type()` method for more information.

Signals

The "set−scroll−adjustments" gtk.Viewport Signal

```
def callback(viewport, hadjustment, vadjustment, user_param1, ...)
```

`viewport` :	the viewport that received the signal
`hadjustment` :	the horizontal `gtk.Adjustment`
`vadjustment` :	the vertical `gtk.Adjustment`
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "set−scroll−adjustments" signal is emitted when one or both of the horizontal and vertical `gtk.Adjustment` objects is changed.

gtk.VPaned

gtk.VPaned — A container with two panes arranged vertically.

Synopsis

```
class gtk.VPaned(gtk.Paned):
    gtk.VPaned()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Paned
          +-- gtk.VPaned
```

Description

The `gtk.VPaned` widget is a container widget subclassed from `gtk.Paned` with two child widgets arranged vertically. The division between the two child widgets can be adjusted by the user by dragging a handle. See the `gtk.Paned` description for more details.

Constructor

```
   gtk.VPaned()
```

Returns :	a new `gtk.VPaned` widget

Creates a new `gtk.HPaned` widget.

gtk.VRuler

gtk.VRuler — a vertical ruler.

Synopsis

```
class gtk.VRuler(gtk.Ruler):
    gtk.VRuler()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Ruler
        +-- gtk.VRuler
```

Description

Note

This widget is considered too specialized or little–used for PyGTK, and will in the future be moved to some other package. If your application needs this widget, feel free to use it, as the widget does work and is useful in some applications; it's just not of general interest. However, it will eventually move out of the PyGTK distribution.

The `gtk.VRuler` widget is arranged vertically creating a ruler that is used in conjunction with other widgets such as a text widget. The ruler is used to show the location of the mouse on the window and to show the size of the window in specified units. The available units of measurement are `gtk.PIXELS` (the default), `gtk.INCHES` and `gtk.CENTIMETERS`. See the `gtk.Ruler` description for more information on the methods that are used to manage a `gtk.VRuler`.

Constructor

```
  gtk.VRuler()
```

Returns :	a new `gtk.VRuler` widget

Creates a new `gtk.VRuler` widget.

gtk.VScale

gtk.VScale a vertical slider widget used to select a value from a range.

Synopsis

```
class gtk.VScale(gtk.Scale):
    gtk.VScale(adjustment=None)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Range
        +-- gtk.Scale
          +-- gtk.VScale
```

Description

The `gtk.VScale` is subclassed from `gtk.Scale` to provide a widget that allows a user to select a value using a horizontal slider. See the `gtk.Scale` description for more information on the methods available to manage a `gtk.VScale`.

Constructor

```
    gtk.VScale(adjustment=None)
```

`adjustment` :	a `gtk.Adjustment` or `None`
Returns :	a new `gtk.VScale` widget

Creates a new `gtk.VScale` widget and associates a `gtk.Adjustment` specified by *`adjustment`* with it. The default value of *`adjustment`* is `None` which creates the vscale with no `gtk.Adjustment`.

gtk.VScrollbar

gtk.VScrollbar — a vertical scrollbar

Synopsis

```
class gtk.VScrollbar(gtk.Scrollbar):
    gtk.VScrollbar(adjustment=None)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Range
        +-- gtk.Scrollbar
          +-- gtk.VScrollbar
```

Description

The `gtk.VScrollbar` widget is subclassed from `gtk.Scrollbar` to provide a horizontal scrollbar. See the `gtk.Scrollbar` reference for details on the methods available for managing vertical scrollbars. A `gtk.Adjustment` may be specified for the scrollbar at creation (or is created automatically if `None` is specified) to handle the adjustment of the scrollbar. See the `gtk.Adjustment` method for more details.

Constructor

```
    gtk.VScrollbar(adjustment=None)
```

adjustment :	a `gtk.Adjustment` object or `None`
Returns :	a new `gtk.VScrollbar` widget

Creates a new `gtk.VScrollbar` with an associated `gtk.Adjustment` specified by *adjustment*. If *adjustment* is `None` or missing a new `gtk.Adjustment` object will be created and associated with the scrollbar.

Prev	Up	Next
gtk.VScale	Home	gtk.VSeparator

gtk.VSeparator

Prev **The gtk Class Reference** Next

gtk.VSeparator

gtk.VSeparator a vertical separator.

Synopsis

```
class gtk.VSeparator(gtk.Separator):
    gtk.VSeparator()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Separator
        +-- gtk.VSeparator
```

Description

The `gtk.VSeparator` widget is a vertical separator, used to group the widgets within a window. It displays a vertical line with a shadow to make it appear sunken into the interface.

Constructor

```
    gtk.VSeparator()
```

Returns :	a new `gtk.VSeparator` widget

Creates a new `gtk.VSeparator` widget.

Prev Up Next

gtk.Widget

gtk.Widget the base class for all PyGTK widgets

Synopsis

```
class gtk.Widget(gtk.Object):
    def get_allocation()
    def drag_check_threshold(start_x, start_y, current_x, current_y)
    def drag_get_data(context, target, time=0L)
    def drag_highlight()
    def drag_unhighlight()
    def drag_dest_set(flags, targets, actions)
    def drag_dest_set_proxy(proxy_window, protocol, use_coordinates)
    def drag_dest_unset()
    def drag_dest_find_target(context, target_list)
    def drag_dest_get_target_list()
    def drag_dest_set_target_list(target_list)
    def drag_dest_add_image_targets()
    def drag_dest_add_text_targets()
    def drag_dest_add_uri_targets()
    def drag_source_set(start_button_mask, targets, actions)
    def drag_source_unset()
    def drag_source_set_icon(colormap, pixmap, mask)
    def drag_source_set_icon_pixbuf(pixbuf)
    def drag_source_set_icon_stock(stock_id)
    def drag_source_add_text_targets()
    def drag_begin(targets, actions, button, event)
    def grab_add()
    def grab_remove()
    def rc_get_style()
    def selection_owner_set(selection, time=0L)
    def selection_add_target(selection, target, info)
    def selection_add_targets(selection, targets)
    def selection_clear_targets(selection)
    def selection_convert(selection, target, time=0L)
    def selection_remove_all()
    def destroy()
    def unparent()
    def show()
    def show_now()
    def hide()
    def show_all()
    def hide_all()
    def set_no_show_all(no_show_all)
    def get_no_show_all()
    def map()
    def unmap()
    def realize()
    def unrealize()
    def queue_draw()
    def queue_draw_area(x, y, width, height)
    def queue_resize()
    def queue_resize_no_redraw()
    def size_request()
    def size_allocate(allocation)
    def get_child_requisition(requisition)
    def add_accelerator(accel_signal, accel_group, accel_key, accel_mods, accel_flags)
    def remove_accelerator(accel_group, accel_key, accel_mods)
    def set_accel_path(accel_path, accel_group)
    def can_activate_accel(signal_id)
    def mnemonic_activate(group_cycling)
    def event(event)
    def send_expose(event)
    def activate()
    def set_scroll_adjustments(hadjustment, vadjustment)
    def reparent(new_parent)
    def intersect(area)
    def freeze_child_notify()
    def child_notify(child_property)
    def thaw_child_notify()
    def is_focus()
    def grab_focus()
    def grab_default()
    def set_name(name)
    def get_name()
    def set_state(state)
    def set_sensitive(sensitive)
    def set_app_paintable(app_paintable)
    def set_double_buffered(double_buffered)
    def set_redraw_on_allocate(redraw_on_allocate)
    def set_parent(parent)
    def set_parent_window(parent_window)
    def set_child_visible(is_visible)
    def get_child_visible()
    def get_parent()
    def get_parent_window()
    def child_focus(direction)
    def set_size_request(width, height)
    def get_size_request()
    def set_events(events)
    def add_events(events)
    def set_extension_events(mode)
    def get_extension_events()
    def get_toplevel()
    def get_ancestor(widget_type)
    def get_colormap()
    def get_visual()
    def get_screen()
    def has_screen()
    def get_display()
    def get_root_window()
    def get_settings()
    def get_clipboard(selection)
    def set_colormap(colormap)
    def get_events()
    def get_pointer()
    def is_ancestor(ancestor)
    def translate_coordinates(dest_widget, src_x, src_y)
    def hide_on_delete()
    def set_style(style)
    def ensure_style()
    def get_style()
    def modify_style(style)
    def get_modifier_style()
    def modify_fg(state, color)
    def modify_bg(state, color)
    def modify_text(state, color)
    def modify_base(state, color)
    def modify_font(font_desc)
    def create_pango_context()
    def get_pango_context()
    def create_pango_layout(text)
    def render_icon(stock_id, size, detail)
    def set_composite_name(name)
    def get_composite_name()
    def reset_rc_styles()
    def style_get_property(property_name)
    def set_direction(dir)
    def get_direction()
    def shape_combine_mask(shape_mask, offset_x, offset_y)
    def reset_shapes()
    def path()
    def class_path()
    def list_mnemonic_labels()
    def add_mnemonic_label(label)
    def remove_mnemonic_label(label)
    def menu_get_for_attach_widget()
```

```
Functions

    def gtk.widget_push_colormap(cmap)
    def gtk.widget_push_composite_child()
    def gtk.widget_pop_composite_child()
    def gtk.widget_pop_colormap()
    def gtk.widget_get_default_style()
    def gtk.widget_set_default_colormap(colormap)
    def gtk.widget_get_default_colormap()
    def gtk.widget_get_default_visual()
    def gtk.widget_set_default_direction(dir)
    def gtk.widget_get_default_direction()
    def gtk.widget_list_style_properties(cmap)
    def gtk.widget_class_install_style_property(widget, pspec)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
```

Properties

"app–paintable"	Read–Write	If `TRUE`, the application will paint directly on the widget
"can–default"	Read–Write	If `TRUE`, the widget can be the default widget
"can–focus"	Read–Write	If `TRUE`, the widget can accept the input focus
"composite–child"	Read–Write	If `TRUE`, the widget is part of a composite widget
"events"	Read–Write	The event mask that decides what kind of `gtk.gdk.Event` this widget gets.
"extension–events"	Read–Write	The mask that decides what kind of extension events this widget gets.
"has–default"	Read–Write	If `TRUE`, the widget is the default widget
"has–focus"	Read–Write	If `TRUE`, the widget has the input focus
"height–request"	Read–Write	The height request of the widget, or −1 if natural request should be used.
"is–focus"	Read–Write	If `TRUE`, the widget is the focus widget within the toplevel
"name"	Read–Write	The name of the widget
"no–show–all"	Read–Write	If `TRUE``show_all()` should not affect this widget. Available in GTK+ 2.4 and above.
"parent"	Read–Write	The parent widget of this widget. Must be a `gtk.Container` widget.

"receives–default"	Read–Write	If `TRUE`, the widget will receive the default action when it is focused.
"sensitive"	Read–Write	If `TRUE`, the widget responds to input
"style"	Read–Write	The style of the widget, which contains information about how it will look (colors etc).
"visible"	Read–Write	If `TRUE`, the widget is visible
"width–request"	Read–Write	The width request of the widget, or –1 if natural request should be used.

Style Properties

"cursor–aspect–ratio"	Read	The aspect ratio with which to draw the insertion cursor
"cursor–color"	Read	The `gtk.gdk.Color` with which to draw insertion cursor
"focus–line–pattern"	Read–Write	The dash pattern used to draw the focus indicator.
"focus–line–width"	Read–Write	The width, in pixels, of the focus indicator line.
"focus–padding"	Read–Write	The width, in pixels, between the focus indicator and the widget 'box'.
"interior–focus"	Read	If `TRUE`, draw the focus indicator inside widgets.
"secondary–cursor–color"	Read	The `gtk.gdk.Color` with which to draw the secondary insertion cursor when editing mixed right–to–left and left–to–right text.

Attributes

"allocation"	Read–Write	The `gtk.gdk.Rectangle` specifying the widget's space allocation. This attribute is writeable in PyGTK 2.4.
"name"	Read	The name of the widget
"parent"	Read	The parent widget of this widget. Must be a `gtk.Container` widget.
"saved_state"	Read	The widget's saved state.
"state"	Read	The widget state: `gtk.STATE_NORMAL`, `gtk.STATE_ACTIVE`, `gtk.STATE_PRELIGHT`, `gtk.STATE_SELECTED` or `gtk.STATE_INSENSITIVE`
"style"	Read	The style of the widget.
"window"	Read–Write	The `gtk.gdk.Window` used by the widget. This attribute is writeable in PyGTK 2.4.

Signal Prototypes

"accel–closures–changed"	`def callback(`*`widget`*`, `*`user_param1`*`, ...)`
"button–press–event"	`def callback(`*`widget`*`, `*`event`*`, `*`user_param1`*`, ...)`
"button–release–event"	`def callback(`*`widget`*`, `*`signal_id`*`, `*`user_param1`*`, ...)`
"can–activate–accel"	`def callback(`*`widget`*`, `*`event`*`, `*`user_param1`*`, ...)`
"child–notify"	`def callback(`*`widget`*`, `*`child_property`*`, `*`user_param1`*`, ...)`
"client–event"	`def callback(`*`widget`*`, `*`event`*`, `*`user_param1`*`, ...)`

"configure−event"	def callback(*widget*, *event*, *user_param1*, *...*)
"delete−event"	def callback(*widget*, *event*, *user_param1*, *...*)
"destroy−event"	def callback(*widget*, *event*, *user_param1*, *...*)
"direction−changed"	def callback(*widget*, *direction*, *user_param1*, *...*)
"drag−begin"	def callback(*widget*, *drag_context*, *user_param1*, *...*)
"drag−data−delete"	def callback(*widget*, *drag_context*, *user_param1*, *...*)
"drag−data−get"	def callback(*widget*, *drag_context*, *selection_data*, *info*, *timestamp*, *user_param1*, *...*)
"drag−data−received"	def callback(*widget*, *drag_context*, *x*, *y*, *selection_data*, *info*, *timestamp*, *user_param1*, *...*)
"drag−drop"	def callback(*widget*, *drag_context*, *x*, *y*, *timestamp*, *user_param1*, *...*)
"drag−end"	def callback(*widget*, *drag_context*, *user_param1*, *...*)
"drag−leave"	def callback(*widget*, *drag_context*, *timestamp*, *user_param1*, *...*)
"drag−motion"	def callback(*widget*, *drag_context*, *x*, *y*, *timestamp*, *user_param1*, *...*)
"enter−notify−event"	def callback(*widget*, *event*, *user_param1*, *...*)
"event"	def callback(*widget*, *event*, *user_param1*, *...*)
"event−after"	def callback(*widget*, *event*, *user_param1*, *...*)
"expose−event"	def callback(*widget*, *event*, *user_param1*, *...*)
"focus"	def callback(*widget*, *direction*, *user_param1*, *...*)
"focus−in−event"	def callback(*widget*, *event*, *user_param1*, *...*)
"focus−out−event"	def callback(*widget*, *event*, *user_param1*, *...*)
"grab−focus"	def callback(*widget*, *user_param1*, *...*)
"grab−notify"	def callback(*widget*, *was_grabbed*, *user_param1*, *...*)
"hide"	def callback(*widget*, *user_param1*, *...*)
"hierarchy−changed"	def callback(*widget*, *previous_toplevel*, *user_param1*, *...*)
"key−press−event"	def callback(*widget*, *event*, *user_param1*, *...*)
"key−release−event"	def callback(*widget*, *event*, *user_param1*, *...*)
"leave−notify−event"	def callback(*widget*, *event*, *user_param1*, *...*)
"map"	def callback(*widget*, *user_param1*, *...*)
"map−event"	def callback(*widget*, *event*, *user_param1*, *...*)
"mnemonic−activate"	def callback(*widget*, *group_cycling*, *user_param1*, *...*)
"motion−notify−event"	def callback(*widget*, *event*, *user_param1*, *...*)
"no−expose−event"	def callback(*widget*, *event*, *user_param1*, *...*)
"parent−set"	def callback(*widget*, *old_parent*, *user_param1*, *...*)
"popup−menu"	def callback(*widget*, *user_param1*, *...*)
"property−notify−event"	def callback(*widget*, *event*, *user_param1*, *...*)
"proximity−in−event"	def callback(*widget*, *event*, *user_param1*, *...*)
"proximity−out−event"	def callback(*widget*, *event*, *user_param1*, *...*)
"realize"	def callback(*widget*, *user_param1*, *...*)
"screen−changed"	def callback(*widget*, *screen*, *user_param1*, *...*)
"scroll−event"	def callback(*widget*, *event*, *user_param1*, *...*)
"selection−clear−event"	def callback(*widget*, *event*, *user_param1*, *...*)

"selection–get"	`def callback(widget, selection_data, info, timestamp, user_param1, ...)`
"selection–notify–event"	`def callback(widget, event, user_param1, ...)`
"selection–received"	`def callback(widget, selection_data, timestamp, user_param1, ...)`
"selection–request–event"	`def callback(widget, event, user_param1, ...)`
"show"	`def callback(widget, user_param1, ...)`
"show–help"	`def callback(widget, help_type, user_param1, ...)`
"size–allocate"	`def callback(widget, allocation, user_param1, ...)`
"size–request"	`def callback(widget, requisition, user_param1, ...)`
"state–changed"	`def callback(widget, state, user_param1, ...)`
"style–set"	`def callback(widget, previous_style, user_param1, ...)`
"unmap"	`def callback(widget, user_param1, ...)`
"unmap–event"	`def callback(widget, event, user_param1, ...)`
"unrealize"	`def callback(widget, user_param1, ...)`
"visibility–notify–event"	`def callback(widget, event, user_param1, ...)`
"window–state–event"	`def callback(widget, event, user_param1, ...)`

Description

The `gtk.Widget` class is the base class for all `PyGTK` widgets. It provides the common set of method and signals for the widgets including:

- drag and drop setting and management methods
- selection methods
- methods to realize, map and show widgets
- methods to manage size allocation and requests
- methods to initiate widget redrawing
- methods to deal with the widget's place in the widget hierarchy
- event management methods
- methods to modify the style settings
- methods to access the default resources

`gtk.Widget` introduces style properties – these are basically object properties that are stored not on the object, but in the style object associated to the widget. Style properties are set in resource files. This mechanism is used for configuring such things as the location of the scrollbar arrows through the theme, giving theme authors more control over the look of applications without the need to write a theme engine in C.

Use the `gtk.widget.list_style_properties()` function to get information about existing style properties and the `style_get_property()` method to obtain the value of a style property.

Methods

gtk.Widget.get_allocation

```
    def get_allocation()
```

Returns :	a `gtk.gdk.Rectangle`

The `get_allocation()` method returns a `gtk.gdk.Rectangle` containing the bounds of the widget's allocation.

gtk.Widget.drag_check_threshold

```
    def drag_check_threshold(start_x, start_y, current_x, current_y)
```

`start_x` :	the X coordinate of start of drag
`start_y` :	the Y coordinate of start of drag
`current_x` :	the current X coordinate
`current_y` :	the current Y coordinate
Returns :	`TRUE` if the drag threshold has been passed.

The check_drag_threshold() method checks to see if a mouse drag starting at (*`start_x`*, *`start_y`*) and ending at (*`current_x`*, *`current_y`*) has passed the+ drag threshhold distance, and thus should trigger the beginning of a drag–and–drop operation.

gtk.Widget.drag_get_data

```
    def drag_get_data(context, target, time=0L)
```

`context` :	a `gtk.gdk.DragContext`
`target` :	an atom
`time` :	a timestamp or 0L to specify the current time

The `drag_get_data()` method gets the data associated with a drag specified by *`drag_context`* and *`target`*. When the data is received or the retrieval fails, a "drag_data_received" signal will be emitted. Failure of the retrieval is indicated by the length field of the `gtk.SelectionData` being negative. However, when the `drag_get_data()` method is called implicitly because `gtk.DRAG_DEFAULT_DROP` was set, the widget will not receive notification of failed drops.

gtk.Widget.drag_highlight

```
    def drag_highlight()
```

The `drag_highlight()` method draws a highlight around a widget. This will attach handlers to "expose_event" and "draw", so the highlight will continue to be displayed until the `drag_unhighlight()` method is called.

gtk.Widget.drag_unhighlight

```
    def drag_unhighlight()
```

The `drag_unhighlight()` method removes the highlight that was set by the `drag_highlight()` method.

gtk.Widget.drag_dest_set

```
    def drag_dest_set(flags, targets, actions)
```

`flags` :	the flags that specify what actions should be taken on behalf of a widget for drops onto that widget. The targets and actions fields only are used if `gtk.DEST_DEFAULT_MOTION` or `gtk.DEST_DEFAULT_DROP` are given.

targets :	a sequence of target tuples
actions :	a bitmask of possible actions for a drop onto this widget.

The `drag_dest_set()` method sets up a widget as a potential drag drop destination. The value of *`flags`* is a combination of the GTK Dest Defaults Constants.

`targets` is a sequence (list or tuple) of 3−tuples that contain information about the targets. The target data contains a string representing the drag type, target flags (a combination of `gtk.TARGET_SAME_APP` and `gtk.TARGET_SAME_WIDGET` or neither) and an application assigned integer ID used for identification purposes.

The value of *`actions`* is one of the GDK Drag Action Constants.

gtk.Widget.drag_dest_set_proxy

```
    def drag_dest_set_proxy(proxy_window, protocol, use_coordinates)
```

proxy_window :	the `gtk.gdk.Window` to forward drag events to
protocol :	the drag protocol that *`proxy_window`* accepts
use_coordinates :	if `TRUE`, send the same coordinates to the destination, because it is an embedded subwindow.

The `drag_dest_set_proxy()` method sets a proxy `gtk.gdk.Window` specified by *`proxy_window`* that drag events are forwarded to on behalf of the widget. The value of *`protocol`* is one of the GDK Drag Protocol Constants.

If *`use_coordinates`* is `TRUE`, the same coordinates are sent to the destination because the widget's an embedded subwindow.

gtk.Widget.drag_dest_unset

```
    def drag_dest_unset()
```

The `drag_dest_unset()` method clears the information about a drop destination set with the `drag_dest_set()` method. The widget will no longer receive notification of drags.

gtk.Widget.drag_dest_find_target

```
    def drag_dest_find_target(context, target_list)
```

context :	the drag context
target_list :	a list of droppable targets, or `None`.
Returns :	the first target that the source offers and the dest can accept, or `None`

The `dest_find_target()` method looks for a match between the targets in the `gtk.gdk.DragContext` specified by *`context`* and the *`target_list`*, returning the first matching target, or `NONE` if no match is found. The list specified by *`target_list`* should usually be the return value from the `drag_dest_get_target_list()` method, but some widgets may have different valid targets for different parts of the widget; in that case, they will have to implement a "drag−motion" handler that passes the correct target list to this method. *`target_list`* is a sequence (list or tuple) of 3−tuples that contain information about the targets. The target data contains a string representing the drag type, target flags (a combination of `gtk.TARGET_SAME_APP` and `gtk.TARGET_SAME_WIDGET` or neither) and an application assigned integer ID used for identification purposes.

gtk.Widget.drag_dest_get_target_list

```
    def drag_dest_get_target_list()
```

Returns :	the list of targets or `None` if no targets are set

The `drag_dest_get_target_list()` method returns the list of targets this widget can accept from drag–and–drop. The returned value is a sequence (list or tuple) of 3–tuples that contain information about the targets. The target data contains a string representing the drag type, target flags (a combination of `gtk.TARGET_SAME_APP` and `gtk.TARGET_SAME_WIDGET` or neither) and an application assigned integer ID used for identification purposes.

gtk.Widget.drag_dest_set_target_list

```
    def drag_dest_set_target_list(target_list)
```

`target_list` :	a list of droppable targets, or `None`

The `drag_dest_set_target_list()` method sets the target types (that this widget can accept from drag–and–drop) to the list specified by *`target_list`*. The widget must first be made into a drag destination with the `drag_dest_set()` method. *`target_list`* is a sequence (list or tuple) of 3–tuples that contain information about the targets. The target data contains a string representing the drag type, target flags (a combination of `gtk.TARGET_SAME_APP` and `gtk.TARGET_SAME_WIDGET` or neither) and an application assigned integer ID used for identification purposes.

gtk.Widget.drag_dest_add_image_targets

```
    def drag_dest_add_image_targets()
```

Note

This method is available in PyGTK 2.6 and above.

The `drag_dest_add_image_targets()` method adds the image targets supported by `gtk.SelectionData` to the target list of the widget's drag destination using an info value of 0.

gtk.Widget.drag_dest_add_text_targets

```
    def drag_dest_add_text_targets()
```

Note

This method is available in PyGTK 2.6 and above.

The `drag_dest_add_text_targets()` method adds the text targets supported by `gtk.SelectionData` to the target list of the widget's drag destination using an info value of 0.

gtk.Widget.drag_dest_add_uri_targets

```
    def drag_dest_add_uri_targets()
```

Note

This method is available in PyGTK 2.6 and above.

The `drag_dest_add_uri_targets`() method adds the URI targets supported by `gtk.SelectionData` to the target list of the widget's drag destination using an info value of 0.

gtk.Widget.drag_source_set

```
    def drag_source_set(start_button_mask, targets, actions)
```

`start_button_mask` :	the bitmask of buttons that can start the drag
`targets` :	a list of targets that the drag will support
`actions` :	the possible actions for a drag from this widget.

The `drag_source_set`() method sets up the widget to start a drag operation when the user clicks and drags on the widget. The widget must have a window. The value of start_button_mask is a combination of the GDK Modifier Constants.

`targets` is a sequence (list or tuple) of 3−tuples that contain information about the targets. The target data contains a string representing the drag type, target flags (a combination of `gtk.TARGET_SAME_APP` and `gtk.TARGET_SAME_WIDGET` or neither) and an application assigned integer ID used for identification purposes.

The value of *`actions`* is one of the GDK Drag Action Constants:

gtk.Widget.drag_source_unset

```
    def drag_source_unset()
```

The `drag_source_unset`() method unsets the drag source for the widget that was set up by the `drag_source_set()` method.

gtk.Widget.drag_source_set_icon

```
    def drag_source_set_icon(colormap, pixmap, mask=None)
```

`colormap` :	the colormap of the icon
`pixmap` :	the image data for the icon
`mask` :	the transparency mask for an image.

The `drag_source_set_icon`() method sets the icon that will be used for drags from the widget using the specified *`pixmap`* and *`mask`*. *`colormap`* specifies the colormap to be used to create the icon. The `drag_source_set_icon_pixbuf()` method should be used instead of this method.

gtk.Widget.drag_source_set_icon_pixbuf

```
    def drag_source_set_icon_pixbuf(pixbuf)
```

`pixbuf` :	the `gtk.gdk.Pixbuf` for the drag icon

The `drag_source_set_icon_pixbuf`() method sets the icon that will be used for drags from the widget from the `gtk.gdk.Pixbuf` specified by *`pixbuf`*.

gtk.Widget.drag_source_set_icon_stock

```
    def drag_source_set_icon_stock(stock_id)
```

stock_id :	the ID of the stock icon to use

The `drag_source_set_icon_stock()` method sets the icon that will be used for drags from a particular source using the stock icon specified by *`stock_id`*.

gtk.Widget.drag_source_get_target_list

```
    def drag_source_get_target_list()
```

Returns :	the list of targets or `None` if no targets are set

Note

This method is available in PyGTK 2.6 and above.

The `drag_source_get_target_list()` method returns the list of targets this widget can provide for drag–and–drop. The returned value is a sequence (list or tuple) of 3–tuples that contain information about the targets. The target data contains a string representing the drag type, target flags (a combination of `gtk.TARGET_SAME_APP` and `gtk.TARGET_SAME_WIDGET` or neither) and an application assigned integer ID used for identification purposes.

gtk.Widget.drag_source_set_target_list

```
    def drag_source_set_target_list(target_list)
```

target_list :	a list of droppable targets, or `None`

Note

This method is available in PyGTK 2.6 and above.

The `drag_source_set_target_list()` method sets the target types (that this widget can provide for drag–and–drop) to the list specified by *`target_list`*. The widget must first be made into a drag source with the `drag_source_set()` method. *`target_list`* is a sequence (list or tuple) of 3–tuples that contain information about the targets. The target data contains a string representing the drag type, target flags (a combination of `gtk.TARGET_SAME_APP` and `gtk.TARGET_SAME_WIDGET` or neither) and an application assigned integer ID used for identification purposes.

gtk.Widget.drag_source_add_text_targets

```
    def drag_source_add_text_targets()
```

Note

This method is available in PyGTK 2.6 and above.

The `drag_source_add_text_targets()` method adds the text targets supported by `gtk.SelectionData` to the target list of the widget's drag source using an info value of 0.

gtk.Widget.drag_begin

```
def drag_begin(targets, actions, button, event)
```

targets :	the list of targets supported by the widget drag
actions :	the allowed drag operations for the drag
button :	the button the user pressed to start the drag
event :	the gtk.gdk.Event that triggered the drag
Returns :	a new gtk.gdk.DragContext

The drag_begin() method starts a drag on the source side. The method only needs to be used when the application is starting drags itself, and is not needed when the drag_source_set() method is used. *targets* is a sequence (list or tuple) of 3−tuples that contain information about the targets. The target data contains a string representing the drag type, target flags (a combination of gtk.TARGET_SAME_APP and gtk.TARGET_SAME_WIDGET or neither) and an application assigned integer ID used for identification purposes.

The value of *actions* is one of the GDK Drag Action Constants.

button is the button that the user pressed to start the drag operation. *event* is the gtk.gdk.Event that triggered the start of the drag operation (the button press). This method returns the gtk.gdk.DragContext for the drag operation.

gtk.Widget.grab_add

```
def grab_add()
```

The grab_add() method makes the widget the current grabbed widget. This means that interaction with other widgets in the same application is blocked and mouse as well as keyboard events are delivered to this widget.

gtk.Widget.grab_remove

```
def grab_remove()
```

The grab_remove() method removes the grab from the widget. You have to pair calls to the grab_add() and grab_remove() methods.

gtk.Widget.rc_get_style

```
def rc_get_style()
```

Returns :	the resulting style.

The rc_get_style() method finds all matching RC styles for the widget, composites them together, and then creates a gtk.Style representing the composite appearance.

gtk.Widget.selection_owner_set

```
def selection_owner_set(selection, time=0L)
```

selection :	an atom representing the selection to claim
time :	a timestamp or 0L to use the current time
Returns :	TRUE if successful

The `selection_owner_set()` method claims the ownership of the selection specified by *`selection`* for the widget.

gtk.Widget.selection_add_target

```
def selection_add_target(selection, target, info)
```

`selection` :	an atom representing the selection
`target` :	an atom representing the target for the selection
`info` :	an integer ID that will be passed to the application

The `selection_add_target()` method adds the specified *`target`* to the list of supported targets for the specified *`selection`*. *`info`* is an integer ID that will be passed to the application when the "selection–get" handler is called.

gtk.Widget.selection_add_targets

```
def selection_add_targets(selection, targets)
```

`selection` :	an atom representing the selection
`targets` :	a sequence of 3–tuples containing target data

The `selection_add_targets()` method adds the list of targets (specified by *`targets`*) to the list of supported targets for the specified *`selection`*. targets is a sequence (Python tuple or list) of 3–tuples that contain information about the targets. The target data contains a string representing the drag type, target flags (a combination of `gtk.TARGET_SAME_APP` and `gtk.TARGET_SAME_WIDGET` or neither) and an application assigned integer ID used for identification purposes.

gtk.Widget.selection_clear_targets

```
def selection_clear_targets(selection)
```

`selection` :	an atom representing a selection

The `selection_clear_targets()` method remove all targets registered for the specified *`selection`* for the widget.

gtk.Widget.selection_convert

```
def selection_convert(selection, target, time=0L)
```

`selection` :	an atom specifying the selection
`target` :	an atom specifying the target type
`time` :	a timestamp for the request or 0L to use the current time
Returns :	`TRUE` if the request succeeded

The `selection_convert()` method requests the contents of the specified *`selection`* for the specified *`target`* type. When received, a "selection_received" signal will be generated.

gtk.Widget.selection_remove_all

```
def selection_remove_all()
```

The `selection_remove_all()` method removes all handlers and unsets ownership of all selections for a widget. This method is called when widget is being destroyed and not usually by applications.

gtk.Widget.destroy

```
    def destroy()
```

The `destroy`() method destroys the widget. When a widget is destroyed, it will break any references it holds to other objects. If the widget is inside a container, the widget will be removed from the container. If the widget is a toplevel (derived from `gtk.Window`), it will be removed from the list of toplevels, and the reference `PyGTK` holds to it will be removed. Removing a widget from its container or the list of toplevels results in the widget being finalized. In most cases, only toplevel widgets (windows) require explicit destruction, because when you destroy a toplevel its children will be destroyed as well.

gtk.Widget.unparent

```
    def unparent()
```

The `unparent`() method is only for use in widget implementations. It should be called by implementations of the remove method on a `gtk.Container`, to dissociate a child widget from the container.

gtk.Widget.show

```
    def show()
```

The `show`() method causes a widget to be displayed as soon as practical. Any widget that isn't shown will not appear on the screen. If you want to show all the widgets in a container, it's easier to call the `show_all()` on the container, instead of individually showing the widgets. Of course you have to show the containers containing a widget, as well as the widget itself, before it will appear onscreen. When a toplevel container is shown, it is immediately realized and mapped; other shown widgets are realized and mapped when their toplevel container is realized and mapped.

gtk.Widget.show_now

```
    def show_now()
```

The `show_now`() method is the same as the `show()` method except if the widget is an unmapped toplevel widget (i.e. a `gtk.Window` that has not yet been shown), it enters the main loop and waits for the window to actually be mapped.

Note

Because the main loop is running, anything can happen during this method.

gtk.Widget.hide

```
    def hide()
```

The `hide`() method reverses the effects of the `show()` method, causing the widget to be hidden (removed from the display) by unmapping it.

gtk.Widget.show_all

```
    def show_all()
```

The `show_all`() method recursively shows the widget, and any child widgets (if the widget is a container).

gtk.Widget.hide_all

```
    def hide_all()
```

The `hide_all()` method recursively hides the widget and its child widgets (if any).

gtk.Widget.set_no_show_all

```
    def set_no_show_all(no_show_all)
```

`no_show_all` :	the new value for the "no_show_all" property

Note

This method is available in PyGTK 2.4 and above.

The `set_no_show_all()` method sets the "no_show_all" property to the value of *`no_show_all`*. If *`no_show_all`* is `TRUE` calls to the `show_all()` and `hide_all()` methods will not affect this widget.

This method is mostly for use in constructing widget hierarchies with externally controlled visibility, see the `gtk.UIManager` reference for mote information.

gtk.Widget.get_no_show_all

```
    def get_no_show_all()
```

Returns :	the current value of the "no_show_all" property.

Note

This method is available in PyGTK 2.4 and above.

The `get_no_show_all()` method returns the current value of the "no_show_all" property. If "no−show−all" is `TRUE` calls to the `show_all()` and `hide_all()` methods will not affect the widget.

gtk.Widget.map

```
    def map()
```

The `map()` method maps the widget (causes it to be displayed). This method will also cause the widget to be realized if it is not currently realized. This method is usually not used by applications.

gtk.Widget.unmap

```
    def unmap()
```

The `unmap()` method unmaps the widget (causes it to be removed from the display). This method is not usually used by applications.

gtk.Widget.realize

```
    def realize()
```

The `realize()` method creates the resources associated with a widget. For example, the widget `gtk.gdk.Window` will be created when the widget is realized. Normally realization happens implicitly; if you show a widget and all its parent containers, then the widget will be realized and mapped automatically. Realizing a widget requires all the widget's parent widgets to be realized; calling the `realize()` method realizes the widget's parents in addition to the widget itself. A widget must be inside a toplevel window when you realize it. This method is primarily used in widget implementations, and not in applications. Many times when you think you might need it, a better approach is to connect to a signal that will be called after the widget is realized automatically, such as "expose_event". Or simply using the `gobject.connect_after()` method to add a handler to the "realize" signal.

gtk.Widget.unrealize

```
    def unrealize()
```

The `unrealize()` method frees all resources associated with the widget, such as the `gtk.gdk.Window`.

gtk.Widget.queue_draw

```
    def queue_draw()
```

The `queue_draw()` method is equivalent to calling the `queue_draw_area()` method for the entire area of a widget.

gtk.Widget.queue_draw_area

```
    def queue_draw_area(x, y, width, height)
```

`x` :	the x coordinate of upper–left corner of rectangle to redraw
`y` :	the y coordinate of upper–left corner of rectangle to redraw
`width` :	the width of rectangle to redraw
`height` :	the height of rectangle to redraw

The `queue_draw_area()` method invalidates the rectangular area of the widget specified by *`x`*, *`y`*, *`width`* and *`height`* by calling the `gtk.gdk.Window.invalidate_rect()` method on the widget's window and all its child windows. Once the main loop becomes idle (after the current batch of events has been processed, roughly), the window will receive expose events for the union of all regions that have been invalidated.

Normally you would only use this method in widget implementations. But you might also use it, or the `gtk.gdk.Window.invalidate_rect()` method directly, to schedule a redraw of a `gtk.DrawingArea` or some portion thereof. Frequently you can just call the `gtk.gdk.Window.invalidate_rect()` method instead of this method. This method will invalidate only a single window, instead of the widget and all its children. The advantage of adding to the invalidated region compared to simply drawing immediately is efficiency; using an invalid region ensures that you only have to redraw one time.

gtk.Widget.queue_resize

```
    def queue_resize()
```

The `queue_resize()` method schedules the widget to have its size renegotiated. This method should be called when a widget for some reason has a new size request. For example, when you change the text in a `gtk.Label`, a resize is queued to ensure there's enough space for the new text.

gtk.Widget.queue_resize_no_redraw

```
def queue_resize_no_redraw()
```

Note

This method is available in PyGTK 2.4 and above.

The `queue_resize_no_redraw`() method works like the `queue_resize()` method, except that the widget is not invalidated.

gtk.Widget.size_request

```
def size_request()
```

Returns :	a tuple containing the widget's required width and height.

The `size_request`() method returns the preferred size of a widget as a tuple containing its required width and height. This method is typically used when implementing a `gtk.Container` subclass to arrange the container's child widgets and decide what size allocations to give them with the `size_allocate()` method. Obtaining a size request requires that the widget be associated with a screen, because font information may be needed.

Also remember that the size request is not necessarily the size a widget will actually be allocated. See the `get_child_requisition()` method.

gtk.Widget.size_allocate

```
def size_allocate(allocation)
```

`allocation` :	the position and size to be allocated to the widget

The `size_allocate`() method sets a size allocation for the widget using the `gtk.gdk.Rectangle` specified by *`allocation`*. This method is only used by `gtk.Container` subclasses, to assign a size and position to their child widgets.

gtk.Widget.get_child_requisition

```
def get_child_requisition(requisition)
```

Returns :	a tuple containing the required size of the widget

The `get_child_requisition`() method returns a tuple containing the widget requisition width and height. This method is only for use in widget container implementations since it obtains the widget requisition directly. By comparison the `size_request()` method actually computes the size request and fills in the widget requisition before returning. Because this method does not recalculate the size request, it can only be used when you know that the widget requisition is up–to–date, i.e. the `size_request()` method has been called since the last time a resize was queued. In general, only container implementations have this information; applications should use the `size_request()` method instead.

gtk.Widget.add_accelerator

```
def add_accelerator(accel_signal, accel_group, accel_key, accel_mods, accel_flags)
```

`accel_signal` :	the widget signal to emit on accelerator activation
`accel_group` :	the accel group for this widget, added to its toplevel

accel_key :	the keyval of the accelerator e.g. `ord('q')`
accel_mods :	the modifier key combination of the accelerator
accel_flags :	the flag accelerators, e.g. `gtk.ACCEL_VISIBLE`

The `add_accelerator()` method installs an accelerator for the widget in *accel_group* that causes *accel_signal* to be emitted if the accelerator is activated. The accelerator key and modifiers are specified by *accel_key* and *accel_mods* respectively. *accel_mods* should be a combination of the GDK Modifier Constants. *accel_flags* is a combination of `gtk.ACCEL_VISIBLE` and `gtk.ACCEL_LOCKED` (see the GTK Accel Flags Constants). The *accel_group* needs to be added to the widget's toplevel via the `gtk.Window.add_accel_group()` method and the signal specified by *accel_signal* must have signal flags that include the `gobject.SIGNAL_ACTION` flag (see the GObject Signal Flag Constants for more information). Accelerators added through this method are not user changeable during runtime. If you want to support accelerators that can be changed by the user, the `set_accel_path()` or `gtk.MenuItem.set_accel_path()` methods instead.

gtk.Widget.remove_accelerator

```
    def remove_accelerator(accel_group, accel_key, accel_mods)
```

accel_group :	the accel group for this widget
accel_key :	the keyval of the accelerator
accel_mods :	the modifier key combination of the accelerator
Returns :	`TRUE` if the accelerator was removed

The `remove_accelerator()` method removes the accelerator specified by *accel_key* and *accel_mods* from the widget's accelerator group (specified by *accel_group*), previously installed with the `add_accelerator()` method.

gtk.Widget.set_accel_path

```
    def set_accel_path(accel_path, accel_group)
```

accel_path :	the path used to look up the the accelerator
accel_group :	a `gtk.AccelGroup`.

The `set_accel_path()` method sets an accelerator (using the key bindings defined in *accel_path*) in the accelerator group specified by *accel_group*. This method removes any accelerators for any accelerator group installed by previous calls to the `set_accel_path()` method. Associating accelerators with paths allows them to be modified by the user and the modifications to be saved for future use. This method is a low level method that would most likely be used by a menu creation system like `gtk.ItemFactory`. If you use `gtk.ItemFactory`, setting up accelerator paths will be done automatically. Even when you you aren't using `gtk.ItemFactory`, if you only want to set up accelerators on menu items the `gtk.MenuItem.set_accel_path()` method provides a somewhat more convenient interface.

gtk.Widget.can_activate_accel

```
    def can_activate_accel(signal_id)
```

signal_id :	the ID of an installed signal
Returns :	`TRUE` if the accelerator can be activated.

Note

This method is available in PyGTK 2.4 and above.

The `can_activate_accel()` method returns `TRUE` if an accelerator that activates the signal specified by *`signal_id`* can currently be activated. This is done by emitting the "can−activate−accel" signal. If the signal isn't overridden by a handler or in a derived widget, then the default check is that the widget must be sensitive, and the widget and all its ancestors mapped.

gtk.Widget.mnemonic_activate

```
    def mnemonic_activate(group_cycling)
```

`group_cycling` :	if TRUE grab the focus instead of activating the widget
Returns :	TRUE if the signal was handled

The `mnemonic_activate()` method emits the "mnemonic−activate" signal on the widget and returns `TRUE` if the signal was handled. *`group_cycling`* is `TRUE` if the focus is being shifted to the widget and `FALSE` if the widget should be activated.

gtk.Widget.event

```
    def event(event)
```

`event` :	a `gtk.gdk.Event`
Returns :	`TRUE` if the event was handled

The `event()` method emits the event signals on a widget (those signals should never be emitted without using this method to do so). If you want to synthesize an event though, don't use this method; instead, use the `gtk.main_do_event()` function so the event will behave as if it were in the event queue. Don't synthesize expose events; instead, use the `gtk.gdk.Window.invalidate_rect()` method to invalidate a region of the window.

gtk.Widget.send_expose

```
    def send_expose(event)
```

`event` :	an expose `gtk.gdk.Event`
Returns :	`TRUE` if the event was handled

The `send_expose()` method emits an expose event signal on a widget. This method is usually used when propagating an expose event to a child `NO_WINDOW` widget, and that is normally done using the `gtk.Container.propagate_expose()` method. If you want to force an area of a window to be redrawn, use the `gtk.gdk.Window.invalidate_rect()` method. To cause the redraw to be done immediately, follow that call with a call to the `gtk.gdk.Window.process_updates()` method.

gtk.Widget.activate

```
    def activate()
```

Returns :	`TRUE` if the widget was activatable

The `activate()` method emits the "activate" signal on the widget that activates it (if it can be activated). Activation is what happens when you press **Enter** on a widget during key navigation; clicking a button, selecting a menu item, etc. If the widget isn't activatable, the method returns `FALSE`.

gtk.Widget.set_scroll_adjustments

```
    def set_scroll_adjustments(hadjustment, vadjustment)
```

hadjustment :	an adjustment for horizontal scrolling, or `None`
vadjustment :	an adjustment for vertical scrolling, or `None`
Returns :	`TRUE` if the widget supports scrolling

The `set_scroll_adjustments()` method sets the horizontal and vertical scroll adjustments specified by *`hadjustment`* and *`vadjustment`* respectively and returns `TRUE`. If the widget doesn't support scrolling this method returns `FALSE`. Widgets that don't support scrolling can be scrolled by placing them in a `gtk.Viewport`, which does support scrolling. This method emits the "set–scroll–adjustments" signal on the widget.

gtk.Widget.reparent

```
    def reparent(new_parent)
```

new_parent :	a `gtk.Container` to move the widget into

The `reparent()` method moves a widget from one `gtk.Container` to another.

gtk.Widget.intersect

```
    def intersect(area, intersection)
```

area :	a rectangle
Returns :	a rectangle of the intersection of the widget and *`area`* or `None`

The `intersect()` method computes the intersection of a the widget's area and *`area`*, and returns the intersection in a `gtk.gdk.Rectangle`. This method returns `FALSE` if there is no intersection.

gtk.Widget.freeze_child_notify

```
    def freeze_child_notify()
```

The `freeze_child_notify()` method freezes the child notify queue that is used to notify child widgets of child property changes.

gtk.Widget.child_notify

```
    def child_notify(child_property)
```

child_property :	a child property

The `child_notify()` method adds a child property to the widget's child notify queue that is used to notify child widgets of a change to a child property.

gtk.Widget.thaw_child_notify

```
    def thaw_child_notify()
```

The `thaw_child_notify()` method reverses the effect of a previous call to the `freeze_child_notify()` method.

gtk.Widget.is_focus

```
    def is_focus()
```

Returns :	`TRUE` if the widget is the focus widget.

The `is_focus()` method returns `TRUE` if the widget is the focus widget within its toplevel. This does not mean that the `gtk.HAS_FOCUS` flag is necessarily set; `gtk.HAS_FOCUS` will only be set if the toplevel widget additionally has the global input focus.

gtk.Widget.grab_focus

```
    def grab_focus()
```

The `grab_focus()` method causes the widget to have the keyboard focus for it's enclosing `gtk.Window`. The widget must be a focusable widget, such as a `gtk.Entry`. Also, the widget must have the `gtk.CAN_FOCUS` flag set.

gtk.Widget.grab_default

```
    def grab_default()
```

The `grab_default()` method causes the widget to become the default widget. The widget must have the `gtk.CAN_DEFAULT` flag set by calling the `gtk.Object.set_flags()` method. The default widget is activated when the user presses **Enter** in a window.

gtk.Widget.set_name

```
    def set_name(name)
```

`name` :	the name for the widget

The `set_name()` method sets the "name" property of the widget to the string specified by *`name`*. Widgets can be named, which allows you to refer to them in a GTK resource file.

gtk.Widget.get_name

```
    def get_name()
```

Returns :	the name of the widget

The `get_name()` method returns the value of the "name" property that contains the name of the widget or `None` if the widget has no name.

gtk.Widget.set_state

```
    def set_state(state)
```

`state` :	the new state for the widget

The set_state() method sets the state of the widget to the value specified by state. The value of state must be one of the GTK State Type Constants.

Usually you should set the state using wrapper methods such as `set_sensitive()`.

gtk.Widget.set_sensitive

```
    def set_sensitive(sensitive)
```

`sensitive` :	if `TRUE` make the widget sensitive

The `set_sensitive()` method sets the "sensitive" property of the widget to the value specified by *`sensitive`*. If *`sensitive`* is `TRUE` the widget will be sensitive and the user can interact with it. An insensitive widget appears "grayed out" and the user can't interact with it. Insensitive widgets are known as "inactive", "disabled", or "ghosted" in some other toolkits.

gtk.Widget.set_app_paintable

```
    def set_app_paintable(app_paintable)
```

`app_paintable` :	if `TRUE` the application will paint directly on the widget

The `set_app_paintable()` method sets the "app−paintable" property to the value of *`app_paintable`*. If *`app_paintable`* is `TRUE` the application will paint directly on the widget.

gtk.Widget.set_double_buffered

```
    def set_double_buffered(double_buffered)
```

`double_buffered` :	if `TRUE` double−buffer a widget

The `set_double_buffered()` method sets the widget's flags according to the value of *`double_buffered`*. If *`double_buffered`* is `TRUE` the `gtk.DOUBLE_BUFFERED` flag is set; otherwise it is unset. Widgets are double buffered by default. "Double buffered" simply means that the `gtk.gdk.Window.begin_paint_rect()` and `gtk.gdk.Window.end_paint()` methods are called automatically around expose events sent to the widget. The `gtk.gdk.Window.begin_paint_rect()` method diverts all drawing to a widget's window to an off screen buffer, and the `gtk.gdk.Window.end_paint()` method draws the buffer to the screen. The result is that users see the window update in one smooth step, and don't see individual graphics primitives being rendered. In very simple terms, double buffered widgets don't flicker, so you would only use this method to turn off double buffering if you had special needs and really knew what you were doing.

gtk.Widget.set_redraw_on_allocate

```
    def set_redraw_on_allocate(redraw_on_allocate)
```

`redraw_on_allocate` :	if `TRUE`, the entire widget will be redrawn when it is allocated to a new size. Otherwise, only the new portion of the widget will be redrawn.

The `set_redraw_on_allocate()` method sets a flag that determines if the entire widget is queued for drawing when a widget's size allocation changes. By default, this setting is `TRUE` and the entire widget is redrawn on every size change. If your widget leaves the upper left are unchanged when made bigger, turning this setting on will improve performance.

Note

For `NO_WINDOW` widgets setting this flag to `FALSE` turns off all allocation on resizing: the widget will not redraw even if its position changes; this is to allow containers that don't draw anything to avoid excess invalidations. If you set this flag on a `NO_WINDOW` widget that *does* draw on the widget's `gtk.gdk.Window`, you are responsible for invalidating both the old and new allocation of the widget when the widget is moved and responsible for invalidating regions newly when the widget increases size.

gtk.Widget.set_parent

```
    def set_parent(parent)
```

`parent` :	a parent container

The `set_parent()` method is useful only when implementing subclasses of `gtk.Container`. This method sets the container as the parent of the widget, and takes care of some details such as updating the state and style of the child to reflect its new location. The reverse method is the `unparent()` method.

gtk.Widget.set_parent_window

```
    def set_parent_window(parent_window)
```

`parent_window` :	the new parent window.

The `set_parent_window()` method sets a non default parent window for the widget.

gtk.Widget.set_child_visible

```
    def set_child_visible(is_visible)
```

`is_visible` :	if `TRUE`, the widget should be mapped along with its parent.

The `set_child_visible()` method determines if the widget should be mapped along with its parent. If *`is_visible`* is `TRUE` the widget will be mapped with its parent if it has called the `show()` method.

The child visibility can be set for widget before it is added to a container to avoid mapping children unnecessarily. The widget's child visibility flag will be reset to its default state of `TRUE` when the widget is removed from a container. Note that changing the child visibility of a widget does not queue a resize on the widget. Most of the time, the size of a widget is computed from all visible children, whether or not they are mapped. If this is not the case, the container can queue a resize itself. This method is only useful for container implementations and never should be called by an application.

gtk.Widget.get_child_visible

```
    def get_child_visible()
```

Returns :	`TRUE` if the widget is mapped with the parent.

The `get_child_visible()` method returns the value set with the `set_child_visible()` method. This method is only useful for container implementations and never should be called by an application.

gtk.Widget.get_parent

```
    def get_parent()
```

Returns :	the parent container of the widget, or `None`

The `get_parent()` method returns the parent container of the widget or `None` if the widget has no parent.

gtk.Widget.get_parent_window

```
    def get_parent_window()
```

Returns :	the parent `gtk.gdk.Window` of the widget

The `get_parent_window()` method returns the widget's parent `gtk.gdk.Window`.

gtk.Widget.child_focus

```
    def child_focus(direction)
```

direction :	the direction of focus movement
Returns :	TRUE if focus ended up inside the widget

The child_focus() method is used by custom widget implementations. If you're writing an application, use the grab_focus() method to move the focus to a particular widget, and the gtk.Container.set_focus_chain() method to change the focus tab order.

The child_focus() method is called by containers as the user moves around the window using keyboard shortcuts. The value of *direction* indicates what kind of motion is taking place: gtk.DIR_TAB_FORWARD, gtk.DIR_TAB_BACKWARD, gtk.DIR_UP, gtk.DIR_DOWN, gtk.DIR_LEFT or gtk.DIR_RIGHT

This method emits the "focus" signal on the widget. Widgets override the default handler for this signal in order to implement appropriate focus behavior. The "focus" default handler for a widget should return:

- TRUE if the focus is left on a focusable location inside the widget, and
- FALSE if the focus moved outside the widget

If returning TRUE, widgets normally call the grab_focus() method to place the focus accordingly; if returning FALSE, they don't modify the current focus location.

gtk.Widget.set_size_request

```
    def set_size_request(width, height)
```

width :	the width the widget should request, or –1 to unset
height :	the height the widget should request, or –1 to unset

The set_size_request() method sets the minimum size of a widget to the values specified by *width* and *height*. You can use this method to force a widget to be either larger or smaller than it normally would be. In most cases, the gtk.Window.set_default_size() is a better choice for toplevel windows than this method. Setting the default size will still allow users to shrink the window but setting the size request will force them to leave the window at least as large as the size request. When dealing with window sizes, the gtk.Window.set_geometry_hints() can be a useful method as well.

Note

There is an inherent danger when setting any fixed size – themes, translations into other languages, different fonts, and user action can all change the appropriate size for a given widget. So, it's basically impossible to hard code a size that will always be correct.

The size request of a widget is the smallest size a widget can accept while still functioning well and drawing itself correctly. However in some strange cases a widget may be allocated less than its requested size, and in many cases a widget may be allocated more space than it requested. If the size request in a given direction is –1 (unset), then the "natural" size request of the widget will be used instead. Widgets can't actually be allocated a size less than 1 by 1, but you can pass 0,0 to this method to mean "as small as possible".

gtk.Widget.get_size_request

```
    def get_size_request()
```

Returns :	a 2–tuple containing the requested width and height

The get_size_request() method returns a 2–tuple containing the width and height of the widget that was explicitly set for the widget using the set_size_request(). A value of –1 for the width or height indicates that that dimension has not been set explicitly and the natural requisition of the widget will be used instead. See the set_size_request() method for more information. To get the size a widget will actually use, call the size_request() instead of this method.

gtk.Widget.set_events

```
def set_events(events)
```

events :	the event mask

The set_events() method sets the event mask for a widget using the value specified by *events*. The event mask determines which events a widget will receive. Keep in mind that different widgets have different default event masks, and by changing the event mask you may disrupt a widget's functionality, so be careful. This method must be called while a widget is unrealized. Consider using the add_events() method for widgets that are already realized, or if you want to preserve the existing event mask. This method can't be used with gtk.NO_WINDOW widgets since a widget must have a gtk.gdk.Window to receive events. To get events on gtk.NO_WINDOW widgets, place them inside a gtk.EventBox and receive events on the event box.

The value of *events* must be a combination of the GDK Event Mask Flag Constants:

gtk.Widget.add_events

```
def add_events(events)
```

events :	an event mask

The add_events() method adds the events specified by *events* to the event mask for the widget. See the set_events() method for details.

gtk.Widget.set_extension_events

```
def set_extension_events(mode)
```

mode :	the extension events to receive

The set_extension_events() method sets the extension events mask to the value specified by *mode*. The value of mode must be one of the GDK Extension Mode Constants.

See the gtk.gdk.Window.input_set_extension_events() method for more information.

gtk.Widget.get_extension_events

```
def get_extension_events()
```

Returns :	the extension events for the widget

The get_extension_events() method returns the extension events the widget will receive. See the gtk.gdk.Window.input_set_extension_events() method for more information.

gtk.Widget.get_toplevel

```
def get_toplevel()
```

Returns :	the topmost ancestor of the widget, or the widget itself if there's no ancestor.

The `get_toplevel()` method returns the topmost widget in the container hierarchy that the widget is a part of. If the widget has no parent widgets, it will be returned as the topmost widget.

Note the difference in behavior as compared to the `get_ancestor()` method that returns `None` if the widget isn't inside a toplevel window, and if the window is inside a widget derived from `gtk.Window` that is in turn inside the toplevel `gtk.Window`. While the second case may seem unlikely, it actually happens when a `gtk.Plug` is embedded inside a `gtk.Socket` within the same application.

To reliably find the toplevel `gtk.Window`, use the `get_toplevel()` method and check if the `gtk.TOPLEVEL` flag is set on the result.

gtk.Widget.get_ancestor

```
def get_ancestor(widget_type)
```

`widget_type` :	a widget type
Returns :	the ancestor widget, or `None` if not found

The `get_ancestor()` method returns the first ancestor of the widget with the type specified by *`widget_type`*. For example:

```
widget.get_ancestor(gtk.Box)
```

returns the first `gtk.Box` that's an ancestor of the widget. See the `get_toplevel()` method for information about checking for a toplevel `gtk.Window`.

gtk.Widget.get_colormap

```
def get_colormap()
```

Returns :	the colormap used by the widget

The `get_colormap()` method returns the colormap that will be used to render the widget.

gtk.Widget.get_visual

```
def get_visual()
```

Returns :	the visual for the widget

The `get_visual()` method returns the visual that will be used to render the widget.

gtk.Widget.get_screen

```
def get_screen()
```

Returns :	the `gtk.gdk.Screen` for the toplevel for this widget.

Note

This method is available in PyGTK 2.2 and above.

The `get_screen`() method returns the `gtk.gdk.Screen` from the toplevel window associated with the widget. This method can only be called after the widget has been added to a widget hierarchy with a `gtk.Window` at the top.

gtk.Widget.has_screen

```
def has_screen()
```

Returns :	`TRUE` if there is a `gtk.gdk.Screen` associated with the widget.

Note

This method is available in PyGTK 2.2 and above.

The `has_screen`() method returns `TRUE` if a `gtk.gdk.Screen` is associated with the widget. All toplevel widgets have an associated screen, as do all widgets added into a hierarchy with a toplevel window.

gtk.Widget.get_display

```
def get_display()
```

Returns :	the `gtk.gdk.Display` for the toplevel for this widget.

Note

This method is available in PyGTK 2.2 and above.

The `get_display`() method returns the `gtk.gdk.Display` for the toplevel window associated with the widget. This method can only be called after the widget has been added to a widget hierarchy with a toplevel `gtk.Window`

gtk.Widget.get_root_window

```
def get_root_window()
```

Returns :	the `gtk.gdk.Window` root window for the toplevel for this widget.

Note

This method is available in PyGTK 2.2 and above.

The `get_root_window`() method returns the root window containing the widget. This method should only be called after the widget has been added to a widget hierarchy with a toplevel `gtk.Window`

The root window is useful for such purposes as creating a popup `gtk.gdk.Window` associated with the window.

gtk.Widget.get_settings

```
def get_settings()
```

Returns :	the associated gtk.Settings object

The `get_settings()` method returns the settings object holding the settings (global property settings, RC file information, etc) used for this widget.

gtk.Widget.get_clipboard

```
def get_clipboard(selection)
```

selection :	a gtk.gdk.Atom or string that identifies the clipboard to use. `gtk.gdk.SELECTION_CLIPBOARD` gives the default clipboard. Another common value is `gtk.gdk.SELECTION_PRIMARY`, which gives the primary X selection.
Returns :	the appropriate gtk.Clipboard object. If no clipboard already exists, a new one will be created. Once a clipboard object has been created, it is persistent for all time.

Note

This method is available in PyGTK 2.2 and above.

The `get_clipboard()` method returns the gtk.Clipboard object for the selection specified by *selection*. The widget must have a gtk.gdk.Display associated with it, and so must be attached to a toplevel window.

gtk.Widget.set_colormap

```
def set_colormap(colormap)
```

colormap :	a gtk.gdk.Colormap

The `set_colormap()` method sets the gtk.gdk.Colormap for the widget to the value specified by *colormap*. Widget must not have been realized.

gtk.Widget.get_events

```
def get_events()
```

Returns :	the event mask for the widget

The `get_events()` method returns the event mask for the widget that determines the events that the widget will receive. See the set_events() method for more detail about events.

gtk.Widget.get_pointer

```
def get_pointer()
```

Returns :	a tuple containing the X and Y coordinates of the mouse pointer

The `get_pointer()` method returns a tuple containing the location of the mouse pointer in widget coordinates. Widget coordinates are a bit odd; for historical reasons, they are defined as:

- the widget gtk.gdk.Window coordinates for widgets that are not `gtk.NO_WINDOW` widgets, or
- the coordinates relative to the widget allocation for widgets that are `gtk.NO_WINDOW` widgets.

gtk.Widget.is_ancestor

```
def is_ancestor(ancestor)
```

ancestor :	another `gtk.Widget`
Returns :	`TRUE` if *ancestor* contains the widget as a child, grandchild, great grandchild, etc.

The `is_ancestor()` method returns `TRUE` if the widget is somewhere inside the hierarchy of the widget specified by*ancestor*

gtk.Widget.translate_coordinates

```
def translate_coordinates(dest_widget, src_x, src_y)
```

dest_widget :	a `gtk.Widget`
src_x :	the X position relative to the widget
src_y :	the Y position relative to the widget
Returns :	a tuple containing the X and Y position relative to *dest_widget*

The `translate_coordinates()` method returns a tuple containing the translation of the widget x and y coordinates specified by *src_x* and *src_y* respectively to coordinates relative to *dest_widget*. In order to perform this operation, both widgets must be realized, and must share a common toplevel.

gtk.Widget.hide_on_delete

```
def hide_on_delete()
```

Returns :	`TRUE`

The `hide_on_delete()` method is a utility method that is intended to be connected to the "delete_event" signal on a `gtk.Window`. The method calls the `hide()` method on the widget, then returns `TRUE`. If connected to "delete_event", the result is that clicking the close button for a window (on the window frame, top right corner usually) will hide but not destroy the window. By default, `PyGTK` destroys windows when "delete_event" is received.

gtk.Widget.set_style

```
def set_style(style)
```

style :	a `gtk.Style`, or `None` to revert to the default style

The `set_style()` method sets the "style" property to the value of *style*. The "style" property contains the `gtk.Style` for the widget. This method interacts badly with themes, because themes work by replacing the `gtk.Style`.

gtk.Widget.ensure_style

```
def ensure_style()
```

The `ensure_style()` method makes sure that the widget has a style. This method is useful if applied to an unrealized widget. Usually, if you want the style, the widget is realized, and guaranteed to have a style.

gtk.Widget.get_style

```
def get_style()
```

Returns :	the widget's `gtk.Style`

The `get_style()` method returns the value of the "style" property.

gtk.Widget.modify_style

```
def modify_style(style)
```

`style` :	the `gtk.RcStyle` holding the style modifications

The `modify_style()` method modifies the style values on the widget using the values in *`style`*. Modifications made using this technique take precedence over style values set via an RC file, however, they will be overridden if a style is explicitly set on the widget using the `set_style()` method. The `gtk.RcStyle` object is designed so each attribute can either be set or unset, so it is possible, using this method, to modify some style values and leave the others unchanged.

Note that modifications made with this method are not cumulative with previous calls to the `modify_style()` method or with such methods as the `modify_fg()` method. If you wish to retain previous values, you must first call the `get_modifier_style()` method, make your modifications to the returned style, then call the `modify_style()` method with that style. On the other hand, if you first call the `modify_style()` method, subsequent calls to such methods as the `modify_fg()` method will have a cumulative effect with the initial modifications.

gtk.Widget.get_modifier_style

```
def get_modifier_style()
```

Returns :	the modifier style for the widget. This `gtk.RcStyle` is owned by the widget.

The `get_modifier_style()` method returns the current modifier style for the widget as set by the `modify_style()` method. If no style was previously set, a new `gtk.RcStyle` object will be created(with all values unset), and set as the modifier style for the widget. If you make changes to this rc style, you must call the `modify_style()` method, passing in the returned rc style, to make sure that your changes take effect.

Caution

Passing the style back to the `modify_style()` method will normally end up destroying it, because the `modify_style()` method copies the passed–in style and sets the copy as the new modifier style, thus dropping any reference to the old modifier style.

gtk.Widget.modify_fg

```
def modify_fg(state, color)
```

`state` :	a widget state.
`color` :	the `gtk.gdk.Color` to assign.

The `modify_fg()` method sets the foreground color to the `gtk.gdk.Color` specified by *`color`* for the widget in the specified *`state`*. All other style values are left untouched. The value of state must be one of the GTK State Type Constants.

gtk.Widget.modify_bg

```
    def modify_bg(state, color)
```

state :	a widget state.
color :	the gtk.gdk.Color to assign

The modify_bg() method sets the background color to the gtk.gdk.Color specified by *color* for the widget in the specified *state*. All other style values are left untouched. See modify_fg() method for detail on the possible values of *state*.

Note

modify_bg() only affects widgets that have an associated gtk.gdk.Window. Widgets that do not have an associated window include gtk.Arrow, gtk.Bin, gtk.Box, gtk.Button, gtk.CheckButton, gtk.Fixed, gtk.Image, gtk.Label, gtk.MenuItem, gtk.Notebook, gtk.Paned, gtk.RadioButton, gtk.Range, gtk.ScrolledWindow, gtk.Separator, gtk.Table, gtk.Toolbar, gtk.AspectFrame, gtk.Frame, gtk.VBox, gtk.HBox, gtk.VSeparator, gtk.HSeparator. These widgets can be added to a gtk.EventBox to overcome this limitation.

gtk.Widget.modify_text

```
    def modify_text(state, color)
```

state :	a widget state.
color :	the gtk.gdk.Color to assign.

The modify_text() method sets the text color to the gtk.gdk.Color specified by *color* for the widget in the specified *state*. All other style values are left untouched. The text color is the foreground color used along with the base color (see the modify_base() method) for widgets such as gtk.Entry and gtk.TextView. See the modify_fg() method for detail on the possible values of *state*.

gtk.Widget.modify_base

```
    def modify_base(state, color)
```

state :	a widget state.
color :	the gtk.gdk.Color to assign

The modify_base() method sets the base color to the gtk.gdk.Color specified by *color* for the widget in the specified *state*. All other style values are left untouched. The base color is the background color used along with the text color (see the modify_text() method) for widgets such as gtk.Entry and gtk.TextView. See modify_fg() method for detail on the possible values of *state*.

gtk.Widget.modify_font

```
    def modify_font(font_desc)
```

font_desc :	a font description to use

The modify_font() method sets the font to use to the value specified by *font_desc* for the widget. All other style values are left untouched.

gtk.Widget.create_pango_context

```
def create_pango_context()
```

Returns :	the new `pango.Context`

The `create_pango_context()` method creates a new `pango.Context` with the appropriate colormap, font description, and base direction for drawing text for this widget. See the `get_pango_context()` method.

gtk.Widget.get_pango_context

```
def get_pango_context()
```

Returns :	the `pango.Context` for the widget.

The `get_pango_context()` method returns the `pango.Context` with the appropriate colormap, font description and base direction for this widget. Unlike the context returned by the `create_pango_context()` method, this context is owned by the widget (it can be used as long as widget exists), and will be updated to match any changes to the widget's attributes.

If you create and keep a `pango.Layout` using this context, you must deal with changes to the context by calling the `pango.Layout.context_changed()` method on the layout in response to the "style–set" and "direction–set" signals for the widget.

gtk.Widget.create_pango_layout

```
def create_pango_layout(text)
```

`text` :	the text to set on the layout
Returns :	the new `pango.Layout`

The `create_pango_layout()` method creates a new `pango.Layout` with the appropriate colormap, font description, and base direction for drawing the specified *`text`* for this widget. If you keep a `pango.Layout` created by this method, you must call `pango.Layout.context_changed()` in response to the "style–set" and "direction–set" signals for the widget to notify the layout of changes to the base direction or font of this widget.

gtk.Widget.render_icon

```
def render_icon(stock_id, size, detail)
```

`stock_id` :	a stock ID
`size` :	a stock size
`detail` :	the render detail to pass to the theme engine
Returns :	a new pixbuf, or `None` if the stock ID wasn't known

The `render_icon()` method is a convenience method that uses the theme engine and RC file settings for the widget to look up the stock icon specified by *`stock_id`* of the specified *`size`* and to render it to a pixbuf that is returned. *`stock_id`* should be a stock icon ID such as `gtk.STOCK_OPEN` or `gtk.STOCK_OK`. *`size`* should be one of the GTK Icon Size Constants:

`detail` should be a string that identifies the widget or code doing the rendering, so that theme engines can special–case rendering for that widget or code.

gtk.Widget.set_composite_name

```
def set_composite_name(name)
```

name :	the name to set.

The `set_composite_name()` method sets a widgets composite name to the value specified by *`name`*. The widget must be a composite child of its parent

gtk.Widget.get_composite_name

```
def get_composite_name()
```

Returns :	the composite name of the widget or `None`

The `get_composite_name()` method returns the composite name of a widget or `None` if the widget is not a composite.

gtk.Widget.reset_rc_styles

```
def reset_rc_styles()
```

The `reset_rc_styles()` method resets the styles of widget and all descendants to the correct values for the currently loaded RC file settings. This method is not useful for applications.

gtk.Widget.style_get_property

```
def style_get_property(property_name)
```

property_name :	the name of a style property
Returns :	the property value

Note

This method is available in PyGTK 2.4 and above.

The `style_get_property()` method returns the value of a style property specified by *`property_name`*.

gtk.Widget.set_direction

```
def set_direction(dir)
```

dir :	the new direction

The `set_direction()` method sets the "direction" property to the value specified by *`dir`*. The "direction" property determines the reading direction of the widget that controls the primary direction for widgets containing text, and also the direction in which the children of a container are packed. The ability to set the direction is to handle localization for languages with right–to–left reading directions. Generally, applications will use the default reading direction, except for containers that are arranged in an order that is explicitly visual rather than logical (such as buttons for text justification). The values of *`dir`* must be one of the GTK Text Direction Constants.

If the direction is set to `gtk.TEXT_DIR_NONE`, then the value set by the `gtk.widget.set_default_direction()` function will be used.

gtk.Widget.get_direction

```
def get_direction()
```

Returns :	the reading direction for the widget.

The `get_direction()` method returns the reading direction for the widget. See the `set_direction()` method for more information.

gtk.Widget.shape_combine_mask

```
def shape_combine_mask(shape_mask, offset_x, offset_y)
```

`shape_mask` :	the shape to be added.
`offset_x` :	the X position of shape mask with respect to the widget's `gtk.gdk.Window`.
`offset_y` :	Y position of shape mask with respect to the widget's `gtk.gdk.Window`.

The `shape_combine_mask()` method sets a shape for the widget's `gtk.gdk.Window` using the mask specified by *`shape_mask`* at the location specified by *`offset_x`* and *`offset_y`*. This allows for transparent windows etc., see the `gtk.gdk.Window.shape_combine_mask()` method for more information.

gtk.Widget.reset_shapes

```
def reset_shapes()
```

The `reset_shapes()` method recursively resets the shapes of the widget and its descendants.

gtk.Widget.path

```
def path()
```

Returns :	the widget's path

The `path()` method returns the full path to the widget. The path is simply the name of a widget and all its parents in the container hierarchy, separated by periods. The name of a widget comes from the `get_name()` method. Paths are used to apply styles to a widget in gtkrc configuration files. Widget names are the type of the widget by default (e.g. "GtkButton") or can be set to an application−specific value with the `set_name()` method. By setting the name of a widget, you allow users or theme authors to apply styles to that specific widget in their gtkrc file.

gtk.Widget.class_path

```
def class_path()
```

Returns :	the widget's class path

The `class_path()` method is similar to the `path()` method, but does not use a custom name set with the `set_name()` (e.g. always uses "GtkButton" even if a custom name is available).

gtk.Widget.list_mnemonic_labels

```
def list_mnemonic_labels()
```

Returns :	the list of mnemonic labels

Note

This method is available in PyGTK 2.4 and above.

The `list_mnemonic_labels`() method returns a list of the widgets, normally labels, for which this widget is a the target of a mnemonic (see for example, the `gtk.Label.set_mnemonic_widget()` method).

gtk.Widget.add_mnemonic_label

```
    def add_mnemonic_label(label)
```

`label` :	a `gtk.Widget` that acts as a mnemonic label.

Note

This method is available in PyGTK 2.4 and above.

The `add_mnemonic_label`() method adds the widget specified by *`label`* to the list of mnemonic labels for the widget.(See the `list_mnemonic_labels()` method for more detail).

gtk.Widget.remove_mnemonic_label

```
    def remove_mnemonic_label(label)
```

`label` :	a `gtk.Widget` that was previously set as a mnemonic label.

Note

This method is available in PyGTK 2.4 and above.

The `remove_mnemonic_label`() method removes the widget specified by *`label`* from the list of mnemonic labels for the widget. (See the `list_mnemonic_labels()` method). *`label`* must have previously been added to the list with the `add_mnemonic_label()`.

gtk.Widget.menu_get_for_attach_widget

```
    def menu_get_for_attach_widget()
```

Returns :	a list of menus attached to this widget.

Note

This method is available in PyGTK 2.6 and above.

The `menu_get_for_attach_widget`() method returns a list of the menus that are attached to this widget.

Functions

gtk.widget_push_colormap

```
    def gtk.widget_push_colormap(cmap)
```

cmap :	a gtk.gdk.Colormap

The `gtk.widget_push_colormap()` function pushes the `gtk.gdk.Colormap` specified by *`cmap`* onto a global stack of colormaps. The topmost colormap on the stack will be used when creating widgets. Remove *`cmap`* with the `gtk.widget_pop_colormap()` function. There's little reason to use this function.

gtk.widget_push_composite_child

```
    def gtk.widget_push_composite_child()
```

The `gtk.widget_push_composite_child()` function creates all new widgets as composite children until the corresponding `gtk.widget_pop_composite_child()` function call. A composite child is a child that's an implementation detail of the container it's inside and should not be visible to people using the container. Composite children aren't treated differently (but see the `gtk.Container.foreach()` method vs. the `gtk.Container.forall()` method), but e.g. GUI builders might want to treat them in a different way.

gtk.widget_pop_composite_child

```
    def gtk.widget_pop_composite_child()
```

The `gtk.widget_pop_composite_child()` function cancels the effect of a previous call to the `gtk.widget_push_composite_child()` function.

gtk.widget_pop_colormap

```
    def gtk.widget_pop_colormap()
```

The `gtk.widget_pop_colormap()` function removes the `gtk.gdk.Colormap` on the top of the global stack of colormaps. This function reverses the effect of the `gtk.widget_push_colormap()` function.

gtk.widget_get_default_style

```
    def gtk.widget_get_default_style()
```

Returns :	the default gtk.Style

The `gtk.widget_get_default_style()` function returns the default `gtk.Style` used by all newly created widgets

gtk.widget_set_default_colormap

```
    def gtk.widget_set_default_colormap(colormap)
```

colormap :	a gtk.gdk.Colormap object

The `gtk.widget_set_default_colormap()` function sets the default `gtk.gdk.Colormap` to use when creating widgets to the value specified by *`colormap`*. The `gtk.widget_push_colormap()` function is a better function to use if you only want to affect a few widgets, rather than all widgets.

gtk.widget_get_default_colormap

```
def gtk.widget_get_default_colormap()
```

Returns :	the default `gtk.gdk.Colormap` object

The `gtk.widget_get_default_colormap()` function returns the default `gtk.gdk.Colormap` used when creating new widgets.

gtk.widget_get_default_visual

```
def gtk.widget_get_default_visual()
```

Returns :	the default `gtk.gdk.Visual` object

The `gtk.widget_get_default_visual()` function returns the default `gtk.gdk.Visual` of the default `gtk.gdk.Colormap`.

gtk.widget_set_default_direction

```
def gtk.widget_set_default_direction(dir)
```

`dir` :	the new default direction – either `gtk.TEXT_DIR_RTL` or `gtk.TEXT_DIR_LTR`.

The `gtk.widget_set_default_direction()` function sets the default text direction to the value specified by *`dir`*. The value of *`dir`* must be either `gtk.TEXT_DIR_RTL` or `gtk.TEXT_DIR_LTR`. The default text direction is used for widgets that have not had a text direction set by the `set_direction()` method.

gtk.widget_get_default_direction

```
def gtk.widget_get_default_direction()
```

Returns :	the default text direction

The `gtk.widget_get_default_direction()` function returns the default text direction as set by the `gtk.widget_set_default_direction()` function.

gtk.widget_list_style_properties

```
def gtk.widget_list_style_properties(widget)
```

`widget` :	a `gtk.Widget`
Returns :	a list of style properties as GParam objects

Note

This function is available in PyGTK 2.4 and above.

The `gtk.widget_list_style_properties()` function returns a list of the style properties of the `gtk.Widget` specified by *`widget`*. The list contains a GParam object for each style property.

gtk.widget_class_install_style_property

```
def gtk.widget_class_install_style_property(widget)
```

`widget` :	a `gtk.Widget`

pspec :	a 4–tuple containing the property spec

Note

This function is available in PyGTK 2.4 and above.

The `gtk.widget_class_install_style_property()` function installs the style property specified by *`pspec`* on the `gtk.Widget` specified by *`widget`*. *`pspec`* is a 4–tuple containing the property name, the property type, a nickname (or `None`) and a description of the property (or None).

This function raises the TypeError exception if *`widget`* is not a `gtk.Widget` or if the property is already installed.

Signals

The "accel–closures–changed" gtk.Widget Signal

```
    def callback(widget, user_param1, ...)
```

`widget` :	the widget that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "accel–closures–changed" signal is emitted when an accelerator is added to or removed from the `gtk.AccelGroup` for *`widget`* or an accelerator path is setup.

The "button–press–event" gtk.Widget Signal

```
    def callback(widget, event, user_param1, ...)
```

`widget` :	the widget that received the signal
`event` :	the event that triggered the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)
Returns :	`TRUE` to stop other handlers from being invoked for the event. `FALSE` to propagate the event further.

The "button–press–event" signal is emitted when a mouse button is pressed.

The "button–release–event" gtk.Widget Signal

```
    def callback(widget, event, user_param1, ...)
```

`widget` :	the widget that received the signal
`event` :	the event that triggered the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)
Returns :	`TRUE` to stop other handlers from being invoked for the event. `FALSE` to propagate the event further.

The "button–release–event" signal is emitted when a mouse button is released.

The "can−activate−accel" gtk.Widget Signal

```
def callback(widget, signal_id, user_param1, ...)
```

widget :	the widget that received the signal
signal_id :	the ID of a signal installed on *widget*
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

Note

This signal is available in GTK+ 2.4 and above.

The "can−activate−accel" signal is emitted when an accelerator is about to activate *widget*. The handler determines if the accelerator that activates the signal identified by *signal_id* can currently be activated. This signal is present to allow applications and derived widgets to override the default GtkWidget handling for determining whether an accelerator can be activated.

The "child−notify" gtk.Widget Signal

```
def callback(widget, child_property, user_param1, ...)
```

widget :	the widget that received the signal
child_property :	the name of a child property
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "child−notify" signal is emitted when *child_property* is changed.

Child properties are available with objects derived from gtk.Container. Those properties are not specific to either the container or the child widget but to their relation. For example, the "pack−type" property of gtk.Box or the "menu−label" property of gtk.Notebook are child properties.

The "client−event" gtk.Widget Signal

```
def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "client−event" signal is emitted when another application has sent an event to *widget*.

The "configure−event" gtk.Widget Signal

```
def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal

user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "configure–event" signal is emitted when the widget's window is allocated a size and width.

The "delete–event" gtk.Widget Signal

```
    def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "delete–event" signal is emitted when a request is made to delete *widget*.

The "destroy–event" gtk.Widget Signal

```
    def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "destroy–event" signal is emitted when a gtk.gdk.Window is destroyed. You rarely get this signal, because most widgets disconnect themselves from their window before they destroy it, so no widget owns the window at destroy time.

The "direction–changed" gtk.Widget Signal

```
    def callback(widget, direction, user_param1, ...)
```

widget :	the widget that received the signal
direction :	the previous direction: gtk.TEXT_DIR_NONE, gtk.TEXT_DIR_LTR or gtk.TEXT_DIR_RTL
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "direction–changed" signal is emitted when the reading direction of *widget* is changed (usually with the set_direction() method)

The "drag–begin" gtk.Widget Signal

```
    def callback(widget, drag_context, user_param1, ...)
```

widget :	the widget that received the signal
drag_context :	the gtk.gdk.DragContext

user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "drag–begin" signal is emitted when the user initiates a drag operation on *widget*. A typical reason to connect to this signal is to set up a custom drag icon with the drag_source_set_icon() method.

The "drag–data–delete" gtk.Widget Signal

```
    def callback(widget, drag_context, user_param1, ...)
```

widget :	the widget that received the signal
drag_context :	the gtk.gdk.DragContext
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "drag–data–delete" signal is emitted when the drag completes a move operation and requires the source data to be deleted. The signal handler is responsible for deleting the data that has been dropped.

The "drag–data–get" gtk.Widget Signal

```
    def callback(widget, drag_context, selection_data, info, timestamp, user_param1, ...)
```

widget :	the widget that received the signal
drag_context :	the gtk.gdk.DragContext
selection_data :	a gtk.SelectionData object
info :	an integer ID for the drag
timestamp :	the time of the drag event
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "drag–data–get" signal is emitted when a drag operation completes that copies data or when a drag drop occurs using the gtk.gdk.DRAG_PROTO_ROOTWIN protocol. The drag source rev=ceives this signal when the drag destination requests the data using the drag_get_data() method. The handler needs to fill *selection_data* with the data in the format specified by the target associated with *info*.

The "drag–data–received" gtk.Widget Signal

```
    def callback(widget, drag_context, x, y, selection_data, info, timestamp, user_param1, ...)
```

widget :	the widget that received the signal
drag_context :	the gtk.gdk.DragContext
x :	the X position of the drop
y :	the Y position of the drop
selection_data :	a gtk.SelectionData object
info :	an integer ID for the drag
timestamp :	the time of the drag event
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "drag–data–received" signal is emitted when the drag destination receives the data from the drag operation. If the data was received in order to determine whether the drop will be accepted, the handler is expected to call the gtk.gdk.DragContext.drag_status() method and not finish the drag. If the data was received in response to a "drag–drop" signal (and this is the last target to be received), the handler for this signal is expected to process the received data and then call the `gtk.gdk.DragContext.finish()` method, setting the *`success`* parameter to `TRUE` if the data was processed successfully.

The "drag−drop" gtk.Widget Signal

```
    def callback(widget, drag_context, x, y, timestamp, user_param1, ...)
```

`widget` :	the widget that received the signal
`drag_context` :	the `gtk.gdk.DragContext`
`x` :	the X position of the drop
`y` :	the Y position of the drop
`timestamp` :	the time of the drag event
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)
Returns :	`TRUE` if the cursor is in a drop zone

The "drag−drop" signal is emitted when the drag initiates a drop operation on the destination *`widget`*. The signal handler must determine whether the cursor position is in a drop zone or not. If it is not in a drop zone, it returns `FALSE` and no further processing is necessary. Otherwise, the handler returns `TRUE`. In this case, the handler must ensure that the `gtk.gdk.DragContext.finish()` method is called to let the source know that the drop is done. The call to the `gtk.gdk.DragContext.finish()` method can be done either directly or in a "drag−data−received" handler that gets triggered by calling the `drag_get_data()` method to receive the data for one or more of the supported targets.

The "drag−end" gtk.Widget Signal

```
    def callback(widget, drag_context, user_param1, ...)
```

`widget` :	the widget that received the signal
`drag_context` :	the `gtk.gdk.DragContext`
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "drag−end" signal is emitted when the drag operation is completed. A typical reason to connect to this signal is to undo things done in a "drag−begin" handler.

The "drag−leave" gtk.Widget Signal

```
    def callback(widget, drag_context, timestamp, user_param1, ...)
```

`widget` :	the widget that received the signal
`drag_context` :	the `gtk.gdk.DragContext`
`timestamp` :	the time of the drag event
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "drag−leave" signal is emitted when the drag operation moves off of a drop target widget. A typical reason to connect to this signal is to undo things done in a "drag−motion" handler, e.g. undo highlighting with the `drag_unhighlight()` method.

The "drag−motion" gtk.Widget Signal

```
def callback(widget, drag_context, x, y, timestamp, user_param1, ...)
```

widget :	the widget that received the signal
drag_context :	the gtk.gdk.DragContext
x :	the X position of the drop
y :	the Y position of the drop
timestamp :	the time of the drag event
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	TRUE if the cursor is in a drop zone

The "drag−motion" signal is emitted when the drag operation moves over a drop target widget. The signal handler must determine if the cursor position is in a drop zone or not. If it is not in a drop zone, it should return FALSE and no further processing is necessary. Otherwise, the handler should return TRUE. In this case, the handler is responsible for providing the necessary information for displaying feedback to the user, by calling the gtk.gdk.DragContext.drag_status() method. If the decision to accept or reject the drop can't be made based solely on the cursor position and the type of the data, the handler may inspect the dragged data by calling the drag_get_data() method and defer the gtk.gdk.DragContext.drag_status() method call to the "drag−data−received" handler.

Note

There is no "drag−enter" signal. The drag receiver has to keep track of any "drag−motion" signals received since the last "drag−leave" signal. The first "drag−motion" signal received after a "drag_leave" signal should be treated as an "enter" signal. Upon an "enter", the handler will typically highlight the drop site with the drag_highlight() method.

The "enter−notify−event" gtk.Widget Signal

```
def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further

The "enter−notify−event" signal is emitted when the mouse pointer enters *widget*.

The "event" gtk.Widget Signal

```
def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further

The "event" signal is emitted when any `gtk.gdk.Event` occurs on *widget*. The "event" signal is emitted before any of the specific `gtk.gdk.Event` signals are emitted.

The "event−after" gtk.Widget Signal

```
def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "event−after" signal is emitted after any other event handling has occurred for *widget*

The "expose−event" gtk.Widget Signal

```
def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)
Returns :	`TRUE` to stop other handlers from being invoked for the event. `FALSE` to propagate the event further

The "expose−event" signal is emitted when *widget* needs to be repainted because it is first displayed or has been partially or fully obscured by another window.

The "focus" gtk.Widget Signal

```
def callback(widget, direction, user_param1, ...)
```

widget :	the widget that received the signal
direction :	the direction: `gtk.DIR_TAB_FORWARD`, `gtk.DIR_TAB_BACKWARD`, `gtk.DIR_UP`, `gtk.DIR_DOWN`, `gtk.DIR_LEFT` or `gtk.DIR_RIGHT`
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)
Returns :	`TRUE` to stop other handlers from being invoked for the event. `FALSE` to propagate the event further

The "focus" signal is emitted when *widget* receives the focus.

The "focus−in−event" gtk.Widget Signal

```
def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)
Returns :	`TRUE` to stop other handlers from being invoked for the event. `FALSE` to propagate the event further

The "focus–in–event" signal is emitted when the focus changes to `widget`.

The "focus–out–event" gtk.Widget Signal

```
    def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	`TRUE` to stop other handlers from being invoked for the event. `FALSE` to propagate the event further

The "focus–out–event" signal is emitted when the focus leaves `widget`.

The "grab–focus" gtk.Widget Signal

```
    def callback(widget, user_param1, ...)
```

widget :	the widget that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "grab–focus" signal is emitted when `widget` grabs the focus usually by calling the grab_focus() method or by the user using a mnemonic accelerator..

The "grab–notify" gtk.Widget Signal

```
    def callback(widget, was_grabbed, user_param1, ...)
```

widget :	the widget that received the signal
was_grabbed :	if `TRUE` *widget* had grabbed the focus
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "grab–notify" signal is emitted when widget (or its ancestor) either is grabbing the focus or has the focus grabbed from it.

The "hide" gtk.Widget Signal

```
    def callback(widget, user_param1, ...)
```

widget :	the widget that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "hide" signal is emitted when `widget` is hidden usually by calling the hide() method.

The "hierarchy–changed" gtk.Widget Signal

```
    def callback(widget, previous_toplevel, user_param1, ...)
```

widget :	the widget that received the signal
previous_toplevel :	the toplevel widget in the previous hierarchy containing *widget*

user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "hierarchy−changed" signal is emitted when *widget* is unparented or has a new parent set.

The "key−press−event" gtk.Widget Signal

```
    def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	`TRUE` to stop other handlers from being invoked for the event. `FALSE` to propagate the event further

The "key−press−event" signal is emitted when the user presses a key on the keyboard.

The "key−release−event" gtk.Widget Signal

```
    def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	`TRUE` to stop other handlers from being invoked for the event. `FALSE` to propagate the event further

The "key−release−event" signal is emitted when the user releases a key on the keyboard.

The "leave−notify−event" gtk.Widget Signal

```
    def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	`TRUE` to stop other handlers from being invoked for the event. `FALSE` to propagate the event further

The "leave−notify−event" signal is emitted when the mouse pointer leaves the area of *widget*.

The "map" gtk.Widget Signal

```
    def callback(widget, user_param1, ...)
```

widget :	the widget that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "map" signal is emitted when *widget* requests to be mapped onto the display usually by calling the show() or map() methods.

The "map−event" gtk.Widget Signal

```
    def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further

The "map−event" signal is emitted when *widget* has been mapped onto the display.

The "mnemonic−activate" gtk.Widget Signal

```
    def callback(widget, group_cycling, user_param1, ...)
```

widget :	the widget that received the signal
group_cycling :	if TRUE shifts the focus instead of activating *widget*
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further

The "mnemonic−activate" signal is emitted when the user uses a mnemonic accelerator to activate *widget*.

The "motion−notify−event" gtk.Widget Signal

```
    def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further

The "motion−notify−event" signal is emitted when the mouse pointer moves while over *widget*.

The "no−expose−event" gtk.Widget Signal

```
    def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further

The "no−expose−event" signal is emitted when a no expose event occurs.

The "parent–set" gtk.Widget Signal

```
    def callback(widget, old_parent, user_param1, ...)
```

`widget` :	the widget that received the signal
`old_parent` :	the previous parent of `widget`
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "parent–set" signal is emitted when the parent of `widget` is set or unset.

The "popup–menu" gtk.Widget Signal

```
    def callback(widget, user_param1, ...)
```

`widget` :	the widget that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)
Returns :	`TRUE` to stop other handlers from being invoked for the event. `FALSE` to propagate the event further
Returns :	`TRUE` if a menu was activated

The "popup–menu" signal is emitted when the user presses the **Shift+F10** or **Menu** keys when `widget` has the focus to popup a menu.

The "property–notify–event" gtk.Widget Signal

```
    def callback(widget, event, user_param1, ...)
```

`widget` :	the widget that received the signal
`event` :	the event that triggered the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)
Returns :	`TRUE` to stop other handlers from being invoked for the event. `FALSE` to propagate the event further.

The "property–notify–event" signal is emitted when a window property value has changed. This is used for selection data retrieval.

The "proximity–in–event" gtk.Widget Signal

```
    def callback(widget, event, user_param1, ...)
```

`widget` :	the widget that received the signal
`event` :	the event that triggered the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)
Returns :	`TRUE` to stop other handlers from being invoked for the event. `FALSE` to propagate the event further.

The "proximity–in–event" (available for devices like touch screens or graphics tablets) is emitted when the pen touches the tablet or when the user's finger touches the screen.

The "proximity−out−event" gtk.Widget Signal

```
def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)
Returns :	`TRUE` to stop other handlers from being invoked for the event. `FALSE` to propagate the event further.

The "proximity−out−event" (available for devices like touch screens or graphics tablets) is emitted when the pen leaves the tablet or when the user's finger leaves the screen.

The "realize" gtk.Widget Signal

```
def callback(widget, user_param1, ...)
```

widget :	the widget that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

The "realize" signal is emitted when *widget* requests to be realized by calling the `realize()` method.

The "screen−changed" gtk.Widget Signal

```
def callback(widget, screen, user_param1, ...)
```

widget :	the widget that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)

Note

This signal is available in GTK+ 2.4 and above.

The "screen−changed" signal is emitted when *screen* becomes the new `gtk.gdk.Screen` for *widget*.

The "scroll−event" gtk.Widget Signal

```
def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
... :	additional user parameters (if any)
Returns :	`TRUE` to stop other handlers from being invoked for the event. `FALSE` to propagate the event further.

The "scroll−event" signal is emitted when widget receives a scroll event.

The "selection−clear−event" gtk.Widget Signal

```
    def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "selection−clear−event" signal is emitted when the selection needs to be cleared.

The "selection−get" gtk.Widget Signal

```
    def callback(widget, selection_data, info, timestamp, user_param1, ...)
```

widget :	the widget that received the signal
selection_data :	a gtk.SelectionData object
info :	an integer ID for the selection
timestamp :	the time the event occurred
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "selection−get" signal is emitted when the selection data is requested from *widget*.

The "selection−notify−event" gtk.Widget Signal

```
    def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "selection−notify−event" signal is emitted when the selection owner has responded to the selection conversion request.

The "selection−received" gtk.Widget Signal

```
    def callback(widget, selection_data, timestamp, user_param1, ...)
```

widget :	the widget that received the signal
selection_data :	a gtk.SelectionData object
timestamp :	the time the event occurred
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "selection−received" signal is emitted when the selection owner has responded to the request for the selection data.

The "selection−request−event" gtk.Widget Signal

```
    def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	`TRUE` to stop other handlers from being invoked for the event. `FALSE` to propagate the event further.

The "selection−request−event" signal is emitted when a selection request is made on *widget*.

The "show" gtk.Widget Signal

```
    def callback(widget, user_param1, ...)
```

widget :	the widget that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "show" signal is emitted when *widget* requests to be displayed using either the show() or show_all() method.

The "show−help" gtk.Widget Signal

```
    def callback(widget, help_type, user_param1, ...)
```

widget :	the widget that received the signal
help_type :	the help type; either `gtk.WIDGET_HELP_TOOLTIP` or `gtk.WIDGET_HELP_WHATS_THIS`
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	`TRUE` to stop other handlers from being invoked.

The "show−help" signal is emitted when the user presses the **Control+F1** key combination.

The "size−allocate" gtk.Widget Signal

```
    def callback(widget, allocation, user_param1, ...)
```

widget :	the widget that received the signal
allocation :	the widget's space allocation in a gtk.gdk.Rectangle
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "size−allocate" signal is emitted when widget is given a new space allocation.

The "size−request" gtk.Widget Signal

```
    def callback(widget, requisition, user_param1, ...)
```

widget :	the widget that received the signal
requisition :	the widget's requested size as a gtk.Requisition

user_param1 :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "size−request" signal is emitted when a new size is requested for *widget* using the `set_size_request()` method.

The "state−changed" gtk.Widget Signal

```
def callback(widget, state, user_param1, ...)
```

widget :	the widget that received the signal
state :	the previous widget state: `gtk.STATE_NORMAL`, `gtk.STATE_ACTIVE`, `gtk.STATE_PRELIGHT`, `gtk.STATE_SELECTED` or `gtk.STATE_INSENSITIVE`
user_param1 :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "state−changed" signal is emitted when the state of *widget* changes.

The "style−set" gtk.Widget Signal

```
def callback(widget, previous_style, user_param1, ...)
```

widget :	the widget that received the signal
previous_style :	the previous widget `gtk.Style`
user_param1 :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "style−set" signal is emitted when the `gtk.Style` of *widget* is set.

The "unmap" gtk.Widget Signal

```
def callback(widget, user_param1, ...)
```

widget :	the widget that received the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "unmap" signal is emitted when *widget* requests to be unmapped from the display.

The "unmap−event" gtk.Widget Signal

```
def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)
Returns :	`TRUE` to stop other handlers from being invoked for the event. `FALSE` to propagate the event further.

The "unmap−event" signal is emitted when *widget* has been unmapped from the display.

The "unrealize" gtk.Widget Signal

```
    def callback(widget, user_param1, ...)
```

widget :	the widget that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "unrealize" signal is emitted when *widget* requests to be unrealized (i.e. have all its resources released).

The "visibility–notify–event" gtk.Widget Signal

```
    def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "visibility–notify–event" signal is emitted when the visibility of *widget* changes i.e. it has been obscured or unobscured.

The "window–state–event" gtk.Widget Signal

```
    def callback(widget, event, user_param1, ...)
```

widget :	the widget that received the signal
event :	the event that triggered the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)
Returns :	TRUE to stop other handlers from being invoked for the event. FALSE to propagate the event further.

The "window–state–event" signal is emitted when window state of widget changes. For example, for a toplevel window this event is signaled when the window is iconified, deiconified, minimized, maximized, made sticky, made not sticky, shaded or unshaded.

Prev	Up	Next
gtk.Viewport	Home	gtk.Window

gtk.Window

Prev **The gtk Class Reference** Next

gtk.Window

gtk.Window a top–level window that holds one child widget.

Synopsis

```
class gtk.Window(gtk.Bin):
    gtk.Window(type=gtk.WINDOW_TOPLEVEL)
    def set_title(title)
    def get_title()
    def set_wmclass(wmclass_name, wmclass_class)
    def set_role(role)
    def get_role()
    def add_accel_group(accel_group)
    def remove_accel_group(accel_group)
    def set_position(position)
    def activate_focus()
    def set_focus(focus)
    def get_focus()
    def set_default(default_widget)
    def activate_default()
    def set_transient_for(parent)
    def get_transient_for()
    def set_type_hint(hint)
    def get_type_hint()
    def set_destroy_with_parent(setting)
    def get_destroy_with_parent()
    def set_resizable(resizable)
    def get_resizable()
    def set_gravity(gravity)
    def get_gravity()
    def set_geometry_hints(geometry_widget, min_width=-1, min_height=-1, max_width=-1, max_heig
    def set_screen(screen)
    def get_screen()
    def is_active()
    def has_toplevel_focus()
    def set_has_frame(setting)
    def get_has_frame()
    def set_frame_dimensions(left, top, right, bottom)
    def get_frame_dimensions()
    def set_decorated(setting)
    def get_decorated()
    def set_icon_list(...)
    def get_icon_list()
    def set_icon(icon)
    def set_icon_from_file(filename)
    def get_icon()
    def set_modal(modal)
    def get_modal()
    def add_mnemonic(keyval, target)
    def remove_mnemonic(keyval, target)
    def mnemonic_activate(keyval, modifier)
    def set_mnemonic_modifier(modifier)
    def get_mnemonic_modifier()
    def activate_key(event)
    def propagate_key_event(event)
    def present()
    def iconify()
    def deiconify()
    def stick()
    def unstick()
    def maximize()
    def unmaximize()
    def fullscreen()
    def unfullscreen()
    def set_keep_above(setting)
    def set_keep_below(setting)
    def begin_resize_drag(edge, button, root_x, root_y, timestamp)
    def begin_move_drag(button, root_x, root_y, timestamp)
    def set_default_size(width, height)
    def get_default_size()
    def resize(width, height)
    def get_size()
    def move(x, y)
    def get_position()
    def parse_geometry(geometry)
    def reshow_with_initial_size()
    def tooltips_get_info_from_tip_window()
    def set_focus_on_map(setting)
    def get_focus_on_map()
    def set_icon_name(name)
    def get_icon_name()
Functions

    def gtk.window_set_default_icon(icon)
    def gtk.window_set_default_icon_from_file(filename)
    def gtk.window_set_default_icon_list(...)
    def gtk.window_get_default_icon_list()
    def gtk.window_set_auto_startup_notification(setting)
    def gtk.window_list_toplevels()
    def gtk.window_set_default_icon_name(name)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Window
```

Properties

"accept–focus"	Read–Write	If `TRUE`, the window should receive the input focus. Default value: `TRUE`. Available in GTK+ 2.4 and above.
"allow–grow"	Read–Write	If `TRUE`, the user can expand the window beyond its minimum size. Default value: `TRUE`.
"allow–shrink"	Read–Write	If `TRUE`, the window has no minimum size. Setting this to `TRUE` is a bad idea 99% of the time. Default value: `FALSE`.
"decorated"	Read–Write	If `TRUE`, the window should be decorated by the window manager. Default value: `TRUE`. Available in GTK+ 2.4 and above.
"default–height"	Read–Write	The default height of the window, used when initially showing the window. Allowed values: >= –1. Default value: –1
"default–width"	Read–Write	The default width of the window, used when initially showing the window. Allowed values: >= –1. Default value: –1
"destroy–with–parent"	Read–Write	If `TRUE`, the window should be destroyed when its parent is destroyed. Default value: `FALSE`.
"focus–on–map"	Read–Write	If `TRUE`, the window should receive the input focus when mapped. Default value: `TRUE`. Available in GTK+ 2.6 and above.
"gravity"	Read–Write	The window gravity of the window. See the `move()` method and the GDK Gravity Constants for more details about window gravity. Default value:

		`gtk.gdk.GRAVITY_NORTH_WEST`. Available in GTK+ 2.4 and above.
"has–toplevel–focus"	Read	If `TRUE`, the input focus is within the window. Default value: `FALSE`. Available in GTK+ 2.2 and above.
"icon"	Read–Write	The icon for this window
"icon–name"	Read–Write	The name of the themed icon to use as the window icon. See `gtk.IconTheme` for more details. Default value: `None`. Available in GTK+ 2.6 and above.
"is–active"	Read	If `TRUE`, the toplevel is the current active window. Default value: `FALSE`. Available in GTK+ 2.2 and above.
"modal"	Read–Write	If `TRUE`, the window is modal (other windows are not usable while this one is up). Default value: `FALSE`.
"resizable"	Read–Write	If `TRUE`, the user can resize the window. Default value: `TRUE`.
"role"	Read–Write	Unique identifier for the window to be used when restoring a session. Default value: `None`. Available in GTK+ 2.4
"screen"	Read–Write	The screen where this window will be displayed. Available in GTK+ 2.2
"skip–pager–hint"	Read–Write	If `TRUE`, the window should not be in the pager. Default value: `FALSE`. Available in GTK+ 2.2 and above.
"skip–taskbar–hint"	Read–Write	If `TRUE`, the window should not be in the task bar. Default value: `FALSE`. Available in GTK+ 2.2 and above.
"title"	Read–Write	The title of the window. Default value: `None`.
"type"	Read–Write	The type of the window. Default value: `gtk.WINDOW_TOPLEVEL`
"type–hint"	Read–Write	Hint to help the desktop environment understand what kind of window this is and how to treat it. Default value: `gtk.gdk.WINDOW_TYPE_HINT_NORMAL`. Available in GTK+ 2.2 and above.
"window–position"	Read–Write	The initial position of the window. Default value: `gtk.WIN_POS_NONE`

Attributes

"allow_grow"	Read	If `TRUE`, users can expand the window beyond its minimum size.
"allow_shrink"	Read	If `TRUE`, the window has no minimum size.
"configure_notify_received"	Read	If `TRUE` a window resize configuration event has been received.
"configure_request_count"	Read	The number of outstanding configuration requests.
"decorated"	Read	If `TRUE` the window will have decorations like a titlebar, resize controls, etc. See the `set_decorated()` method.
"default_widget"	Read	The child widget that will be activated by default. See the `set_default()` method.

"destroy_with_parent"	Read	If TRUE the window is destroyed when its transient parent is destroyed.
"focus_widget"	Read	The child widget that has the focus.
"frame"	Read	The frame gtk.gdk.Window (if any). See the set_has_frame() and set_frame_dimensions() methods.
"frame_bottom"	Read	The height of the bottom frame border. See the set_frame_dimensions() and set_has_frame() methods.
"frame_left"	Read	The width of the left frame border. See the set_frame_dimensions() and set_has_frame() methods.
"frame_right"	Read	The width of the right frame border. See the set_frame_dimensions() and set_has_frame() methods.
"frame_top"	Read	The height of the top frame border. See the set_frame_dimensions() and set_has_frame() methods.
"gravity"	Read	The window's gravity. See the move() and set_gravity() methods.
"group"	Read	The gtk.WindowGroup (if any) the window belongs to.
"has_focus"	Read	If TRUE the window has the focus.
"has_frame"	Read	If TRUE the window has a frame window. See the set_has_frame() method.
"has_user_ref_count"	Read	If TRUE the window has not been destroyed.
"iconify_initially"	Read	If TRUE the window has been iconified by a call to the iconify() method.
"keys_changed_handler"	Read	The idle handler ID for handling accelerator group changes.
"maximize_initially"	Read	If TRUE the window has been maximized by a call to the maximize() method.
"mnemonic_modifier"	Read	The mnemonic modifier used with a key to activate an accelerator. See the set_mnemonic_modifier() method
"modal"	Read	If TRUE the window is modal. See the set_modal() method.
"need_default_position"	Read	If TRUE the window needs an initial position calculated.
"need_default_size"	Read	If TRUE the window needs an initial size calculated.
"position"	Read	The initial position of the window. See the set_position() method.
"stick_initially"	Read	If TRUE the window has been made sticky by a call to the stick() method.
"title"	Read	The title of the window.
"transient_parent"	Read	The transient parent window. See the set_transient_for() method.
"type"	Read	The type of the window: gtk.WINDOW_TOPLEVEL or gtk.WINDOW_POPUP.

"type_hint"	Read	The window's type hint. See the `set_type_hint()` method.
"wmclass_class"	Read	The window system class hint. See the `set_wmclass()` method.
"wmclass_name"	Read	The window system name hint. See the `set_wmclass()` method.
"wm_role"	Read	The unique identifier for the window. See the `set_role()` method.

Signal Prototypes

"activate–default"	`def callback(`*`window`*`, `*`user_param1`*`, `*`...`*`)`
"activate–focus"	`def callback(`*`window`*`, `*`user_param1`*`, `*`...`*`)`
"frame–event"	`def callback(`*`window`*`, `*`event`*`, `*`user_param1`*`, `*`...`*`)`
"keys–changed"	`def callback(`*`window`*`, `*`user_param1`*`, `*`...`*`)`
"move–focus"	`def callback(`*`window`*`, `*`direction`*`, `*`user_param1`*`, `*`...`*`)`
"set–focus"	`def callback(`*`window`*`, `*`widget`*`, `*`user_param1`*`, `*`...`*`)`

Description

A `gtk.Window` provides a widget that users commonly think of as a window. That is, an area of the display that is managed by the window manager and usually decorated with a title bar, and items to allow the user to close, resize and move the window. `PyGTK` provides two types of windows (see the GTK Window Type Constants section for more information):

`gtk.WINDOW_TOPLEVEL`	A window that has no parent and usually has a frame and decorations supplied by a window manager. Toplevels windows are the main application window and dialogs.
`gtk.WINDOW_POPUP`	A window that is ignored by the window manager and has no frame or decorations. A popup window is used for menus and tooltips.

Typically, applications only directly create and use toplevel windows.

A `gtk.Window` is a container (subclass of `gtk.Bin`) holding one child widget.

Constructor

```
gtk.Window(type=gtk.WINDOW_TOPLEVEL)
```

type :	the type of window: `gtk.WINDOW_TOPLEVEL` or `gtk.WINDOW_POPUP`
Returns :	a new `gtk.Window`.

Creates a new `gtk.Window`, which is a toplevel window that can contain other widgets. Nearly always, the type of the window should be `gtk.WINDOW_TOPLEVEL` (see the GTK Window Type Constants for more details). `gtk.WINDOW_POPUP` is used to create a pop–up menu or pop–up tooltip. On X11, popup windows are not controlled by the window manager. If you simply want an undecorated window (no window borders), use the `set_decorated()` method with a toplevel window, don't use a popup window.

Methods

gtk.Window.set_title

```
def set_title(title)
```

`title` :	the title of the window

The `set_title()` method sets the "title" property of the `gtk.Window` to the value specified by *`title`*. The title of a window will be displayed in its title bar. On the X Window System, the title bar is rendered by the window manager, so exactly how the title appears to users may vary according to a user's exact configuration. The title should help a user distinguish this window from other windows they may have open. A good title might include the application name and current document filename.

gtk.Window.get_title

```
def get_title()
```

Returns :	the title of the window, or `None`.

The `get_title()` method returns the value of the "title" property of the window. See the `set_title()` method.

gtk.Window.set_wmclass

```
def set_wmclass(wmclass_name, wmclass_class)
```

`wmclass_name` :	the window name hint
`wmclass_class` :	the window class hint

The `set_wmclass()` method sets the X Window System "class" and "name" hints for a window. Applications should not call this method. According to the ICCCM, you should always set these to the same value for all windows in an application, and `PyGTK` sets them to that value by default, so calling this method is sort of pointless. However, you may want to call the `set_role()` method on each window in your application, for the benefit of the session manager. Setting the role allows the window manager to restore window positions when loading a saved session.

gtk.Window.set_role

```
def set_role(role)
```

`role` :	a unique identifier for the window

The `set_role()` method sets a unique identifier (specified by *`role`*) for the window to be used when restoring a session. This method is only useful on X11. In combination with the window title, the window role allows a window manager to identify "the same" window when an application is restarted. So for example you might set the "toolbox" role on your app's toolbox window, so that when the user restarts their session, the window manager can put the toolbox back in the same place. If a window already has a unique title, you don't need to set the role, since the WM can use the title to identify the window when restoring the session.d

gtk.Window.get_role

```
def get_role()
```

Returns :	the role of the window if set, or `None`.

The `get_role()` method returns the role of the window. See the `set_role()` method for further explanation.

gtk.Window.add_accel_group

```
    def add_accel_group(accel_group)
```

`accel_group` :	a `gtk.AccelGroup`

The `add_accel_group()` method associates the accelerator group specified by *`accel_group`* with the window.

gtk.Window.remove_accel_group

```
    def remove_accel_group(accel_group)
```

`accel_group` :	a `gtk.AccelGroup`

The `remove_accel_group()` method dissociates the accelerator group specified by *`accel_group`* from the widget. This method reverses the effects of the `add_accel_group()` method.

gtk.Window.set_position

```
    def set_position(position)
```

`position` :	a position constraint.

The `set_position()` method sets the "window–position" property to the value of *`position`*. The value of *`position`* must be one of the GTK Window Position Constants.

If the old or new window position constraint is `gtk.WIN_POS_CENTER_ALWAYS`, this will also cause the window to be repositioned to satisfy the new constraint.

gtk.Window.activate_focus

```
    def activate_focus()
```

Returns :	`TRUE` if the window has a focus widget

The `activate_focus()` method activates the child widget with the focus. This method returns `TRUE` if the window has a widget with the focus.

gtk.Window.set_focus

```
    def set_focus(focus)
```

`focus` :	the widget to be the new focus widget or `None` to unset a focus widget

The `set_focus()` method sets the widget specified by *`focus`* to be the focus widget for the window. If *`focus`* is `None` the window's focus widget is unset. To set the focus to a particular widget in the toplevel, it is usually more convenient to use the `gtk.Widget.grab_focus()` method instead of this method.

gtk.Window.get_focus

```
    def get_focus()
```

Returns :	the currently focused widget.

The `get_focus()` method returns the current focused widget within the window. The focus widget is the widget that would have the focus if the toplevel window is focused.

gtk.Window.set_default

```
def set_default(default_widget)
```

`default_widget` :	the widget to be the default, or `None` to unset the default widget.

The `set_default()` method sets the window's default widget to the value specified by *`default_widget`*. If *`default_widget`* is `None` the window's default widget is unset. The default widget is the widget that's activated when the user presses **Enter** in a window. When setting (rather than unsetting) the default widget it's generally easier to call the `gtk.Widget.grab_default()` method on the widget. Before making a widget the default widget, you must set the `gtk.CAN_DEFAULT` flag on the widget you'd like to make the default using the `gtk.Object.set_flags()` method.

gtk.Window.activate_default

```
def activate_default()
```

Returns :	`TRUE` if the window has a default widget or a focus widget.

The `activate_default()` method activates the default widget. If there is no default widget or the default widget cannot be activated, the window's focus widget (if any) is activated. This method returns `FALSE` if no default widget could be activated or there is no focus widget.

gtk.Window.set_transient_for

```
def set_transient_for(parent)
```

`parent` :	the parent window or `None` to remove the transient parent

The `set_transient_for()` method sets the window as a transient window for the window specified by *`parent`*. Dialog windows should be set transient for the main application window they were spawned from. This allows window managers to keep the dialog on top of the main window, or center the dialog over the main window. The gtk.Dialog() constructor and other convenience functions in `PyGTK` will sometimes call the `set_transient_for()` method on your behalf.

On Windows, this method will and put the child window on top of the parent, much as the window manager would have done on X.

gtk.Window.get_transient_for

```
def get_transient_for()
```

Returns :	the transient parent for this window, or `None` if no transient parent has been set.

The `get_transient_for()` method returns the transient parent for this window or `None` if no transient window is set. See the `set_transient_for()` method.

gtk.Window.set_type_hint

```
def set_type_hint(hint)
```

`hint` :	the window type

The `set_type_hint()` method sets the window type hint for the window to the value specified by *`hint`*. The value of *`hint`* must be one of the GDK Window Type Hint Constants.

By setting the type hint for the window, you allow the window manager to decorate and handle the window in a way which is suitable to the method of the window in your application. This method should be called before the window becomes visible. The gtk.Dialog() constructor and other convenience functions in `PyGTK` will sometimes call this method on your behalf.

gtk.Window.get_type_hint

```
    def get_type_hint()
```

Returns :	the type hint for the window.

The `get_type_hint()` method returns the type hint for this window. See the `set_type_hint()` method.

gtk.Window.set_destroy_with_parent

```
    def set_destroy_with_parent(setting)
```

`setting` :	if `TRUE` destroy the window with its transient parent

The `set_destroy_with_parent()` method sets the "destroy−with−parent" property to the value specified by *`setting`*. If *`setting`* is `TRUE`, destroying the transient parent of the window will also destroy the window itself. This is useful for dialogs that shouldn't persist beyond the lifetime of the main window they're associated with.

gtk.Window.get_destroy_with_parent

```
    def get_destroy_with_parent()
```

Returns :	`TRUE` if the window will be destroyed with its transient parent.

The `get_destroy_with_parent()` method returns the value of the "destroy−with−parent" property that determines if the window will be destroyed with its transient parent. See the `set_destroy_with_parent()` method.

gtk.Window.set_resizable

```
    def set_resizable(resizable)
```

`resizable` :	if `TRUE` the user can resize this window

The `set_resizable()` method sets the "resizable" property to the value of *`resizable`*. If *`resizable`* is `TRUE` the user can resize the window. Windows are user resizable by default.

gtk.Window.get_resizable

```
    def get_resizable()
```

Returns :	`TRUE` if the user can resize the window

The `get_resizable()` method returns the value of the "resizable" property. See the `set_resizable()` method.

gtk.Window.set_gravity

```
    def set_gravity(gravity)
```

`gravity` :	the window gravity

The `set_gravity()` method sets the gravity of the window to the value specified by *`gravity`*. The window gravity defines the meaning of coordinates passed to the `move()` method. The value of gravity must be one of the GDK Gravity Constants.

The default window gravity is `gtk.gdk.GRAVITY_NORTH_WEST` which will typically "do what you want."

gtk.Window.get_gravity

```
def get_gravity()
```

Returns :	the window gravity

The `get_gravity()` method returns window gravity. See the `set_gravity()` method.

gtk.Window.set_geometry_hints

```
def set_geometry_hints(geometry_widget, min_width=-1, min_height=-1, max_width=-1, max_heigh
```

`geometry_widget` :	the widget the geometry hints will be applied to
`min_width` :	the minimum width of window (or −1 to use requisition)
`min_height` :	the minimum height of window (or −1 to use requisition)
`max_width` :	the maximum width of window (or −1 to use requisition)
`max_height` :	the maximum height of window (or −1 to use requisition)
`base_width` :	allowed window widths are base_width + width_inc * N where N is any integer
`base_height` :	allowed window heights are base_height + width_inc * N where N is any integer
`width_inc` :	the width resize increment
`height_inc` :	the height resize increment
`min_aspect` :	the minimum width to height ratio
`max_aspect` :	the maximum width to height ratio

The `set_geometry_hints()` method sets up hints about how a window can be resized by the user. You can set the minimum and maximum widths and heights, the base width and height for resizing, the allowed width and height resize increments (e.g. for xterm, you can only resize by the size of a character), and the minimum and maximum aspect ratios. If *`geometry_widget`* is not `None` it specifies the widget to figure the geometry on.

gtk.Window.set_screen

```
def set_screen(screen)
```

`screen` :	a `gtk.gdk.Screen`.

Note

This method is available in PyGTK 2.2 and above.

The `set_screen()` method sets the "screen" property to the `gtk.gdk.Screen` specified by *`screen`*. The "screen" property contains the screen that the window is displayed on. If the window is already mapped, it will be unmapped, and then remapped on the new screen.

gtk.Window.get_screen

```
def get_screen()
```

Returns :	a `gtk.gdk.Screen`.

Note

This method is available in PyGTK 2.2 and above.

The `get_screen`() method returns the `gtk.gdk.Screen` that the window is displayed on.

gtk.Window.is_active

```
def is_active()
```

Returns :	`TRUE` if the window is part of the current active window.

Note

This method is available in PyGTK 2.4 and above.

The `is_active`() method returns `TRUE` if the window is part of the current active toplevel, i.e., the toplevel window receiving keystrokes. The return value is `TRUE` if the window is active the toplevel itself, or if it is, for example, a `gtk.Plug` embedded in the active toplevel. You might use this method if you wanted to draw a widget differently in an active window from a widget in an inactive window. See the `has_toplevel_focus()` method.

gtk.Window.has_toplevel_focus

```
def has_toplevel_focus()
```

Returns :	`TRUE` if the the input focus is within the window

Note

This method is available in PyGTK 2.4 and above.

The `has_toplevel_focus`() method returns `TRUE` if the input focus is within the window. For real toplevel windows, this is identical to `is_active()`, but for embedded windows, like a `gtk.Plug`, the results will differ.

gtk.Window.set_has_frame

```
def set_has_frame(setting)
```

`setting` :	if `TRUE` PyGTK draws the window border

The `set_has_frame`() method sets the flag that causes `PyGTK` to draw its own window border for the window.

Note

This is a special−purpose method for the framebuffer port. For most applications, you want the `set_decorated()` method instead,that tells the window manager whether to draw the window border.

If this method is called on a window with setting of `TRUE`, before it is realized or showed, it will have a "frame" window around the window's `gtk.gdk.Window`, accessible in the window's frame. Using the signal "frame−event" you can receive all events targeted at the frame. This method is used by the linux−fb port to implement managed windows, but it could conceivably be used by X−programs that want to do their own window decorations.

gtk.Window.get_has_frame

```
def get_has_frame()
```

Returns :	`TRUE` if a frame has been added to the window via the `set_has_frame()` method.

The `get_has_frame()` method returns the value of the window's "has_frame" flag that determines if the window has a frame window exterior to its `gtk.gdk.Window`. See the `set_has_frame()` method for more information.

gtk.Window.set_frame_dimensions

```
def set_frame_dimensions(left, top, right, bottom)
```

`left` :	the width of the left border
`top` :	the height of the top border
`right` :	the width of the right border
`bottom` :	the height of the bottom border

The `set_frame_dimensions()` method sets the size of the frame around the window to the values specified by *`left`*, *`top`*, *`bottom`* and *`right`*.

Note

This is a special−purpose method intended for the framebuffer port. See the `set_has_frame()` method. It will have no effect on the window border drawn by the window manager, which is the normal case when using the X Window system.

For windows with frames (see the `set_has_frame()` method) this method can be used to change the size of the frame border.

gtk.Window.get_frame_dimensions

```
def get_frame_dimensions()
```

Returns :	a tuple containing the frame dimensions: the width of the frame at the left; the height of the frame at the top; the width of the frame at the right; and, the height of the frame at the bottom.

The `get_frame_dimensions()` method returns a tuple containing the frame dimensions: the width of the frame at the left; the height of the frame at the top; the width of the frame at the right; and, the height of the frame at the bottom.

Note

This is a special−purpose method intended for the framebuffer port See the `set_has_frame()` method. It will not return the size of the window border drawn by the window manager, which is the normal case when using a windowing system. See the `get_frame_extents()` to get the standard window border extents.

See the `set_has_frame()` and `set_frame_dimensions()` methods for more information.

gtk.Window.set_decorated

```
    def set_decorated(setting)
```

setting :	if TRUE decorate the window

The set_decorated() method sets the decorated flag to the value specified by *setting*. If *setting* is TRUE the window will be decorated. By default, windows are decorated with a title bar, resize controls, etc. Some window managers allow PyGTK to disable these decorations, creating a borderless window. If you set the decorated property to FALSE using this method, PyGTK will do its best to convince the window manager not to decorate the window. On Windows, this method always works, since there's no window manager policy involved.

gtk.Window.get_decorated

```
    def get_decorated()
```

Returns :	TRUE if the window has been set to have decorations

The get_decorated() method returns the value of the decorated flag that determines if the window has been set to have decorations such as a title bar. See the set_decorated() method.

gtk.Window.set_icon_list

```
    def set_icon_list(...)
```

... :	zero or more gtk.gdk.Pixbuf objects

The set_icon_list() method sets up the icon representing the window using the set of gtk.gdk.Pixbuf objects passed as arguments. If no gtk.gdk.Pixbuf objects are passed in the icon is unset and reverts to the default icon. The icon is used when the window is minimized (also known as iconified). Some window managers or desktop environments may also place it in the window frame, or display it in other contexts. This method allows you to pass in the same icon in several hand−drawn sizes. The gtk.gdk.Pixbuf objects should contain the natural sizes your icon is available in; i.e., don't scale the image before passing it to PyGTK. Scaling is postponed until the last minute, when the desired final size is known, to allow best quality. By passing several sizes, you may improve the final image quality of the icon, by reducing or eliminating automatic image scaling. The recommended sizes to provide are: 16x16, 32x32, 48x48 at minimum, and larger images (64x64, 128x128) if you have them.

See the gtk.window_set_default_icon_list() function to set the icon for all windows in your application in one go. Note that transient windows (those who have been set transient for another window using the set_transient_for() method) will inherit their icon from their transient parent. So there's no need to explicitly set the icon on transient windows.

gtk.Window.get_icon_list

```
    def get_icon_list()
```

Returns :	a copy of the window's icon list

The get_icon_list() method returns the list of icons set by the set_icon_list() method.

gtk.Window.set_icon

```
    def set_icon(icon)
```

icon :	an icon image, or None

The `set_icon`() method sets the "icon" property to the value specified by *`icon`*. This icon is used when the window is minimized (also known as iconified). Some window managers or desktop environments may also place it in the window frame, or display it in other contexts. The icon should be provided in whatever size it was naturally drawn; that is, don't scale the image before passing it to `PyGTK`. Scaling is postponed until the last minute, when the desired final size is known, to allow best quality. If you have your icon hand−drawn in multiple sizes, use the `set_icon_list()` method. Then the best size will be used.

This method is equivalent to calling the `set_icon_list()` method with a 1−element list. See the `gtk.window_set_default_icon_list()` function to set the icon for all windows in your application in one go.

gtk.Window.set_icon_from_file

```
def set_icon_from_file(filename)
```

`filename` :	the name of a file containing an icon image
Returns :	`TRUE` if the icon was loaded.

The `set_icon_from_file`() method sets the "icon" property to the icon loaded from the file specified by *`filename`*. The icon is used when the window is minimized (also known as iconified). See the `set_icon()`) method for more information. This method is equivalent to calling the `set_icon()` method with a pixbuf created by loading the image from *`filename`*.

The GError exception is raised if an error occurs while loading the pixbuf from *`filename`*.

gtk.Window.get_icon

```
def get_icon()
```

Returns :	the icon for window

The `get_icon`() method returns the value of the "icon" property set by the `set_icon()` (or if you've called the `set_icon_list()` method, returns the first icon in the icon list).

gtk.Window.set_modal

```
def set_modal(modal)
```

`modal` :	if `TRUE` the window is modal

The `set_modal`() method sets the "modal" property to the value of *`modal`*. If *`modal`* is `TRUE` the window becomes modal. Modal windows prevent interaction with other windows in the same application. Typically modal windows are used for `gtk.Dialog` windows that require a user response before the application can continue. To keep modal dialogs on top of the main application windows, use the `set_transient_for()` method to make the dialog transient for the parent − most window managers will then disallow lowering the dialog below the parent.

gtk.Window.get_modal

```
def get_modal()
```

Returns :	`TRUE` if the window is set to be modal and establishes a grab when shown

The `get_modal`() method returns the value of the "modal" property. If "modal" is `TRUE` the window is modal. See the `set_modal()` method.

gtk.Window.add_mnemonic

```
    def add_mnemonic(keyval, target)
```

`keyval` :	the mnemonic key
`target` :	the widget that gets activated by the mnemonic

The `add_mnemonic()` method adds a mnemonic key specified by *`keyval`* to this window. When the mnemonic key is pressed the widget specified by *`target`* will be activated.

gtk.Window.remove_mnemonic

```
    def remove_mnemonic(keyval, target)
```

`keyval` :	the mnemonic key
`target` :	the widget that gets activated by the mnemonic

The `remove_mnemonic()` method removes the mnemonic specified by *`keyval`* for the widget specified by *`target`* from this window.

gtk.Window.mnemonic_activate

```
    def mnemonic_activate(keyval, modifier)
```

`keyval` :	the mnemonic key
`modifier` :	the modifiers
Returns :	TRUE if the activation was done

The `mnemonic_activate()` method activates the targets associated with the mnemonic specified by *`keyval`*. The window's mnemonic modifier must match *`modifier`* to allow the activation to proceed. See the `set_mnemonic_modifier()` method for more information.

gtk.Window.set_mnemonic_modifier

```
    def set_mnemonic_modifier(modifier)
```

`modifier` :	the modifier mask used to activate mnemonics on this window.

The `set_mnemonic_modifier()` method sets the mnemonic modifier for this window to the value specified by *`modifier`*. The value of *`modifier`* is one of:

`gtk.gdk.SHIFT_MASK`	The Shift key.
`gtk.gdk.CONTROL_MASK`	The Control key.
`gtk.gdk.MOD1_MASK`	The fourth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier, but normally it is the Alt key).

gtk.Window.get_mnemonic_modifier

```
    def get_mnemonic_modifier()
```

Returns :	the modifier mask used to activate mnemonics on this window.

The `get_mnemonic_modifier()` method returns the mnemonic modifier for this window. See the `set_mnemonic_modifier()` method for more detail.

gtk.Window.activate_key

```
def activate_key(event)
```

event :	a gtk.gdk.Event
Returns :	TRUE if a mnemonic or accelerator was found and activated.

Note

This method is available in PyGTK 2.4 and above.

The `activate_key`() method activates mnemonics and accelerators for the window. This is normally called by the default `gtk.Widget` "key−press−event" signal handler for toplevel windows, however in some cases it may be useful to call this directly when overriding the standard key handling for a toplevel window.

gtk.Window.propagate_key_event

```
def propagate_key_event(event)
```

event :	a gtk.gdk.Event
Returns :	TRUE if a widget in the focus chain handled the event.

Note

This method is available in PyGTK 2.4 and above.

The `propagate_key_event`() method propagates a key press or release event to the focus widget and up the focus container chain until a widget handles the key event specified by *event*. This is normally called by the default `gtk.Widget` "key−press−event" and "key−release−event" signal handlers for toplevel windows, however in some cases it may be useful to call this directly when overriding the standard key handling for a toplevel window.

gtk.Window.present

```
def present()
```

The `present`() method presents a window to the user. This may mean raising the window in the stacking order, deiconifying it, moving it to the current desktop, and/or giving it the keyboard focus, possibly dependent on the user's platform, window manager, and preferences. If the window is hidden, this method calls the the `gtk.Widget.show()` method as well. This method should be used when the user tries to open a window that's already open. Say for example the preferences dialog is currently open, and the user chooses Preferences from the menu a second time; use the `present`() method to move the already−open dialog where the user can see it.

gtk.Window.iconify

```
def iconify()
```

The `iconify`() method asks the window manager to iconify (i.e. minimize) the specified the window. Note that you shouldn't assume the window is definitely iconified afterward, because other entities (e.g. the user or window manager) could deiconify it again, or there may not be a window manager in which case iconification isn't possible, etc. But normally the window will end up iconified. Just don't write code that crashes if not. This method can be called before showing a window, in which case the window will be iconified before it ever appears on−screen. You can track iconification via the `gtk.Widget` "window−state−event" signal.

gtk.Window.deiconify

```
    def deiconify()
```

The `deiconify`() method asks the window manager to deiconify (i.e. unminimize) the specified the window. Note that you shouldn't assume the window is definitely deiconified afterward, because other entities (e.g. the user or window manager) could iconify it again before your code which assumes deiconification gets to run. You can track iconification via the `gtk.Widget` "window−state−event" signal.

gtk.Window.stick

```
    def stick()
```

The `stick`() method asks the window manager to stick the window, which means that it will appear on all user desktops. Note that you shouldn't assume the window is definitely stuck afterward, because other entities (e.g. the user or window manager) could unstick it again, and some window managers do not support sticking windows. But normally the window will end up stuck. Just don't write code that crashes if not. This method can be called before showing a window. You can track stickiness via the `gtk.Widget` "window−state−event" signal.

gtk.Window.unstick

```
    def unstick()
```

The unstick() method asks the window manager to unstick the window, which means that it will appear on only one of the user's desktops. Note that you shouldn't assume the window is definitely unstuck afterward, because other entities (e.g. the user or window manager) could stick it again. But normally the window will end up stuck. Just don't write code that crashes if not. You can track stickiness via the `gtk.Widget` "window−state−event" signal.

gtk.Window.maximize

```
    def maximize()
```

The `maximize`() method asks the window manager to maximize the window, so that it becomes full−screen. Note that you shouldn't assume the window is definitely maximized afterward, because other entities (e.g. the user or window manager) could unmaximize it again, and not all window managers support maximization. But normally the window will end up maximized. This method can be called before showing a window, in which case the window will be maximized when it appears on−screen initially. You can track maximization via the `gtk.Widget` "window−state−event" signal.

gtk.Window.unmaximize

```
    def unmaximize()
```

The `unmaximize`() method asks the window manager to unmaximize the window. Note that you shouldn't assume the window is definitely unmaximized afterward, because other entities (e.g. the user or window manager) could maximize it again, and not all window managers honor requests to unmaximize. But normally the window will end up unmaximized. You can track maximization via the `gtk.Widget` "window−state−event" signal.

gtk.Window.fullscreen

```
def fullscreen()
```

Note

This method is available in PyGTK 2.2 and above.

The `fullscreen`() method requests the window manager to place the window in the fullscreen state. Note you shouldn't assume the window is definitely full screen afterward, because other entities (e.g. the user or window manager) could unfullscreen it again, and not all window managers honor requests to fullscreen windows. But normally the window will end up fullscreen. Just don't write code that crashes if not.

You can track the fullscreen state via the `gtk.Widget` "window−state−event" signal.

gtk.Window.unfullscreen

```
def unfullscreen()
```

Note

This method is available in PyGTK 2.2 and above.

The `unfullscreen`() method requests the window manager to toggle off the fullscreen state for the window. Note that you shouldn't assume the window is definitely not full screen afterward, because other entities (e.g. the user or window manager) could fullscreen it again, and not all window managers honor requests to unfullscreen windows. But normally the window will end up restored to its normal state. Just don't write code that crashes if not.

You can track the fullscreen state via the `gtk.Widget` "window_state_event" signal.

gtk.Window.set_keep_above

```
def set_keep_above(setting)
```

`setting` :	if `TRUE` keep the window above other windows

Note

This method is available in PyGTK 2.4 and above.

The `set_keep_above`() method requests the window manager to keep the window on top if *`setting`* is `TRUE`. Note that you shouldn't assume the window is definitely above afterward, because other entities (e.g. the user or window manager) could not keep it above, and not all window managers support keeping windows above. But normally the window will end kept above. Just don't write code that crashes if not.

It's permitted to call this method before showing a window, so the window will be kept above when it appears on−screen initially.

You can track the above state via the `gtk.Widget` "window_state_event" signal.

Note that, according to the Extended Window Manager Hints specification, the above state is mainly meant for user preferences and should not be used by applications e.g. for drawing attention to their dialogs.

gtk.Window.set_keep_below

```
def set_keep_below(setting)
```

setting :	if `TRUE` keep the window below other windows

Note

This method is available in PyGTK 2.4 and above.

The `set_keep_below()` method requests the window manager to keep the window on the bottom (i.e. below all other windows). Note that you shouldn't assume the window is definitely below afterward, because other entities (e.g. the user or window manager) could not keep it below, and not all window managers support putting windows below. But normally the window will be kept below. Just don't write code that crashes if not.

It's permitted to call this function before showing a window, in which case the window will be kept below when it appears on−screen initially.

You can track the below state via the `gtk.Widget` "window_state_event" signal.

Note that, according to the Extended Window Manager Hints specification, the above state is mainly meant for user preferences and should not be used by applications e.g. for drawing attention to their dialogs.

gtk.Window.begin_resize_drag

```
def begin_resize_drag(edge, button, root_x, root_y, timestamp)
```

edge :	the position of the resize control
button :	the mouse button that initiated the drag
root_x :	the X position where the user clicked to initiate the drag, in root window coordinates
root_y :	the Y position where the user clicked to initiate the drag
timestamp :	the timestamp from the click event that initiated the drag

The `begin_resize_drag()` method starts resizing a window from the edge specified by *edge*. The mouse button that started the resize is specified by *button*; the location, by *root_x* and *root_y*; and the time of the event, by *timestamp*. The value of *edge* must be one of the GDK Window Edge Constants.

This method is used if an application has window resizing controls. When `PyGTK` can support it, the resize will be done using the standard mechanism for the window manager or windowing system. Otherwise, `PyGTK` will try to emulate window resizing, potentially not all that well, depending on the windowing system.

gtk.Window.begin_move_drag

```
def begin_move_drag(button, root_x, root_y, timestamp)
```

button :	the mouse button that initiated the drag
root_x :	the X position where the user clicked to initiate the drag, in root window coordinates
root_y :	the Y position where the user clicked to initiate the drag
timestamp :	the timestamp from the click event that initiated the drag

The `begin_move_drag()` method starts moving a window when the user presses the mouse button specified by *button* at the location specified by *root_x* and *root_y* at the time specified by *timestamp*. This method is used if an application has window movement grips. When `PyGTK` can support it, the window movement will be done using the standard mechanism for the window manager or windowing system. Otherwise, PyGTK will try to emulate window movement, potentially not all that well, depending on the windowing system.

gtk.Window.set_default_size

```
    def set_default_size(width, height)
```

width :	the width in pixels, or −1 to unset the default width
height :	the height in pixels, or −1 to unset the default height

The `set_default_size()` method sets the default size of the window to the specified *width* and *height*. If the window's "natural" size (its size request) is larger than the default, the default will be ignored. More generally, if the default size does not obey the geometry hints for the window (the `set_geometry_hints()` method can be used to set these explicitly), the default size will be clamped to the nearest permitted size.

Unlike the `gtk.Widget.set_size_request()` method, which sets a size request for a widget and thus would keep users from shrinking the window, this method only sets the initial size, just as if the user had resized the window themselves. Users can still shrink the window again as they normally would. Setting a default size of −1 means to use the "natural" default size (the size request of the window). For more control over a window's initial size and how resizing works, investigate the `set_geometry_hints()` method.

For some uses, the `resize()` method is more appropriate as it changes the current size of the window, rather than the size to be used on initial display. The `resize()` method always affects the window itself, not the geometry widget. The default size of a window only affects the first time a window is shown; if a window is hidden and re−shown, it will remember the size it had prior to hiding, rather than using the default size. Windows can't actually be 0x0 in size, they must be at least 1x1, but passing 0 for *width* and *height* is OK, resulting in a 1x1 default size.

gtk.Window.get_default_size

```
    def get_default_size()
```

Returns :	a tuple containing the default width and height of the window

The `get_default_size()` method returns a tuple containing the default width and height of the window. A value of −1 for the returned width or height indicates that a default size has not been explicitly set for that dimension, so the "natural" size of the window will be used. See the `set_default_size()` method for more information

gtk.Window.resize

```
    def resize(width, height)
```

width :	the width in pixels to resize the window to
height :	the height in pixels to resize the window to

The `resize()` method resizes the window to the specified *width* and *height* as if the user had done so, obeying geometry constraints. The default geometry constraint is that windows may not be smaller than their size request; to override this constraint, call the `gtk.Widget.set_size_request()` method to set the window's request to a smaller value. If the `resize()` method is called before showing a window for the first time, it overrides any default size set with the `set_default_size()` method. Windows may not be resized smaller than 1 by 1 pixels.

gtk.Window.get_size

```
def get_size()
```

Returns :	a tuple containing the width and height of the window

The `get_size`() method returns a tuple containing the current width and height of the window. If the window is not on–screen, it returns the size `PyGTK` will suggest to the window manager for the initial window size (but this is not reliably the same as the size the window manager will actually select). The size obtained by the `get_size`() method is the last size received in a configure event, that is, `PyGTK` uses its locally–stored size, rather than querying the X server for the size. As a result, if you call the `resize()` method then immediately call the `get_size`() method, the size won't have taken effect yet. After the window manager processes the resize request, PyGTK receives notification that the size has changed via a configure event, and the size of the window gets updated.

Note

Nearly any use of this method creates a race condition, because the size of the window may change between the time that you get the size and the time that you perform some action assuming that size is the current size. To avoid race conditions, connect to "configure_event" on the window and adjust your size–dependent state to match the size delivered in the configure event.

The returned size does *not* include the size of the window manager decorations (aka the window frame or border). Those are not drawn by `PyGTK` which has no reliable method of determining their size.

If you are getting a window size in order to position the window on–screen, there may be a better way. The preferred way is to simply set the window's semantic type with the `set_type_hint()` method, that allows the window manager to center dialogs, etc. Also, if you set the transient parent of dialogs with the `set_transient_for()` method, window managers will often center the dialog over its parent window. It's much preferred to let the window manager handle these things rather than doing it yourself, because all apps will behave consistently and according to user prefs if the window manager handles it. Also, the window manager can take the size of the window decorations/border into account, while your application cannot.

In any case, if you insist on application–specified window positioning, there's *still* a better way than doing it yourself – the `set_position()` method will frequently handle the details for you.

gtk.Window.move

```
def move(x, y)
```

`x` :	the X coordinate to move window to
`y` :	the Y coordinate to move window to

The `move`() method asks the window manager to move the window to the position specified by *`x`* and *`y`*. Window managers are free to ignore this. In fact, most window managers ignore requests for initial window positions (instead using a user–defined placement algorithm) and honor requests after the window has already been shown.

The position is the position of the gravity–determined reference point for the window. The gravity determines two things: first, the location of the reference point in root window coordinates; and second, which point on the window is positioned at the reference point. By default the gravity is `gtk.gdk.GRAVITY_NORTH_WEST`, so the reference point is simply the *`x`*, *`y`* supplied to the `move`() method. The top–left corner of the window decorations (aka window frame or border) will be placed at *`x`*, *`y`*. Therefore, to position a window at the top left of the screen, you want to use the default gravity (which is `gtk.gdk.GRAVITY_NORTH_WEST`) and move the window to 0,0.

To position a window at the bottom right corner of the screen, you would set `gtk.gdk.GRAVITY_SOUTH_EAST`, which means that the reference point is at *x* + the window width and *y* + the window height, and the bottom–right corner of the window border will be placed at that reference point. So, to place a window in the bottom right corner you would first set gravity to south east, then move the window:

```
  window.set_gravity(gtk.gdk.GRAVITY_SOUTH_EAST)
  width, height = window.get_size()
  window.move(gtk.gdk.screen_width() - width, gtk.gdk.screen_height() - height)
```

The extended window manager hints specification at http://www.freedesktop.org/standards/wm–spec.html has a nice table of gravities in the "implementation notes" section. The `get_position()` method documentation may also be relevant.

gtk.Window.get_position

```
    def get_position()
```

Returns :	a tuple containing the X and Y coordinates of the gravity–determined reference point

The `get_position()` method returns a tuple containing the x and y coordinates of the window that you would need to pass to the `move()` method to keep the window in its current position. This means that the meaning of the returned value varies with window gravity. See the `move()` method for more details. If you haven't changed the window gravity, its gravity will be `gtk.gdk.GRAVITY_NORTH_WEST`. This means that the `get_position()` method gets the position of the top–left corner of the window manager frame for the window. The `move()` method sets the position of this same top–left corner.

The `get_position()` method is not 100% reliable because the X Window System does not specify a way to obtain the geometry of the decorations placed on a window by the window manager. Thus `PyGTK` is using a "best guess" that works with most window managers. Moreover, nearly all window managers are historically broken with respect to their handling of window gravity. So moving a window to its current position as returned by the `get_position()` method tends to result in moving the window slightly. Window managers are slowly getting better over time.

If a window has gravity `gtk.gdk.GRAVITY_STATIC` the window manager frame is not relevant, and thus the `get_position()` method will always produce accurate results. However you can't use static gravity to do things like place a window in a corner of the screen, because static gravity ignores the window manager decorations. If you are saving and restoring your application's window positions, you should know that it's impossible for applications to do this without getting it somewhat wrong because applications do not have sufficient knowledge of window manager state. The correct mechanism is to support the session management protocol (see the "GnomeClient" object in the GNOME libraries for example) and allow the window manager to save your window sizes and positions.

gtk.Window.parse_geometry

```
    def parse_geometry(geometry)
```

`geometry` :	the geometry string
Returns :	`TRUE` if string was parsed successfully

The `parse_geometry()` method parses the standard X Window System geometry string specified by *`geometry`*. The geometry string has the format "WIDTHxHEIGHT+XOFFSET+YOFFSET" where WIDTH, HEIGHT, XOFFSET and YOFFSET are specified in pixels (see the X documentation for more details). This method works work on all `PyGTK` ports including Win32 but is primarily intended for an X environment. If either a size or a position can be extracted from the geometry string, the `parse_geometry()` method returns `TRUE` and calls the `set_default_size()` and `move()` methods to resize and move the window.

If the `parse_geometry`() method returns `TRUE`, it will also set the `gtk.gdk.HINT_USER_POS` and `gtk.gdk.HINT_USER_SIZE` hints indicating to the window manager that the size and position of the window was user−specified. This causes most window managers to honor the geometry.

gtk.Window.reshow_with_initial_size

```
    def reshow_with_initial_size()
```

The `reshow_with_initial_size`() method hides the window, then reshows it, resetting the default size and position of the window. Used by GUI builders only.

gtk.Window.tooltips_get_info_from_tip_window

```
    def tooltips_get_info_from_tip_window()
```

Returns :	a 2−tuple containing the `gtk.Tooltips` and `gtk.Widget` displayed in the window or `None`.

Note

This method is available in PyGTK 2.4 and above.

The `tooltips_get_info_from_tip_window`() method returns a 2−tuple containing the `gtk.Tooltips` and `gtk.Widget` displayed in the window. If the window is not displaying tooltips this method returns None. This method is mostly intended for use by accessibility technologies − applications should have little use for it.

gtk.Window.set_focus_on_map

```
    def set_focus_on_map(setting)
```

`setting` :	If `TRUE` this window would like to receive focus when mapped.

Note

This method is available in PyGTK 2.6 and above.

The `set_focus_on_map`() method sets the "focus−on−map" property to the value of *`setting`*. If *`setting`* is `TRUE` a hint is set asking the desktop environment to give focus to the window when it is mapped.

gtk.Window.get_focus_on_map

```
    def get_focus_on_map()
```

Returns :	`TRUE` if the window would like to receive focus when mapped.

Note

This method is available in PyGTK 2.6 and above.

The `get_focus_on_map`() method returns the value of the "focus−on−map" property. See the `set_focus_on_map()` method for more information.

gtk.Window.set_icon_name

```
    def set_icon_name(name)
```

name :	he name of the themed icon or `None`

Note

This method is available in PyGTK 2.6 and above.

The `set_icon_name`() method sets the "icon−name" property to the value of *`name`*. If name is `None`, then the default themed icon will be used. The "icon−name" property contains the name of the icon used for the window. See the `gtk.IconTheme` reference for more information.

gtk.Window.get_icon_name

```
    def get_icon_name()
```

Returns :	The name of the themed icon used for the window icon or `None` if no icon is set.

Note

This method is available in PyGTK 2.6 and above.

The `get_icon_name`() method returns the name of the themed icon for the window, see the `set_icon_name()` method for more information.

Functions

gtk.window_set_default_icon_list

```
    def gtk.window_set_default_icon_list(...)
```

`...` :	zero or more `gtk.gdk.Pixbuf` objects

The `gtk.window_set_default_icon_list`() function sets an icon list to be used as fallback for windows that haven't had the `set_icon_list()` method called on them to set up a window−specific icon list. This function allows you to set up the icon for all windows in your app at once. See the `set_icon_list()` method documentation for more details.

gtk.window_set_default_icon

```
    def gtk.window_set_default_icon(icon)
```

icon :	a `gtk.gdk.Pixbuf`

Note

This function is available in PyGTK 2.4 and above.

The `gtk.window_set_default_icon`() function sets an icon specified by *`icon`* to be used as the fallback for windows that haven't had the `set_icon()` method called on them to set up a window−specific icon. This function allows you to set up the icon for all windows in your app at once.

gtk.window_set_default_icon_from_file

```
    def gtk.window_set_default_icon_from_file(filename)
```

filename :	an icon file name

Note

This function is available in PyGTK 2.2 and above.

The `gtk.window_set_default_icon_from_file()` function sets an icon contained in the file specified by *`filename`* to be used as the fallback for windows that haven't had the `set_icon()` method called on them to set up a window−specific icon. This function allows you to set up the icon for all windows in your app at once.

The GError exception is raised if an error occurs while loading the icon pixbuf from *`filename`*.

gtk.window_get_default_icon_list

```
    def gtk.window_get_default_icon_list()
```

Returns :	a copy of the applications default icon list

The `gtk.window_get_default_icon_list()` function returns the application's default icon list as set by the `gtk.window_set_default_icon_list()` function. See the `set_icon_list()` method documentation for more details.

gtk.window_set_auto_startup_notification

```
    def gtk.window_set_auto_startup_notification()
```

setting :	if `TRUE`, automatically do startup notification

Note

This function is available in PyGTK 2.2 and above.

The `gtk.window_set_auto_startup_notification()` function sets the auto startup notification setting to the value of *`setting`*. If *`setting`* is `TRUE` startup notification will be done automatically.

By default, after showing the first `gtk.Window` for each `gtk.gdk.Screen`, GTK+ calls the `gdk_notify_startup_complete()` function. Call this function to disable the automatic startup notification. You might do this if your first window is a splash screen, and you want to delay notification until after your real main window has been shown. In that example, you would disable startup notification temporarily, show your splash screen, then re−enable it so that showing the main window would automatically result in notification.

gtk.window_list_toplevels

```
    def gtk.window_list_toplevels()
```

Returns :	a list of all the toplevel `gtk.Window` widgets

The `gtk.window_list_toplevels()` function returns a list of all the toplevel `gtk.Window` widgets in the application.

gtk.window_set_default_icon_name

```
def gtk.window_set_default_icon_name()
```

`setting` :	if `TRUE`, automatically do startup notification

Note

This function is available in PyGTK 2.6 and above.

The `gtk.window_set_default_icon_name()` function sets an icon to be used as fallback for windows that haven't had the `set_icon_list()` method called on them from a named themed icon, see the `set_icon_name()` method.

Signals

The "activate−default" gtk.Window Signal

```
def callback(window, user_param1, ...)
```

`window` :	the window that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "activate−default" signal is emitted when the default child widget of *`window`* is activated usually by the user pressing the **Return** or **Enter** key.

The "activate−focus" gtk.Window Signal

```
def callback(window, user_param1, ...)
```

`window` :	the window that received the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)

The "activate−focus" signal is emitted when the child widget with the focus is activated usually by the user pressing the **Space** key.

The "frame−event" gtk.Window Signal

```
def callback(window, event, user_param1, ...)
```

`window` :	the window that received the signal
`event` :	the event that triggered the signal
`user_param1` :	the first user parameter (if any) specified with the `connect()` method
`...` :	additional user parameters (if any)
Returns :	`TRUE` to stop other handlers from being invoked for the event. `FALSE` to propagate the event further.

The "frame−event" signal is emitted when an event other than key press or release or focus change is received on the window's frame.

The "keys−changed" gtk.Window Signal

```
    def callback(window, user_param1, ...)
```

window :	the window that received the signal
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "keys−changed" signal is emitted when a mnemonic accelerator is added, removed or changed or the mnemonic modifier is set.

The "move−focus" gtk.Window Signal

```
    def callback(window, direction, user_param1, ...)
```

window :	the window that received the signal
direction :	the move direction: `gtk.DIR_TAB_FORWARD`, `gtk.DIR_TAB_BACKWARD`, `gtk.DIR_UP`, `gtk.DIR_DOWN`, `gtk.DIR_LEFT` or `gtk.DIR_RIGHT`
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "move−focus" signal is emitted when the focus is changed within the window's child widgets. Usually this happens when the user presses the **Tab**, the **Shift+Tab** or the **Up**, **Down**, **Left** or **Right** arrow keys.

The "set−focus" gtk.Window Signal

```
    def callback(window, widget, user_param1, ...)
```

window :	the window that received the signal
widget :	the widget that receives the focus
user_param1 :	the first user parameter (if any) specified with the connect() method
... :	additional user parameters (if any)

The "set−focus" signal is emitted when the focus changes to *widget* in *window*.

Prev Up Next
gtk.Widget Home gtk.WindowGroup

gtk.WindowGroup
Prev **The gtk Class Reference** Next

gtk.WindowGroup

gtk.WindowGroup   a group of `gtk.Window` widgets

Synopsis

```
class gtk.WindowGroup(gobject.GObject):
    gtk.WindowGroup()
    def add_window(window)
    def remove_window(window)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.WindowGroup
```

Description

A `gtk.WindowGroup` object contains a set of `gtk.Window` widgets that are managed together by some `PyGTK` functions and methods.

Constructor

```
    gtk.WindowGroup()
```

Returns :	a new `gtk.WindowGroup`.

Creates a new `gtk.WindowGroup` object. Grabs added with `gtk.Widget.grab_add()` only affect windows within the same `gtk.WindowGroup`.

Methods

gtk.WindowGroup.add_window

```
    def add_window(window)
```

`window` :	the `gtk.Window` to add

The `add_window()` method adds the `gtk.Window` specified by *`window`* to the windowgroup.

gtk.WindowGroup.remove_window

```
    def remove_window(window)
```

`window` :	the `gtk.Window` to remove

The `remove_window()` method removes the `gtk.Window` specified by *`window`* from the windowgroup.

PyGTK Class Hierarchy

```
gobject.GBoxed
    gtk.Border
    gtk.IconInfo
    gtk.IconSet
    gtk.IconSource
    gtk.Requisition
    gtk.SelectionData
    gtk.TextAttributes
    gtk.TextIter
    gtk.TreeIter
    gtk.TreeRowReference
    gtk.gdk.Color
    gtk.gdk.Cursor
    gtk.gdk.Event
    gtk.gdk.Rectangle
    pango.AttrList
    pango.Color
    pango.FontDescription
    pango.FontMetrics
    pango.GlyphString
    pango.Language
    pango.TabArray
gobject.GInterface
    gtk.CellEditable
    gtk.CellLayout
    gtk.Editable
    gtk.FileChooser
    gtk.TreeDragDest
    gtk.TreeDragSource
    gtk.TreeModel
    gtk.TreeSortable
gobject.GObject
    gtk.AccelGroup
    gtk.Action
        gtk.ToggleAction
            gtk.RadioAction
    gtk.ActionGroup
    gtk.Clipboard
    gtk.EntryCompletion (implements gtk.CellLayout)
    gtk.GenericTreeModel (implements gtk.TreeModel)
    gtk.IconFactory
    gtk.IconTheme
    gtk.ListStore (implements gtk.TreeModel, gtk.TreeDragSource, gtk.TreeDragDest, gtk.TreeSort
    gtk.Object
        gtk.Adjustment
        gtk.CellRenderer
            gtk.CellRendererPixbuf
            gtk.CellRendererText
            gtk.CellRendererToggle
            gtk.GenericCellRenderer
        gtk.FileFilter
        gtk.IMContext
            gtk.IMContextSimple
            gtk.IMMulticontext
        gtk.Tooltips
        gtk.TreeViewColumn (implements gtk.CellLayout)
        gtk.Widget
            gtk.Calendar
            gtk.Container
                gtk.Bin
                    gtk.Alignment
                    gtk.Button
                        gtk.ColorButton
                        gtk.FontButton
                        gtk.ToggleButton
                            gtk.CheckButton
                                gtk.RadioButton
                    gtk.ComboBox (implements gtk.CellLayout)
                        gtk.ComboBoxEntry (implements gtk.CellLayout)
                    gtk.EventBox
                    gtk.Expander
                    gtk.Frame
                        gtk.AspectFrame
                    gtk.HandleBox
                    gtk.Item
                        gtk.MenuItem
                            gtk.CheckMenuItem
                                gtk.RadioMenuItem
                            gtk.ImageMenuItem
                            gtk.SeparatorMenuItem
                            gtk.TearoffMenuItem
                    gtk.ScrolledWindow
                    gtk.ToolItem
                        gtk.SeparatorToolItem
                        gtk.ToolButton
                            gtk.ToggleToolButton
                                gtk.RadioToolButton
                    gtk.Viewport
                    gtk.Window
                        gtk.Dialog
                            gtk.ColorSelectionDialog
                            gtk.FileChooserDialog (implements gtk.FileChooser)
                            gtk.FileSelection
                            gtk.FontSelectionDialog
                            gtk.InputDialog
                            gtk.MessageDialog
                        gtk.Plug
                gtk.Box
                    gtk.ButtonBox
                        gtk.HButtonBox
                        gtk.VButtonBox
                    gtk.HBox
                        gtk.Statusbar
                    gtk.VBox
                        gtk.ColorSelection
                        gtk.FileChooserWidget (implements gtk.FileChooser)
                        gtk.FontSelection
                        gtk.GammaCurve
                gtk.Fixed
                gtk.Layout
                gtk.MenuShell
                    gtk.Menu
                    gtk.MenuBar
                gtk.Notebook
                gtk.Paned
                    gtk.HPaned
                    gtk.VPaned
                gtk.Socket
                gtk.Table
                gtk.TextView
                gtk.Toolbar
                gtk.TreeView
            gtk.DrawingArea
                gtk.Curve
            gtk.Entry (implements gtk.Editable, gtk.CellEditable)
                gtk.SpinButton
            gtk.Invisible
            gtk.Misc
                gtk.Arrow
                gtk.Image
                gtk.Label
                    gtk.AccelLabel
                gtk.ProgressBar
            gtk.Range
                gtk.Scale
                    gtk.HScale
                    gtk.VScale
                gtk.Scrollbar
                    gtk.HScrollbar
                    gtk.VScrollbar
```

```
            gtk.Ruler
                gtk.HRuler
                gtk.VRuler
            gtk.Separator
                gtk.HSeparator
                gtk.VSeparator
    gtk.RcStyle
    gtk.Settings
    gtk.SizeGroup
    gtk.Style
    gtk.TextBuffer
    gtk.TextChildAnchor
    gtk.TextMark
    gtk.TextTag
    gtk.TextTagTable
    gtk.TreeModelFilter (implements gtk.TreeModel, gtk.TreeDragSource)
    gtk.TreeModelSort (implements gtk.TreeModel, gtk.TreeSortable)
    gtk.TreeSelection
    gtk.TreeStore (implements gtk.TreeModel, gtk.TreeDragSource, gtk.TreeDragDest, gtk.TreeSort
    gtk.UIManager
    gtk.WindowGroup
    gtk.gdk.Colormap
    gtk.gdk.Device
    gtk.gdk.Display
    gtk.gdk.DisplayManager
    gtk.gdk.DragContext
    gtk.gdk.Drawable
        gtk.gdk.Pixmap
        gtk.gdk.Window
    gtk.gdk.GC
    gtk.gdk.Image
    gtk.gdk.Keymap
    gtk.gdk.Pixbuf
    gtk.gdk.PixbufAnimation
    gtk.gdk.PixbufAnimationIter
    gtk.gdk.PixbufLoader
    gtk.gdk.Screen
    gtk.gdk.Visual
    pango.Context
    pango.Font
    pango.FontFace
    pango.FontFamily
    pango.FontMap
    pango.Fontset
        pango.FontsetSimple
    pango.Layout
gobject.GPointer
```

Prev Up Next
Copyright and License Notice Home The gobject Class Reference

pango.Attribute
Prev **The pango Class Reference** Next

pango.Attribute

pango.Attribute — an attribute that applies to a section of text

Synopsis

```
class pango.Attribute:
    def copy()
Functions

    def pango.attr_type_register(name)
    def pango.AttrLanguage(language, start_index=0, end_index=1)
    def pango.AttrFamily(family, start_index=0, end_index=1)
    def pango.AttrForeground(red, green, blue, start_index=0, end_index=1)
    def pango.AttrBackground(red, green, blue, start_index=0, end_index=1)
    def pango.AttrSize(size, start_index=0, end_index=1)
    def pango.AttrStyle(style, start_index=0, end_index=1)
    def pango.AttrWeight(weight, start_index=0, end_index=1)
    def pango.AttrVariant(variant, start_index=0, end_index=1)
    def pango.AttrStretch(stretch, start_index=0, end_index=1)
    def pango.AttrFontDesc(desc, start_index=0, end_index=1)
    def pango.AttrUnderline(underline, start_index=0, end_index=1)
    def pango.AttrStrikethrough(strikethrough, start_index=0, end_index=1)
    def pango.AttrRise(rise, start_index=0, end_index=1)
    def pango.AttrShape(ink_rect, logical_rect, start_index=0, end_index=1)
    def pango.AttrScale(scale, start_index=0, end_index=1)
    def pango.AttrFallback(fallback, start_index=0, end_index=1)
```

Attributes

All `pango.Attribute` objects support the following three attributes.

"end_index"	Read−Write	The index of the end of the application of the attribute in the associated text.
"start_index"	Read−Write	The index of the start of the application of the attribute in the associated text.
"type"	Read	The attribute type.

In addition each `pango.Attribute` type supports one or more additional attributes that are specific to the type:

"value"	Read	`ATTR_LANGUAGE`	The `pango.Language`.
"value"	Read	`ATTR_FAMILY`	The string containing the font family name list (e.g. "normal,sans,serif,monospace")
"value"	Read	`ATTR_STYLE`	The font slant style. See the `pango.AttrStyle()` function for more details.
"value"	Read	`ATTR_WEIGHT`	The font weight. See the `pango.AttrWeight()` function for more detail.
"value"	Read	`ATTR_VARIANT`	The font variant. See the `pango.AttrVariant()` function for more detail.
"value"	Read	`ATTR_STRETCH`	The font stretch. See the `pango.AttrStretch()` function for more details.
"value"	Read	`ATTR_SIZE`	The font size.in thousandths of a point.
"desc"	Read	`ATTR_FONT_DESC`	The `pango.FontDescription` object.
"color"	Read	`ATTR_FOREGROUND`	The foreground `pango.Color` object.

"color"	Read	`ATTR_BACKGROUND`	The background `pango.Color` object.
"value"	Read	`ATTR_UNDERLINE`	The underline style. See the `pango.AttrUnderline()` function for more details.
"value"	Read	`ATTR_STRIKETHROUGH`	`TRUE` if the text is struck through.
"value"	Read	`ATTR_RISE`	The displacement of the text from the baseline.
"ink_rect"	Read	`ATTR_SHAPE`	The 4–tuple specifying the ink rectangle. See the `pango.AttrShape()` function for more details.
"logical_rect"	Read	`ATTR_SHAPE`	The 4–tuple specifying the logical rectangle. See the `pango.AttrShape()` function for more details.
"value"	Read	`ATTR_SCALE`	The font size scale factor as a float.
"value"	Read	`ATTR_FALLBACK`	`TRUE` if font fallback is enabled.

Description

The `pango.Attribute` object contains an attribute that applies to a section of text. The predefined attribute types are:

`pango.ATTR_LANGUAGE`	Specifies a `pango.Language`.
`pango.ATTR_FAMILY`	Specifies a font family name list as a string.
`pango.ATTR_STYLE`	Specifies a font slant style. See the `pango.AttrStyle()` function for more details.
`pango.ATTR_WEIGHT`	Specifies a font weight. See the `pango.AttrWeight()` function for more detail.
`pango.ATTR_VARIANT`	Specifies a font variant (normal or small caps). See the `pango.AttrVariant()` function for more detail.
`pango.ATTR_STRETCH`	Specifies a font stretch. See the `pango.AttrStretch()` function for more details.
`pango.ATTR_SIZE`	Specifies a font size in thousandths of a point.
`pango.ATTR_FONT_DESC`	Specifies a `pango.FontDescription`.
`pango.ATTR_FOREGROUND`	Specifies a foreground `pango.Color`.
`pango.ATTR_BACKGROUND`	Specifies a background `pango.Color`.
`pango.ATTR_UNDERLINE`	Specifies an underline style. See the `pango.AttrUnderline()` function for more details.
`pango.ATTR_STRIKETHROUGH`	If `TRUE` the text is struck through.
`pango.ATTR_RISE`	Specifies the displacement of the text from the baseline.
`pango.ATTR_SHAPE`	Specifies a shape. See the `pango.AttrShape()` function for more details.
`pango.ATTR_SCALE`	Specifies a font size scale factor.
`pango.ATTR_FALLBACK`	if `TRUE`, fallback to other fonts is enabled (

Additional attribute types can be registered with the `pango.attr_type_register()` function.

Methods

pango.Attribute.copy

```
def copy()
```

Returns :	a new pango.Attribute object

The `copy()` method returns a new pango.Attribute object that is a copy of this attribute.

Functions

pango.attr_type_register

```
def pango.attr_type_register(name)
```

name :	a name for the type. (Currently not used.)
Returns :	the new attribute type ID integer.

The `attr_type_register()` function returns a new attribute type ID integer value.

pango.AttrLanguage

```
def pango.AttrLanguage(language, start_index=0, end_index=1)
```

language :	a pango.Language object.
start_index :	the index of the start of the attribute application in the text.
end_index :	the index of the end of the attribute application in the text.
Returns :	a new pango.Attribute object.

The `AttrLanguage()` function creates a new pango.Attribute object of the type `pango.ATTR_LANGUAGE` with the pango.Language specified by *language* and the text range specified by *start_index* and *end_index*.

pango.AttrFamily

```
def pango.AttrFamily(family, start_index=0, end_index=1)
```

family :	the string containing a font family name list.
start_index :	the index of the start of the attribute application in the text.
end_index :	the index of the end of the attribute application in the text.
Returns :	a new pango.Attribute object.

The `AttrFamily()` function creates a new pango.Attribute object of the type `pango.ATTR_FAMILY` with the font family name list specified by the string *family* and the text range specified by *start_index* and *end_index*.

pango.AttrForeground

```
def pango.AttrForeground(red, green, blue, start_index=0, end_index=1)
```

red :	the red component of the color in the range 0 to 65535.
green :	the green component of the color in the range 0 to 65535.

`blue` :	the blue component of the color in the range 0 to 65535.
`start_index` :	the index of the start of the attribute application in the text.
`end_index` :	the index of the end of the attribute application in the text.
Returns :	a new `pango.Attribute` object.

The `AttrForeground()` function creates a new `pango.Attribute` of the type `pango.ATTR_FOREGROUND` with the RGB color specified by *`red`*, *`green`* and *`blue`* and the text range specified by *`start_index`* and *`end_index`*.

pango.AttrBackground

```
def pango.AttrBackground(red, green, blue, start_index=0, end_index=1)
```

`red` :	the red component of the color in the range 0 to 65535.
`green` :	the green component of the color in the range 0 to 65535.
`blue` :	the blue component of the color in the range 0 to 65535.
`start_index` :	the index of the start of the attribute application in the text.
`end_index` :	the index of the end of the attribute application in the text.
Returns :	a new `pango.Attribute` object.

The `AttrBackground()` function creates a new pangoAttribute of the type `pango.ATTR_BACKGROUND` with the RGB color specified by *`red`*, *`green`* and *`blue`* and the text range specified by *`start_index`* and *`end_index`*.

pango.AttrSize

```
def pango.AttrSize(size, start_index=0, end_index=1)
```

`size` :	the font size in thousandths of a point.
`start_index` :	the index of the start of the attribute application in the text.
`end_index` :	the index of the end of the attribute application in the text.
Returns :	a new `pango.Attribute` object

The `AttrSize()` function creates a new `pango.Attribute` object of the type `pango.ATTR_SIZE` with the font size specified by *`size`* and the text range specified by *`start_index`* and *`end_index`*.

pango.AttrStyle

```
def pango.AttrStyle(style, start_index=0, end_index=1)
```

`style` :	the font slant style.
`start_index` :	the index of the start of the attribute application in the text.
`end_index` :	the index of the end of the attribute application in the text.
Returns :	a new `pango.Attribute` object

The `AttrStyle()` function creates a new `pango.Attribute` object of the type `pango.ATTR_STYLE` with the font slant style specified by *`style`* and the text range specified by *`start_index`* and *`end_index`*. The value of *`style`* must be one of:

`pango.STYLE_NORMAL`	The font is upright.
`pango.STYLE_OBLIQUE`	The font is slanted in a roman style.
`pango.STYLE_ITALIC`	The font is slanted in an italic style.

pango.AttrWeight

```
def pango.AttrWeight(weight, start_index=0, end_index=1)
```

weight :	the font weight.
start_index :	the index of the start of the attribute application in the text.
end_index :	the index of the end of the attribute application in the text.
Returns :	a new `pango.Attribute` object

The `AttrWeight()` function creates a new `pango.Attribute` object of the type `pango.ATTR_WEIGHT` with the font weight specified by *`weight`* and the text range specified by *`start_index`* and *`end_index`*. The value of *`style`* must be one of:

`pango.WEIGHT_ULTRALIGHT`	The ultralight weight (= 200).
`pango.WEIGHT_LIGHT`	The light weight (=300).
`pango.WEIGHT_NORMAL`	The default weight (= 400).
`pango.WEIGHT_BOLD`	The bold weight (= 700).
`pango.WEIGHT_ULTRABOLD`	The ultrabold weight (= 800).
`pango.WEIGHT_HEAVY`	The heavy weight (= 900).

pango.AttrVariant

```
def pango.AttrVariant(variant, start_index=0, end_index=1)
```

variant :	the font variant.
start_index :	the index of the start of the attribute application in the text.
end_index :	the index of the end of the attribute application in the text.
Returns :	a new `pango.Attribute` object

The `AttrVariant()` function creates a new `pango.Attribute` object of the type `pango.ATTR_VARIANT` with the font variant specified by *`variant`* and the text range specified by *`start_index`* and *`end_index`*. The value of *`variant`* must be one of:

`pango.VARIANT_NORMAL`	A normal font.
`pango.VARIANT_SMALL_CAPS`	A font with the lower case characters replaced by smaller variants of the capital characters.

pango.AttrStretch

```
def pango.AttrStretch(stretch, start_index=0, end_index=1)
```

stretch :	the font stretch style.
start_index :	the index of the start of the attribute application in the text.
end_index :	the index of the end of the attribute application in the text.
Returns :	a new `pango.Attribute` object

The `AttrStretch()` function creates a new `pango.Attribute` object of the type `pango.ATTR_STRETCH` with the font stretch style specified by *`stretch`* and the text range specified by *`start_index`* and *`end_index`*. The value of *`stretch`* must be one of:

`pango.STRETCH_ULTRA_CONDENSED`	The most narrow width
`pango.STRETCH_EXTRA_CONDENSED`	
`pango.STRETCH_CONDENSED`	

`pango.STRETCH_SEMI_CONDENSED`	
`pango.STRETCH_NORMAL`	The normal width.
`pango.STRETCH_SEMI_EXPANDED`	
`pango.STRETCH_EXPANDED`	
`pango.STRETCH_EXTRA_EXPANDED`	
`pango.STRETCH_ULTRA_EXPANDED`	The most expanded width

pango.AttrFontDesc

```
    def pango.AttrFontDesc(desc, start_index=0, end_index=1)
```

`desc` :	a `pango.FontDescription` object.
`start_index` :	the index of the start of the attribute application in the text.
`end_index` :	the index of the end of the attribute application in the text.
Returns :	a new `pango.Attribute` object

The `AttrFontDesc()` function creates a new `pango.Attribute` object of the type `pango.ATTR_FONT_DESC` with the `pango.FontDescription` specified by *`desc`* and the text range specified by *`start_index`* and *`end_index`*.

pango.AttrUnderline

```
    def pango.AttrUnderline(underline, start_index=0, end_index=1)
```

`underline` :	the underline style.
`start_index` :	the index of the start of the attribute application in the text.
`end_index` :	the index of the end of the attribute application in the text.
Returns :	a new `pango.Attribute` object

The `AttrUnderline()` function creates a new `pango.Attribute` object of the type `pango.ATTR_UNDERLINE` with the underline style specified by *`underline`* and the text range specified by *`start_index`* and *`end_index`*. The value of *`underline`* must be one of:

`pango.UNDERLINE_NONE`	No underline should be drawn.
`pango.UNDERLINE_SINGLE`	A single underline should be drawn.
`pango.UNDERLINE_DOUBLE`	A double underline should be drawn.
`pango.UNDERLINE_LOW`	A single underline should be drawn at a position beneath the ink extents of the text being underlined. This should be used only for underlining single characters, such as for keyboard accelerators. `pango.UNDERLINE_SINGLE` should be used for extended portions of text.

pango.AttrStrikethrough

```
    def pango.AttrStrikethrough(strikethrough, start_index=0, end_index=1)
```

`strikethrough` :	if `TRUE` the text should be struck through.
`start_index` :	the index of the start of the attribute application in the text.
`end_index` :	the index of the end of the attribute application in the text.
Returns :	a new `pango.Attribute` object

The `AttrStrikethrough`() function creates a new `pango.Attribute` object of the type `pango.ATTR_STRIKETHROUGH` with the value specified by *`strikethrough`* and the text range specified by *`start_index`* and *`end_index`*. If strikethough is `TRUE` the text should be struck through.

pango.AttrRise

```
    def pango.AttrRise(rise, start_index=0, end_index=1)
```

`rise` :	the displacement of the text from the baseline.
`start_index` :	the index of the start of the attribute application in the text.
`end_index` :	the index of the end of the attribute application in the text.
Returns :	a new `pango.Attribute` object

The `AttrRise`() function creates a new `pango.Attribute` object of the type `pango.ATTR_RISE` with the text displacement from the baseline specified by *`rise`* and the text range specified by *`start_index`* and *`end_index`*.

pango.AttrShape

```
    def pango.AttrShape(ink_rect, logical_rect, start_index=0, end_index=1)
```

`ink_rect` :	the ink rectangle of the shape.
`logical_rect` :	the logical rectangle of the shape.
`start_index` :	the index of the start of the attribute application in the text.
`end_index` :	the index of the end of the attribute application in the text.
Returns :	a new `pango.Attribute` object

The `AttrShape`() function creates a new `pango.Attribute` object of the type `pango.ATTR_SHAPE` with the shape rectangles specified by *`ink_rect`* and *`logical_rect`* and the text range specified by *`start_index`* and *`end_index`*. A shape is used to impose a particular ink and logical rect on the result of shaping a particular glyph. This might be used, for instance, for embedding a picture or a widget inside a `pango.Layout`.

pango.AttrScale

```
    def pango.AttrScale(scale, start_index=0, end_index=1)
```

`scale` :	the font size scale factor as a float.
`start_index` :	the index of the start of the attribute application in the text.
`end_index` :	the index of the end of the attribute application in the text.
Returns :	a new `pango.Attribute` object

The `AttrScale`() function creates a new `pango.Attribute` object of the type `pango.ATTR_SCALE` with the font size scale factor specified by *`scale`* and the text range specified by *`start_index`* and *`end_index`*. The value of *`scale`* can also be one of the following pre–defined values:

`pango.SCALE_XX_SMALL`	0.5787037037037 – the scale factor for three shrinking steps (1 / (1.2 * 1.2 * 1.2)).
`pango.SCALE_X_SMALL`	0.6444444444444 – the scale factor for two shrinking steps (1 / (1.2 * 1.2)).
`pango.SCALE_SMALL`	0.8333333333333 – the scale factor for one shrinking step (1 / 1.2).
`pango.SCALE_MEDIUM`	1.0 – The scale factor for normal size.
`pango.SCALE_LARGE`	1.2 – the scale factor for one magnification step.

`pango.SCALE_X_LARGE`	1.4399999999999 – the scale factor for two magnification steps (1.2 * 1.2).
`pango.SCALE_XX_LARGE`	1.728 – the scale factor for three magnification steps (1.2 * 1.2 * 1.2).

pango.AttrFallback

```
    def pango.AttrFallback(fallback, start_index=0, end_index=1)
```

`fallback` :	if `TRUE`, we should fall back on other fonts for characters the active font is missing.
`start_index` :	the index of the start of the attribute application in the text.
`end_index` :	the index of the end of the attribute application in the text.
Returns :	a new `pango.Attribute` object

Note

This function is available in PyGTK 2.4 and above.

The `AttrFallback`() function creates a new `pango.Attribute` object of the type `pango.ATTR_FALLBACK`. If fallback is `TRUE` other fonts on the system can be used to provide characters missing from the current font. Otherwise, only characters from the closest matching font can be used.

Prev	Up	Next
The pango Class Reference	Home	pango.AttrIterator

pango.AttrIterator

Prev	**The pango Class Reference**	Next

pango.AttrIterator

pango.AttrIterator   an object pointing to a location in a `pango.AttrList`.

Synopsis

```
class pango.AttrIterator:
    def copy()
    def range()
    def next()
    def get(type)
    def get_font()
    def get_attrs()
```

Description

The `pango.AttrIterator` object contains a pointer into a `pango.AttrList`. A `pango.AttrIterator` object is created using the `pango.AttrList.get_iterator()` method and then can be advanced through the attribute changes in the text using the `next()` method. The `pango.AttrIterator` object can access information about the current attributes applied at the iterator location (using the `get()` method) and the range of text that the current attribute change applies to (using the `range()` method).

The `pango.AttrIterator` object is only valid as long as the associated `pango.AttrList` is not changed.

Methods

pango.AttrIterator.copy

```
def copy()
```

Returns :	a new pango.AttrIterator object

The `copy()` method returns a new pango.AttrIterator object that is a copy of this attribute iterator.

pango.AttrIterator.range

```
def range()
```

Returns :	a 2–tuple containing the start and end index of the current attribute in the text.

The `range()` method returns a 2–tuple containing the start and end index of the application of the current attribute change in the text.

pango.AttrIterator.next

```
def next()
```

Returns :	`FALSE` if the iterator is at the end of the pango.AttrList.

The `next()` method advances the iterator to the next attribute change in its pango.AttrList. The `next` method returns `FALSE` if the iterator is at the end of the pango.AttrList; otherwise, `TRUE`.

pango.AttrIterator.get

```
def get(type)
```

`type` :	the pango.Attribute type to find.
Returns :	the pango.Attribute matching type or `None`.

The `get()` method returns the pango.Attribute at the iterator location that matches the specified *`type`*. When multiple attributes of the same type overlap, the attribute whose range starts closest to the current location is returned. If no attribute matches, `None` is returned.

pango.AttrIterator.get_font

```
def get_font()
```

Returns :	a 3–tuple containing a pango.FontDescription, a pango.Language and a list of non–font pango.Attribute objects at the current iterator location.

The `get_font()` method returns a 3–tuple containing a pango.FontDescription holding the current font attributes, a pango.Language object (if a language is set) or `None` and a list of non–font pango.Attribute objects in effect at the current iterator location.

pango.AttrIterator.get_attrs

```
def get_attrs()
```

Returns :	a tuple containing the pango.Attribute objects in effect at the current iterator location.

Note

This method is available in PyGTK 2.4 and above.

The `get_attrs()` method returns a tuple containing the `pango.Attribute` objects in effect at the current iterator location.

Prev	Up	Next
pango.Attribute	Home	pango.AttrList

pango.AttrList

pango.AttrList   an object holding attributes that apply to a section of text

Synopsis

```
class pango.AttrList(gobject.GBoxed):
    pango.AttrList()
    def copy()
    def insert(attr)
    def insert_before(attr)
    def change(attr)
    def splice(other, pos, len)
    def get_iterator()
    def filter(func, data=None)
```

```
Functions

    def pango.parse_markup(markup_text, accel_marker)
```

Description

The `pango.AttrList` object contains a list of attributes that apply to a section of text. The attributes are, in general, allowed to overlap in an arbitrary fashion, however, if the attributes are manipulated only through the `change()` method, the overlap between properties will meet stricter criteria. Since the `pango.AttrList` object is stored as a linear list, it is not suitable for storing attributes for large amounts of text. In general, you should not use a single `pango.AttrList` for more than one paragraph of text.

Constructor

```
    pango.AttrList()
```

Returns :	a new `pango.AttrList` object.

Creates a new `pango.AttrList` object.

Methods

pango.AttrList.copy

```
def copy()
```

Returns :	a new pango.AttrList object

The copy() method returns a new pango.AttrList object that is a copy of the attribute list

pango.AttrList.insert

```
def insert(attr)
```

attr :	the pango.Attribute to insert.

The insert() method inserts the pango.Attribute specified by *attr* into the attribute list. It will be inserted after all other attributes with a matching "start_index".

pango.AttrList.insert_before

```
def insert_before(attr)
```

attr :	the pango.Attribute to insert.

The insert_before() method inserts the pango.Attribute specified by attr into the attribute list. It will be inserted before all other attributes with a matching "start_index".

pango.AttrList.change

```
def change(attr)
```

attr :	the pango.Attribute to insert.

The change() method inserts the pango.Attribute specified by *attr* into the attribute list. It will replace any attributes of the same type on that segment and be merged with any adjoining attributes that are identical. This method is slower than the insert() method for creating an attribute list in order (potentially much slower for large lists). However, the insert() method is not suitable for continually changing a set of attributes since it never removes or combines existing attributes.

pango.AttrList.splice

```
def splice(other, pos, len)
```

other :	another pango.AttrList
pos :	the position in this attribute list to insert *other*
len :	the length of the spliced segment. (Note that this must be specified since the attributes in *other* may only be present at some subsection of this range)

The splice() method splices the pango.AttrList specified by *other* into this attribute list. This method stretches each attribute with a range including the position specified by *pos* in the list by the amount specified by *len*, and then calls the change() method with a copy of each attribute in *other* in sequence (offset in position by *pos*).

pango.AttrList.get_iterator

```
def get_iterator()
```

Returns :	a new pango.AttrIterator.

The `get_iterator()` method creates and returns a `pango.AttrIterator` initialized to the beginning of the attribute list.

pango.AttrList.filter

```
    def filter(func, data=None)
```

`func` :	a callback function that returns `TRUE` if an attribute should be filtered out.
`data` :	data to be passed to *`func`*
Returns :	a new `pango.AttrList` or `None`.

Note

This method is available in PyGTK 2.4 and above.

The `filter()` method arranges for the callback function specified by *`function`* to be called on each `pango.Attribute` in the attribute list with the optional user data specified by `data`. The callback function signature is:

```
  def func(attribute, user_data)
```

where *`attribute`* is an attribute of the attribute list and *`user_data`* is *`data`*. The callback function returns `TRUE` if the attribute should be filtered out of the attribute list. The `filter` method returns a new `pango.AttrList` containing the attributes that were filtered out (removed from) the attribute list.

Functions

pango.parse_markup

```
    def pango.parse_markup(markup_text, accel_marker)
```

`markup_text` :	a string containing the markup to parse.
`accel_marker` :	the Unicode character that precedes an accelerator, or 0 for none
Returns :	a 3–tuple containing a `pango.AttrList`, the plain text in *`markup_text`*, and the first accelerator character in the text.

The `parse_markup()` function parses the text specified by *`markup_text`* and returns a 3–tuple containing a `pango.AttrList`, the plain text from *`markup_text`* and the first accelerator character that is preceded by the Unicode character specified by *`accel_marker`*. The returned accelerator character is a Unicode character that may be u'\x00' if there is no accelerator character.

See the The Pango Markup Language reference for more information.

The GError exception is raised if an error occurs while parsing the markup text.

Prev	Up	Next
pango.AttrIterator	Home	pango.Color

pango.Color

Prev **The pango Class Reference** Next

pango.Color

pango.Color	an object representing a RGB color

Synopsis

```
class pango.Color(gobject.GBoxed):
    def parse(spec)
```

Attributes

"red"	Read	The red component of the color with a value between 0 and 65535.
"green"	Read	The green component of the color with a value between 0 and 65535.
"blue"	Read	The blue component of the color with a value between 0 and 65535.

Description

A `pango.Color` object is a `gobject.GBoxed` type that represents a color with RGB components. Each component value ranges from 0 to 65535.

Methods

pango.Color.parse

```
    def parse(spec)
```

spec :	a string specifying the new color
Returns :	`TRUE` if *spec* was successfully parsed

The `parse`() method fills in the color attributes of the `pango.Color` from the string *spec*. The string in *spec* can either one of a large set of standard names. (Taken from the X11 `rgb.txt` file), or it can be a hex value in the form '`rgb`' '`rrggbb`' '`rrrgggbbb`' or '`rrrrggggbbbb`' where 'r', 'g' and 'b' are hex digits of the red, green, and blue components of the color, respectively. (White in the four forms is '`fff`' '`ffffff`' '`fffffffff`' and '`ffffffffffff`')

Prev	Up	Next
pango.AttrList	Home	pango.Context

pango.Context

Prev **The pango Class Reference** Next

pango.Context

pango.Context	stores global information used to control rendering.

Synopsis

```
class pango.Context(gobject.GObject):
    def list_families()
    def load_font(desc)
    def load_fontset(desc, language)
    def get_metrics(desc, language)
    def set_font_description(desc)
    def get_font_description()
    def get_language()
    def set_language(language)
    def set_base_dir(direction)
    def get_base_dir()
```

Ancestry

```
+-- gobject.GObject
  +-- pango.Context
```

Description

A `pango.Context` object contains global information used to control the rendering process. The information accessible using the `pango.Context` methods includes:

- the default `pango.FontDescription`
- the default `pango.Language`
- the base direction of the text

A `pango.Context` can be created with the `gtk.Widget.create_pango_context()` method. The `pango.Context` associated with a `gtk.Widget` is retrieved using the `gtk.Widget.get_pango_context()`) method.

Methods

pango.Context.list_families

```
    def list_families()
```

Returns :	a tuple containing a set of `pango.FontFamily` objects.

The `list_families()` method returns a tuple containing the list of all the `pango.FontFamily` objects for a context.

pango.Context.load_font

```
    def load_font(desc)
```

`desc` :	a `pango.FontDescription` describing the font to load
Returns :	the `pango.Font` loaded, or `None` if no font matched.

The `load_font()` method loads and returns the `pango.Font` from one of the fontmaps in the context that is the closest match for the `pango.FontDescription` specified by *`desc`*.

pango.Context.load_fontset

```
    def load_fontset(desc, language)
```

desc :	a pango.FontDescription describing the fonts to load
language :	a pango.Language the fonts will be used for
Returns :	the pango.FontSet, or None if no font matched.

The `load_fontset`() method loads and returns a set of fonts (in a pango.FontSet) from the context that can be used to render a font matching the pango.FontDescription specified by *desc* with the pango.Language specified by *language*.

pango.Context.get_metrics

```
    def get_metrics(desc, language)
```

desc :	a pango.FontDescription object
language :	the pango.Language that determines the script to get the metrics for, or None to get the metrics for the entire font.
Returns :	a pango.FontMetrics object.

The `get_metrics`() method returns the overall metric information for the font described by the pango.FontDescription specified by *desc*. Since the metrics may be substantially different for different scripts, a pango.Language (specified by *language*) can be provided to indicate that the metrics should correspond to the script(s) used by that language. If the value of *language* is None, the returned pango.FontMetrics covers the entire font.

The family name in the pango.FontDescription may be a comma separated list of families. If characters from multiple of these families would be used to render the string, then the returned fonts would be a composite of the metrics for the fonts loaded for the individual families.

pango.Context.set_font_description

```
    def set_font_description(desc)
```

desc :	a pango.FontDescription object

The `set_font_description`() method sets the default pango.FontDescription (specified by *desc*) for the context.

pango.Context.get_font_description

```
    def get_font_description()
```

Returns :	the default font description of the context.

The `get_font_description`() method returns the default pango.FontDescription for the context.

pango.Context.get_language

```
    def get_language()
```

Returns :	the global pango.Language.

The `get_language`() method returns the global pango.Language for the context.

pango.Context.set_language

```
    def set_language(language)
```

language :	a pango.Language.

The set_language() method sets the global pango.Language for the context to the value specified by *language*.

pango.Context.set_base_dir

```
    def set_base_dir(direction)
```

direction :	the new base direction

The set_base_dir() method sets the base text direction for the context to the value specified by *direction*. The value of *direction* must be one of:

pango.DIRECTION_LTR	The text is written left–to–right
pango.DIRECTION_RTL	The text is written right–to–left
pango.DIRECTION_TTB_LTR	The text is written vertically top–to–bottom, with the rows ordered from left to right.
pango.DIRECTION_TTB_RTL	The text is written vertically top–to–bottom, with the rows ordered from right to left.

pango.Context.get_base_dir

```
    def get_base_dir()
```

Returns :	the base direction for the context.

The get_base_dir() method returns the base text direction for the context. See the set_base_dir() method for more information.

Prev	Up	Next
pango.Color	Home	pango.Font

	pango.Font	
Prev	**The pango Class Reference**	Next

pango.Font

pango.Font — a rendering–system independent font representation.

Synopsis

```
class pango.Font(gobject.GObject):
    def describe()
    def get_metrics(language)
    def get_glyph_extents(glyph)
Functions

    def pango.PIXELS(size)
    def pango.ASCENT(rect)
    def pango.DESCENT(rect)
    def pango.RBEARING(rect)
    def pango.LBEARING(rect)
```

Ancestry

```
+-- gobject.GObject
  +-- pango.Font
```

Description

A `pango.Font` object represents a font in a rendering–system independent way. A `pango.Font` is returned from the `pango.Context.load_font()`, `pango.FontSet.get_font()` and `pango.FontMap.load_font()`) methods.

Methods

pango.Font.describe

```
    def describe()
```

Returns :	a `pango.FontDescription` object.

The `describe()` method returns a description of the font in a `pango.FontDescription` object.

pango.Font.get_metrics

```
    def get_metrics(language)
```

`language` :	a `pango.Language` that determines the script to get the metrics for, or `None` to get the metrics for the entire font.
Returns :	a `pango.FontMetrics` object.

The `get_metrics()` method returns a `pango.FontMetrics` object containing the metric information for a font using the `pango.Language` specified by *`language`* to limit the metrics to the script(s) used by *`language`*. If the value of *`language`* is `None`, the font metrics for the entire font is returned.

pango.Font.get_glyph_extents

```
    def get_glyph_extents(glyph)
```

`glyph` :	the glyph index
Returns :	a 2–tuple containing two 4–tuples representing the ink and logical rectangles used to store the extents of *`glyph`* as drawn.

The `get_glyph_extents()` method returns a 2–tuple containing two 4–tuples representing the values of the logical and ink extent rectangles of the specified *`glyph`* within a font. The coordinate system for each rectangle has its origin at the base line and horizontal origin of the character with increasing coordinates extending to the right and down. The `pango.ASCENT()`, `pango.DESCENT()`, `pango.LBEARING()`, and `pango.RBEARING` functions can be used to convert from the extents rectangle to more traditional font metrics. The units of the rectangles are in `1/pango.SCALE` of a device unit.

Functions

pango.PIXELS

```
    def pango.PIXELS(size)
```

size :	the integer value to convert to pango pixels
Returns :	the pixel value

The `pango.PIXELS`() function converts and returns the integer value specified by *size* to pango pixels.

pango.ASCENT

```
    def pango.ASCENT(rect)
```

rect :	a 4–tuple representing an extent rectangle's (x, y, width, height) value
Returns :	the ascent value of *rect*

The `pango.ASCENT`() function returns the ascent value of the extent rectangle specified by *rect*.

pango.DESCENT

```
    def pango.DESCENT(rect)
```

rect :	a 4–tuple representing an extent rectangle's (x, y, width, height) value
Returns :	the descent value of *rect*

The `pango.DESCENT`() function returns the descent value of the extent rectangle specified by *rect*.

pango.RBEARING

```
    def pango.RBEARING(rect)
```

rect :	a 4–tuple representing an extent rectangle's (x, y, width, height) value
Returns :	the right bearing value of *rect*

The `pango.RBEARING`() function returns the right bearing value of the extent rectangle specified by *rect*.

pango.LBEARING

```
    def pango.LBEARING(rect)
```

rect :	a 4–tuple representing an extent rectangle's (x, y, width, height) value
Returns :	the left bearing value of *rect*

The `pango.LBEARING`() function returns the left bearing value of the extent rectangle specified by *rect*.

Prev	Up	Next
pango.Context	Home	pango.FontDescription

pango.FontDescription

Prev **The pango Class Reference** Next

pango.FontDescription

pango.FontDescription — an object representing a description of a font.

Synopsis

```
class pango.FontDescription(gobject.GBoxed):
    pango.FontDescription(str=None)
    def copy()
    def copy_static()
    def hash()
    def set_family(family)
    def set_family_static(family)
    def get_family()
    def set_style(style)
    def get_style()
    def set_variant(variant)
    def get_variant()
    def set_weight(weight)
    def get_weight()
    def set_stretch(stretch)
    def get_stretch()
    def set_size(size)
    def get_size()
    def get_set_fields()
    def unset_fields(to_unset)
    def merge(desc_to_merge, replace_existing)
    def merge_static(desc_to_merge, replace_existing)
    def better_match(old_match, new_match)
    def to_string()
    def to_filename()
```

Description

A `pango.FontDescription` represents the description of an ideal font. They are used to both specify the characteristics of a font to load and to list the available fonts on the system.

Constructor

```
  pango.FontDescription()
```

Returns :	a `pango.FontDescription` object.

Creates a new `pango.FontDescription` object from the representation in the string specified by *`str`*. The format of the string representation is:

```
  "[FAMILY-LIST] [STYLE-OPTIONS] [SIZE]"
```

where `FAMILY-LIST` is a comma separated list of families optionally terminated by a comma, `STYLE_OPTIONS` is a whitespace separated list of words where each `WORD` describes one of style, variant, weight, or stretch, and `SIZE` is an decimal number (size in points). For example the following are all valid string representations:

```
  "sans bold 12"
  "serif,monospace bold italic condensed 16"
  "normal 10"
```

The commonly available font families are: Normal, Sans, Serif and Monospace. The available styles are:

Normal	the font is upright.
Oblique	the font is slanted, but in a roman style.
Italic	the font is slanted in an italic style.

The available weights are:

Ultra–Light	the ultralight weight (= 200)
Light	the light weight (=300)
Normal	the default weight (= 400)
Bold	the bold weight (= 700)
Ultra–Bold	the ultra–bold weight (= 800)
Heavy	the heavy weight (= 900)

The available variants are:

Normal
Small–Caps

The available stretch styles are:

Ultra–Condensed	the smallest width
Extra–Condensed	
Condensed	
Semi–Condensed	
Normal	the normal width
Semi–Expanded	
Expanded	
Extra–Expanded	
Ultra–Expanded	the widest width

Methods

pango.FontDescription.copy

```
    def copy()
```

Returns :	a new pango.FontDescription

The copy() method returns a new copy of this font description.

pango.FontDescription.copy_static

```
    def copy_static()
```

Returns :	a new pango.FontDescription..

The copy_static() method is like the copy() method, but only a shallow copy is made of the family name and other allocated attribute fields. The result can only be used until the original font description is modified or freed. This is meant to be used when the copy is only needed temporarily.

pango.FontDescription.hash

```
def hash()
```

Returns :	the hash value.

The `hash`() method computes and returns a hash of the `pango.FontDescription`.

pango.FontDescription.set_family

```
def set_family(family)
```

`family` :	a string representing the family name.

The `set_family`() method sets the family name attribute field of the font description to the value specified by *`family`*. The family name represents a family of related font styles, and will resolve to a particular `pango.FontFamily`. In some uses of `pango.FontDescription`, it is also possible to use a comma separated list of family names for this field.

pango.FontDescription.set_family_static

```
def set_family_static(family)
```

`family` :	a string representing the family name.

The set_family_static() method is like the `set_family()`, except that no copy of *`family`* is made. The caller must make sure that the string passed in stays around until the font description is no longer needed or the family name is set again. This method can be used if *`family`* is only needed temporarily.

pango.FontDescription.get_family

```
def get_family()
```

Returns :	the family name or `None` if not previously set.

The `get_family`() method returns the family name attribute field of the font description. See the `set_family()` method for more information.

pango.FontDescription.set_style

```
def set_style(style)
```

`style` :	the style for the font description

The `set_style`() method sets the slant style attribute field of the `pango.FontDescription` to the value specified by *`style`*. The value of *`style`* must be either `pango.STYLE_NORMAL`, `pango.STYLE_ITALIC`, or `pango.STYLE_OBLIQUE`. Most fonts will either have a italic style or an oblique style, but not both, and font matching in Pango will match italic specifications with oblique fonts and vice−versa if an exact match is not found.

pango.FontDescription.get_style

```
def get_style()
```

Returns :	the slant style for the font description.

The `get_style`() method returns the slant style attribute field of the `pango.FontDescription`. See the `set_style()` method for more details. Use the `get_set_fields()` method to find out if the field was explicitly set.

pango.FontDescription.set_variant

```
def set_variant(variant)
```

variant :	the variant type for the font description.

The `set_variant()` method sets the variant attribute field of a font description to the value specified by *`variant`*. The value of *`variant`* must be either `pango.VARIANT_NORMAL` or `pango.VARIANT_SMALL_CAPS`.

pango.FontDescription.get_variant

```
def get_variant()
```

Returns :	the variant style of the font description.

The `get_variant()` method returns the variant style of a `pango.FontDescription`. See the `set_variant()` method for more information. Use the `get_set_fields()` method to find out if the field was explicitly set.

pango.FontDescription.set_weight

```
def set_weight(weight)
```

weight :	the weight for the font description.

The `set_weight()` method sets the weight attribute field of a font description to the value specified by *`weight`*. The value of *`weight`* specifies how bold or light the font should be in a range from 100 to 900. The predefined values of weight are:

`pango.WEIGHT_ULTRALIGHT`	the ultralight weight (= 200)
`pango.WEIGHT_LIGHT`	the light weight (=300)
`pango.WEIGHT_NORMAL`	the default weight (= 400)
`pango.WEIGHT_BOLD`	the bold weight (= 700)
`pango.WEIGHT_ULTRABOLD`	the ultrabold weight (= 800)
`pango.WEIGHT_HEAVY`	the heavy weight (= 900)

pango.FontDescription.get_weight

```
def get_weight()
```

Returns :	the weight field for the font description.

The `get_weight()` method returns the value of the weight attribute of a font description. See the `set_weight()` method for more information. Use the `get_set_fields()` method to determine if the attribute was explicitly set.

pango.FontDescription.set_stretch

```
def set_stretch(stretch)
```

stretch :	the stretch for the font description

The `set_stretch()` method sets the stretch attribute field of a font description to the value specified by *`stretch`*. The value of *`stretch`* specifies how narrow or wide the font should be relative to the base width of the font family:

`pango.STRETCH_ULTRA_CONDENSED`	the narrowest width
`pango.STRETCH_EXTRA_CONDENSED`	
`pango.STRETCH_CONDENSED`	
`pango.STRETCH_SEMI_CONDENSED`	
`pango.STRETCH_NORMAL`	the normal (base) width
`pango.STRETCH_SEMI_EXPANDED`	
`pango.STRETCH_EXPANDED`	
`pango.STRETCH_EXTRA_EXPANDED`	
`pango.STRETCH_ULTRA_EXPANDED`	the widest width

pango.FontDescription.get_stretch

```
    def get_stretch()
```

Returns : the stretch field for the font description

The `get_stretch()` method returns the stretch attribute field of a font description. See the `set_stretch()` method for more details. Use the `get_set_fields()` method to determine if the field was explicitly set.

pango.FontDescription.set_size

```
    def set_size(size)
```

`size` : the size for the font description in pango units.

The `set_size()` method sets the size attribute field of a font description to the value specified by *`size`*. The value of *`size`* is specified in pango units. There are `pango.SCALE` (1024) pango units in one device unit (the device unit is a point for font sizes).

pango.FontDescription.get_size

```
    def get_size()
```

Returns : the size for the font description in pango units.

The `get_size()` method returns the value of the size attribute field of a font description. See the `get_size()` method for more information. There are `pango.SCALE` pango units in one device unit (point). If the stretch attribute field has not previously been set, 0 is returned. Use the `get_set_fields()` method to determine if the field was explicitly set.

pango.FontDescription.get_set_fields

```
    def get_set_fields()
```

Returns : a bitmask with bits set corresponding to the font description attribute fields that have been set.

The `get_set_fields()` method returns a value that indicates which attribute fields in a font description have been set. The value returned is a combination of:

`pango.FONT_MASK_FAMILY`	the font family has been set.
`pango.FONT_MASK_STYLE`	the font slant style has been set.
`pango.FONT_MASK_VARIANT`	the font variant has been set.
`pango.FONT_MASK_WEIGHT`	the font weight has been set.

`pango.FONT_MASK_STRETCH`	the font stretch has been set.
`pango.FONT_MASK_SIZE`	the font size has been set.

pango.FontDescription.unset_fields

```
    def unset_fields(to_unset)
```

`to_unset` :	a bitmask of attribute fields in the font description to unset.

The `unset_fields`() method unsets the attribute fields (specified by *`to_unset`*) in the `pango.FontDescription`. Note that this merely marks the attribute fields cleared, it does not clear the settings.

pango.FontDescription.merge

```
    def merge(desc_to_merge, replace_existing)
```

`desc_to_merge` :	the `pango.FontDescription` to merge into the font description
`replace_existing` :	if `TRUE`, replace attribute fields in the font description with the corresponding values from *`desc_to_merge`*, even if they are already exist.

The `merge`() method merges the attribute fields that are set in the `pango.FontDescription` specified by *`desc_to_merge`* into the attribute fields in the font description. If *`replace_existing`* is `FALSE`, only fields in the font description that are not already set are affected. If *`replace_existing`* is `TRUE`, attribute fields that are already set will also be replaced.

pango.FontDescription.merge_static

```
    def merge_static(desc_to_merge, replace_existing)
```

`desc_to_merge` :	the `pango.FontDescription` to merge from
`replace_existing` :	if `TRUE`, replace attribute fields in the font description with the corresponding values from *`desc_to_merge`*, even if they are already exist.

The `merge_static`() method is similar to the `merge()` method, but only a shallow copy is made of the family name and other allocated fields. The font description can only be used until *`desc_to_merge`* is modified or freed. This is meant to be used when the merged font description is only needed temporarily.

pango.FontDescription.better_match

```
    def better_match(old_match, new_match)
```

`old_match` :	a `pango.FontDescription`, or `None`
`new_match` :	a `pango.FontDescription`
Returns :	`TRUE` if *`new_match`* is a better match

The `better_match`() method determines if the attributes of the `pango.FontDescription` specified by *`new_match`* are a closer match for the font description than the `pango.FontDescription` specified by *`old_match`*. If *`old_match`* is `None`, determines if *`new_match`* is a match at all. The weight and style attribute need only match approximately but the other attributes must match exactly.

pango.FontDescription.to_string

```
    def to_string()
```

Returns :	a string representation of the font description.

The to_string() method returns a string representation of the font description. See the pango.FontDescription() constructor for a description of the format of the string representation. The family list in the string description will only have a terminating comma if the last word of the list is a valid style option.

pango.FontDescription.to_filename

```
    def to_filename()
```

Returns :	a string representation of the font description as a filename.

The to_filename() method returns a filename representation of a font description. The filename is identical to the result from calling the to_string() method, but underscores replace characters that are not typically used in filenames, and it is in lower case only.

pango.FontFace

pango.FontFace — an object representing a group of fonts varying only in size.

Synopsis

```
class pango.FontFace(gobject.GObject):
    def describe()
    def get_face_name()
    def list_sizes()
```

Description

A pango.FontFace object represents a group of fonts with the same family, weight, slant, stretch and width but varying sizes. A list of font faces can be retrieved from a pango.FontFamily object using the pango.FontFamily.list_faces() method.

Ancestry

```
+-- gobject.GObject
  +-- pango.FontFace
```

Methods

pango.FontFace.describe

```
def describe()
```

Returns :	a pango.FontDescription object containing the description of the face.

The `describe()` method returns a pango.FontDescription object containing the family, style, variant, weight and stretch of the pango.FontFace. The size attribute field will be unset.

pango.FontFace.get_face_name

```
def get_face_name()
```

Returns :	the face name for the face.

The `get_face_name()` method returns a string representing this font face. This name is unique among all faces in the family and is suitable for displaying to users.

pango.FontFace.list_sizes

```
def list_sizes()
```

Returns :	a tuple containing a list of face sizes in pango units or `None`

Note

This method is available in PyGTK 2.4 and above.

The `list_sizes()` method returns a tuple containing the available sizes for a bitmap font. If the font face is a scalable font this method returns `None`.

Prev Up Next
pango.FontDescription Home pango.FontFamily

pango.FontFamily

pango.FontFamily — an object representing a family of related font faces.

Synopsis

```
class pango.FontFamily(gobject.GObject):
    def list_faces()
    def get_name()
    def is_monospace()
```

Ancestry

```
+-- gobject.GObject
  +-- pango.FontFamily
```

Description

The `pango.FontFamily` object is used to represent a family of related font faces. The faces in a family share a common design, but differ in slant, weight, width and other aspects. A list of `pango.FontFamily` objects can be retrieved from a `pango.Context` object using the `pango.Context.list_families()` method and from a `pango.FontMap` object using the `pango.FontMap.list_families()` method.

Methods

pango.FontFamily.list_faces

```
    def list_faces()
```

Returns :	a list of `pango.FontFace` objects.

The `list_faces()` method returns a list of the different `pango.FontFace` object that make up the font family>. The faces in a family share a common design, but differ in slant, weight, width and other aspects.

pango.FontFamily.get_name

```
    def get_name()
```

Returns :	the name of the family.

The `get_name()` method returns a string containing the name of the font family. The name is unique among all fonts for the font backend and can be used in a `pango.FontDescription` to specify that a face from this family is desired.

pango.FontFamily.is_monospace

```
    def is_monospace()
```

Returns :	the name of the family.

The `is_monospace()` method returns `TRUE` if the font family describes a monospace font. A monospace font is a font designed for text display where the the characters form a regular grid. For Western languages this would mean that the advance width of all characters are the same, but this categorization also includes Asian fonts which include double−width characters: characters that occupy two grid cells. The best way to find out the grid−cell size is to call the `get_approximate_digit_width()` method, since the results of the `get_approximate_char_width()`

Prev	Up	Next
pango.FontFace	Home	pango.FontMap

pango.FontMap

Prev **The pango Class Reference** Next

pango.FontMap

pango.FontMap   an object that represents the set of fonts available for a particular rendering system.

Synopsis

```
class pango.FontMap(gobject.GObject):
    def load_font(context, desc)
    def load_fontset(context, desc, language)
    def list_families()
    def get_shape_engine_type()
```

Ancestry

```
+-- gobject.GObject
  +-- pango.FontMap
```

Description

A `pango.FontMap` object represents the set of fonts available for a particular rendering system. There appears to be no way to retrieve a `pango.FontMap` object in `PyGTK`.

Methods

pango.FontMap.load_font

```
    def load_font(context, desc)
```

context :	the `pango.Context` the font will be used with
desc :	a `pango.FontDescription` describing the font to load
Returns :	the loaded font , or `None` if no font matched.

The `load_font`() method loads the `pango.Font` in the fontmap that is the closest match for the `pango.FontDescription` specified by *`desc`* in the `pango.Context` specified by *`context`*.

pango.FontMap.load_fontset

```
    def load_fontset(context, desc, language)
```

context :	the `pango.Context` the font will be used with
desc :	a `pango.FontDescription` describing the font to load
language :	a `pango.Language` the fonts will be used for
Returns :	a `pango.FontSet`, or `None` if no font matched.

The `load_fontset`() method loads a set of `pango.Font` objects in the fontmap that can be used to render a font matching the `pango.FontDescription` specified by *`desc`* for the `pango.Language` specified by *`language`* in the `pango.Context` specified by *`context`*.

pango.FontMap.list_families

```
    def list_families()
```

Returns :	a list of `pango.FontFamily` objects.

The `list_families()` method returns a list of all `pango.FontFamily` objects for the fontmap.

pango.FontMap.get_shape_engine_type

```
def get_shape_engine_type()
```

Returns :	the ID string for the shape engines for the font map.

Note

This method is available in PyGTK 2.4 and above.

The `get_shape_engine_type`() method returns the render ID for the shape engines for the font map

pango.FontMetrics

pango.FontMetrics — an object containing overall metric information for a font.

Synopsis

```
class pango.FontMetrics(gobject.GBoxed):
    def get_ascent()
    def get_descent()
    def get_approximate_char_width()
    def get_approximate_digit_width()
```

Description

A `pango.FontMetrics` object holds the overall metric information for a font. A `pango.FontMetrics` object is returned from the following methods:

- `pango.Context.get_metrics()`
- `pango.Font.get_metrics()`
- `pango.Fontset.get_metrics()`

Methods

pango.FontMetrics.get_ascent

```
def get_ascent()
```

Returns :	the ascent in pango units. (1 point == `pango.SCALE` pango units.)

The `get_ascent()` method returns the font ascent in pango units where one font point is equal to pango.SCALE (1024) pango units. The ascent is the distance from the baseline to the logical top of a line of text. (The logical top may be above or below the top of the actual drawn ink. It is necessary to lay out the text to figure where the ink will be.)

pango.FontMetrics.get_descent

```
  def get_descent()
```

Returns : the descent in pango units. (1 point == `pango.SCALE` pango units.)

The `get_descent()` method returns the font descent in pango units where one font point is equal to pango.SCALE (1024) pango units. The descent is the distance from the baseline to the logical bottom of a line of text. (The logical bottom may be above or below the bottom of the actual drawn ink. It is necessary to lay out the text to figure where the ink will be.)

pango.FontMetrics.get_approximate_char_width

```
  def get_approximate_char_width()
```

Returns : the character width in pango units. (1 point == `pango.SCALE` pango units.)

The `get_approximate_char_width()` method returns the approximate character width for a font in pango units where one font point is equal to pango.SCALE (1024) pango units. This is merely a representative value that is useful, for example, for determining the initial size for a window. Actual characters in text will be wider and narrower than this.

pango.FontMetrics.get_approximate_digit_width

```
  def get_approximate_digit_width()
```

Returns : the digit width in pango units. (1 point == `pango.SCALE` pango units.)

The `get_approximate_digit_width()` method returns the approximate digit width for a font in pango units where one font point is equal to pango.SCALE (1024) pango units. This is merely a representative value that is useful, for example, for determining the initial size for a window. Actual digits in text can be wider and narrower than this, though this value is generally somewhat more accurate than the result of the `get_approximate_char_width()` method.

Prev Up Next
pango.FontMap Home pango.Fontset

pango.Fontset
Prev **The pango Class Reference** Next

pango.Fontset

pango.Fontset an object containing a set of `pango.Font` objects.

Synopsis

```
class pango.Fontset(gobject.GObject):
    def get_font(wc)
    def get_metrics()
    def foreach(func, data=None)
```

Ancestry

```
+-- gobject.GObject
  +-- pango.Fontset
```

Description

A `pango.Fontset` object holds a set of `pango.Font` objects. A `pango.FontSet` object is returned from the following methods:

- `pango.Context.load_fontset()`
- `pango.FontMap.load_fontset()`

Methods

pango.Fontset.get_font

```
def get_font(wc)
```

`wc` :	a unicode character
Returns :	a `pango.Font`.

The `get_font()` method returns the `pango.Font` in the fontset that contains the best glyph for the unicode character specified by *`wc`*.

pango.Fontset.get_metrics

```
def get_metrics()
```

Returns :	a `pango.FontMetrics` object.

The `get_metrics()` method returns a `pango.FontMetrics` object that contains the overall metric information for the fonts in the fontset.

pango.Fontset.foreach

```
def foreach(func, data=None)
```

`func` :	a callback function
`data` :	user data to pass to *`func`*

Note

This method is available in PyGTK 2.4 and above.

The `foreach()` method invokes the function specified by *`func`* on each `pango.Font` of the font set passing it the optional user data specified by *`data`*. The signature of *`func`* is:

```
def func(fontset, font, user_data)
```

where *`fontset`* is the `pango.Fontset` containing the `pango.Font` *`font`* and *`user_data`* is *`data`*

pango.FontsetSimple

pango.FontsetSimple — a simple container for a set of fonts

Synopsis

```
class pango.FontsetSimple(pango.Fontset):
    pango.FontsetSimple(language)
    def append(font)
    def size()
```

Ancestry

```
+-- gobject.GObject
  +-- pango.Fontset
    +-- pango.FontsetSimple
```

Description

Note

This object is available in PyGTK 2.4 and above.

A pango.FontsetSimple is a subclass of pango.Fontset that provides a simple container for storing a set of pango.Font objects. The set of fonts in a pango.FontsetSimple are assemble by using the append() method.

Constructor

```
pango.FontsetSimple(language)
```

language :	a pango.Language object
Returns :	a new pango.FontsetSimple object.

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new pango.FontsetSimple for the pango.Language specified by language.

Methods

pango.FontsetSimple.append

```
def append(font)
```

font :	a pango.Font.

Note

This method is available in PyGTK 2.4 and above.

The `append`() method adds the `pango.Font` specified by font to the fontset.

pango.FontsetSimple.size

```
    def size()
```

Returns :	the size of the font set.

Note

This method is available in PyGTK 2.4 and above.

The `size`() method returns the number of fonts in the fontset.

pango.GlyphString

pango.GlyphString — an object holding strings of glyphs and glyph information.

Synopsis

```
class pango.GlyphString(gobject.GBoxed):
    pango.GlyphString()
    def set_size(new_len)
    def copy()
    def extents(font)
    def extents_range(start, end, font)
    def get_logical_widths(text, embedding_level)
```

Attributes

"num_glyphs"	Read	The number of glyphs in the glyph string.

Description

A `pango.GlyphString` object contains strings of glyphs with geometry and visual attribute information.

Constructor

```
    pango.GlyphString()
```

Returns :	a new `pango.GlyphString`

Creates a new pango.GlyphString containing no glyphs.

Methods

pango.GlyphString.set_size

```
def set_size(new_len)
```

new_len :	the new length of the string.

The set_size() method resizes the glyph string to the length specified by *new_len*.

pango.GlyphString.copy

```
def copy()
```

Returns :	a pango.GlyphString

The copy() method returns a pango.GlyphString that is a copy of the glyph string.

pango.GlyphString.extents

```
def extents(font)
```

font :	a pango.Font
Returns :	a 2–tuple containing two 4–tuples representing the ink and logical extents rectangles of the glyph string.

The extents() method returns a 2–tuple containing two 4–tuples representing the logical and ink extents rectangles of the glyph string as rendered in the pango.Font specified by *font*. See the pango.Font.get_glyph_extents() for details about the interpretation of the rectangles.

pango.GlyphString.extents_range

```
def extents_range(start, end, font)
```

start :	start index
end :	end index
font :	a pango.Font
Returns :	a 2–tuple containing two 4–tuples representing the ink and logical extents rectangles of the glyph string range.

The extents_range() method returns a 2–tuple containing two 4–tuples representing the logical and ink extents rectangles of a range (specified by *start* and *end*) of the glyph string as rendered in the pango.Font specified by *font*. The extents are relative to the start of the glyph string range (the origin of their coordinate system is at the start of the range, not at the start of the entire glyph string).

pango.GlyphString.get_logical_widths

```
def get_logical_widths(text, embedding_level)
```

text :	the text corresponding to the glyphs
embedding_level :	the embedding level of the string

Returns :	a list containing the calculated character widths.

The `get_logical_widths()` method returns a list of the screen width of the characters in the specified *`text`* that corresponds to the glyph string. When multiple characters compose a single cluster, the width of the entire cluster is divided equally among the characters.

pango.Language

pango.Language — an object that represents a language tag.

Synopsis

```
class pango.Language(gobject.GBoxed):
    pango.Language(language)
    def matches(range_list)
    def to_string()
Functions

    def pango.pango_language_from_string(language)
    def pango.pango_language_matches(language, range_list)
```

Description

A `pango.Language` object represents a language tag meeting the RFC−3066 standard. The `pango.Language` can be retrieved from a `pango.Context` by using the `pango.Context.get_language()` method or created using the pango.Language() constructor. Example RFC−3066 language tags include: "en−us", "fr", and "sgn−us−ma".

Constructor

```
    pango.Language(language>)
```

`language` :	a string representing a language tag
Returns :	a new `pango.Language` object

Note

This constructor is available in PyGTK 2.4 and above.

Creates a new `pango.Language` object from the RFC−3066 language tag specified by *`language`*. This constructor first canonicalizes the string in *`language`* by converting it to lowercase, mapping '_' to '−', and stripping all characters other than letters and '−'.

Methods

pango.Language.matches

```
def matches(range_list)
```

>range_list :	a list of language ranges, separated by ';' characters.
Returns :	`TRUE` if a match was found.

Note

This method is available in PyGTK 2.4 and above.

The `matches`() method returns `TRUE` if the language matches one of the language ranges in the list specified by *`range_list`*.A language tag is considered to match a range in the list if

- the range is '*'
- the range is exactly the same as the tag, or
- the range is a prefix of the tag, and the character after the matching portion of the tag is '–'

each range must either be '*', or a canonicalized RFC–3066 language range (see the pango.Language() constructor for more information).

pango.Language.matches

```
def to_string()
```

Returns :	the string representation of the language tag

Note

This method is available in PyGTK 2.4 and above.

The `to_string`() method returns a string representation of the canonicalized language tag. See the pango.Language() constructor for more information.

Functions

pango.pango_language_from_string

```
def pango.pango_language_from_string(language)
```

language :	a string representing a language tag
Returns :	a new `pango.Language` object

Note

This function is deprecated in PyGTK 2.4 and above. Use the pango.Language() constructor instead.

The `pango.pango_language_from_string`() function takes a RFC–3066 format language tag as a string (specified by *`language`*) and converts it to a `pango.Language` object. This function first canonicalizes the string by converting it to lowercase, mapping '_' to '–', and stripping all characters other than letters and '–'.

pango.pango_language_matches

```
    def pango.pango_language_matches()
```

language :	a language tag (see the `pango.pango_language_from_string()` function), `None` is allowed and matches nothing but '*'
range_list :	a list of language ranges, separated by ';' characters. each element must either be '*', or a RFC 3066 language range canonicalized as by the `pango.pango_language_from_string()` function.
Returns :	`TRUE` if a match was found.

Note

This function is deprecated in PyGTK 2.4 and above. Use the `matches()` method instead.

The `pango.pango_language_matches()` function checks if a language tag matches one of the elements in a list of language ranges. A language tag is considered to match a range in the list if the range is '*', the range is exactly the tag, or the range is a prefix of the tag, and the character after the tag is '–'.

Prev	Up	Next
pango.GlyphString	Home	pango.Layout

pango.Layout

Prev **The pango Class Reference** Next

pango.Layout

pango.Layout — an object representing a paragraph of text with attributes.

Synopsis

```
class pango.Layout(gobject.GObject):
    pango.Layout(context)
    def copy()
    def get_context()
    def set_attributes(attrs)
    def get_attributes()
    def set_text(text)
    def get_text()
    def set_markup(markup)
    def set_markup_with_accel(markup, accel_marker)
    def set_font_description(desc)
    def set_width(width)
    def get_width()
    def set_wrap(wrap)
    def get_wrap()
    def set_indent(indent)
    def get_indent()
    def set_spacing(spacing)
    def get_spacing()
    def set_justify(justify)
    def get_justify()
    def set_alignment(alignment)
    def get_alignment()
    def set_tabs(tabs)
    def get_tabs()
    def set_single_paragraph_mode(setting)
    def get_single_paragraph_mode()
    def context_changed()
    def index_to_pos(index)
    def get_cursor_pos(index)
    def move_cursor_visually(strong, old_index, old_trailing, direction)
    def xy_to_index(x, y)
    def get_extents()
    def get_pixel_extents()
    def get_size()
    def get_pixel_size()
    def get_line_count()
    def get_iter()
```

Ancestry

```
+-- gobject.GObject
  +-- pango.Layout
```

Description

A `pango.Layout` object represents a paragraph of text with a `pango.Context`, a UTF–8 text string and a set of attributes for that string. The set of formatted lines can be extracted from the object, the layout can be rendered, and conversion between logical character positions within the layout's text, and the physical position of the resulting glyphs can be made. Also there are a number of attributes that adjust the formatting of the layout.

Constructor

```
  pango.Layout(context)
```

context :	a `pango.Context`
Returns :	a new `pango.Layout`.

Creates a new `pango.Layout` object with attributes initialized to the default values of the `pango.Context` specified by *context*.

Methods

pango.Layout.copy

```
  def copy()
```

Returns :	a new `pango.Layout` that is a copy of the layout

The `copy()` method returns a `pango.Layout` that is a deep copy–by–value of the layout. The attribute list, tab array, and text from the layout are all copied by value.

pango.Layout.get_context

```
  def get_context()
```

Returns :	the `pango.Context` for the layout.

The `get_context()` method returns the `pango.Context` used for this layout.

pango.Layout.set_attributes

```
def set_attributes(attrs)
```

attrs :	a pango.AttrList

The set_attributes() method sets the pango.AttrList for the layout object to the value specified by *attrs*.

pango.Layout.get_attributes

```
def get_attributes()
```

Returns :	a pango.AttrList

The get_attributes() method returns the pango.AttrList for the layout, if any.

pango.Layout.set_text

```
def set_text(text)
```

text :	a UTF8–string

The set_text() method sets the text of the layout to the value specified by *text*.

pango.Layout.get_text

```
def get_text()
```

Returns :	the text in the layout

The get_text() method returns the text in the layout.

pango.Layout.set_markup

```
def set_markup(markup)
```

markup :	marked–up text

The set_markup() method is the same as the set_markup_with_accel() method but the markup text isn't scanned for accelerators.

pango.Layout.set_markup_with_accel

```
def set_markup_with_accel(markup, accel_marker)
```

markup :	some marked–up text (see the Pango Markup Language reference page)
accel_marker :	marker for accelerators in the text
Returns :	the accelerator character if any

The set_markup_with_accel() method sets the layout text and attribute list from marked–up text to the value specified by *markup_format* (see the Pango Markup Language reference page). The current text and attribute list of the layout are replaced. If *accel_marker* is nonzero the markup will be parsed for the marker and the character following the first marker becomes the accelerator character. For example, if the accelerator marker is an underscore, the character after the first underscore will be the accelerator character. All characters marked as an accelerator will be displayed with a pango.UNDERLINE_LOW attribute, and the accelerator character will be returned in *accel_char*. A literal *accel_marker* character can be put in the markup by using two *accel_marker* characters together.

pango.Layout.set_font_description

```
    def set_font_description(desc)
```

desc :	the new pango.FontDescription, or None to unset the current font description.

The set_font_description() method set the default pango.FontDescription for the layout to the value specified by *desc*. If no font description is set on the layout, the font description from the layout's context is used.

pango.Layout.set_width

```
    def set_width(width)
```

width :	the desired width, or −1 to indicate that no wrapping should be performed.

The set_width() method sets the wrap width for the lines of the pango.Layout to the value specified by *width*. If the value of *width* is −1 no wrapping should be performed.

pango.Layout.get_width

```
    def get_width()
```

Returns :	the width

The get_width() method returns the width at which the lines of the pango.Layout should be wrapped.

pango.Layout.set_wrap

```
    def set_wrap(wrap)
```

wrap :	the wrap mode

The set_wrap() method sets the wrap style to the value specified by *wrap*. The value of wrap must be one of:

pango.WRAP_WORD	Wrap lines at word boundaries.
pango.WRAP_CHAR	Wrap lines at character boundaries.

The wrap style is in effect if a width is set on the layout with the pango.Layout.set_width(). To turn off wrapping, set the width to −1.

pango.Layout.get_wrap

```
    def get_wrap()
```

Returns :	Active wrap mode.

The get_wrap() method returns the value of the wrap mode for the layout. See the set_wrap() method for more information.

pango.Layout.set_indent

```
    def set_indent(indent)
```

indent :	the amount by which to indent

The set_indent() method sets the indentation of the first line of the layout to the value specified by *indent*. The value of *indent* may be negative to provide a hanging indent.

pango.Layout.get_indent

```
def get_indent()
```

Returns :	the indent

The `get_indent()` method returns the amount of indentation of the first line of the layout.

pango.Layout.set_spacing

```
def set_spacing(spacing)
```

`spacing` :	the amount of spacing (in thousandths of a device unit)

The `set_spacing()` method sets the amount of spacing between the lines of the layout to the value specified by *`spacing`*.

pango.Layout.get_spacing

```
def get_spacing()
```

Returns :	the spacing (in thousandths of a device unit)

The `get_spacing()` method returns the amount of spacing between the lines of the layout.

pango.Layout.set_justify

```
def set_justify(justify)
```

`justify` :	if `TRUE` the lines in the layout should be justified.

The `set_justify()` method sets the justification attribute to the value of *`justify`*. If *`justify`* is `TRUE` each complete line should be stretched to fill the entire width of the layout. This stretching is typically done by adding whitespace, but for some scripts (such as Arabic), the justification is done by extending the characters.

pango.Layout.get_justify

```
def get_justify()
```

Returns :	`TRUE` if justification will be used

The `get_justify()` method returns `TRUE` if each complete line should be stretched to fill the entire width of the layout.

pango.Layout.set_alignment

```
def set_alignment(alignment)
```

`alignment` :	the new alignment

The `set_alignment()` method sets the alignment (how partial lines are positioned within the horizontal space available) for the layout to the value specified by *`alignment`*. The value of *`alignment`* must be one of:

`pango.ALIGN_LEFT`	Put all available space on the right
`pango.ALIGN_CENTER`	Center the line within the available space
`pango.ALIGN_RIGHT`	Put all available space on the left

pango.Layout.get_alignment

```
def get_alignment()
```

Returns :	the alignment value

The `get_alignment()` method returns the alignment (how partial lines are positioned within the horizontal space available) for the layout. See the `set_alignment()` method for more information.

pango.Layout.set_tabs

```
def set_tabs(tabs)
```

`tabs` :	a `pango.TabArray`

The `set_tabs()` method sets the tabs to the value specified by *`tabs`* thereby overriding the default tabs (every 8 spaces). If *`tabs`* is `None`, the default tabs are reinstated.

pango.Layout.get_tabs

```
def get_tabs()
```

Returns :	a copy of the tabs for this layout, or `None`

The `get_tabs()` method returns the current `pango.TabArray` used by this layout. If no `pango.TabArray` has been set, then the default tabs (every 8 spaces) are in use and `None` is returned

pango.Layout.set_single_paragraph_mode

```
def set_single_paragraph_mode(setting)
```

`setting` :	if `TRUE` newlines, etc. are not treated as paragraph separators.

The `set_single_paragraph_mode()` method sets the single paragraph mode attribute to the value specified by *`setting`*. If *`setting`* is `TRUE`, do not treat newlines and similar characters as paragraph separators; instead, keep all text in a single paragraph, and display a glyph for paragraph separator characters. Used when you want to allow editing of newlines on a single text line.

pango.Layout.get_single_paragraph_mode

```
def get_single_paragraph_mode()
```

Returns :	`TRUE` if the layout does not break paragraphs at paragraph separator characters

The `get_single_paragraph_mode()` method returns the value set by the `set_single_paragraph_mode()` method.

pango.Layout.context_changed

```
def context_changed()
```

The `context_changed()` method forces recomputation of any state in the `pango.Layout` that might depend on the layout's context. This method should be called if you make changes to the `pango.Context` subsequent to creating the layout.

pango.Layout.index_to_pos

```
    def index_to_pos(index)
```

`index` :	byte index within the layout
Returns :	a 4–tuple representing the grapheme's position

The `index_to_pos`() method converts from the specified *`index`* within a `pango.Layout` to the onscreen position corresponding to the grapheme at that index, which is represented as a 4–tuple (x, y, width, height). Note that `x` is always the leading edge of the grapheme and `x + width` the trailing edge of the grapheme. If the directionality of the grapheme is right–to–left, then `width` will be negative.

pango.Layout.get_cursor_pos

```
    def get_cursor_pos(index)
```

`index` :	the byte index of the cursor
Returns :	a 2–tuple containing two 4–tuples representing the strong and weak cursor positions

The `get_cursor_pos`() method returns a 2–tuple containing two 4–tuples representing the strong and weak cursor positions of the specified *`index`* within a layout. The position of each cursor is stored as a zero–width rectangle represented by a 4–tuple (`x`, `y`, `width`, `height`). The strong cursor location is the location where characters of the directionality equal to the base direction of the layout are inserted. The weak cursor location is the location where characters of the directionality opposite to the base direction of the layout are inserted.

pango.Layout.move_cursor_visually

```
    def move_cursor_visually(strong, old_index, old_trailing, direction)
```

`strong` :	if `TRUE` the moving cursor is the strong cursor; otherwise, the weak cursor. The strong cursor is the cursor corresponding to text insertion in the base direction for the layout.
`old_index` :	the byte index of the grapheme for the old index
`old_trailing` :	if 0, the cursor was at the trailing edge of the grapheme indicated by *`old_index`*, if > 0, the cursor was at the leading edge.
`direction` :	direction to move cursor. A negative value indicates motion to the left.
Returns :	a 2–tuple containing: the new cursor byte index (a value of –1 indicates that the cursor has been moved off the beginning of the layout while a value of G_MAXINT indicates that the cursor has been moved off the end of the layout); and, the number of characters to move forward (from the new cursor position) to get the position where the cursor should be displayed.

The `move_cursor_visually`() returns a 2–tuple containing:

- a new cursor position calculated from an old position (specified by *`old_index`*) and the specified *`direction`* to move visually
- the number of characters to move forward (from the new cursor position) to get the position where the cursor should be displayed. This allows distinguishing the position at the beginning of one line from the position at the end of the preceding line. the first value is always on the line where the cursor should be displayed.

If *`direction`* is positive, then the new strong cursor position will be one position to the right of the old cursor position. If *`direction`* is negative then the new strong cursor position will be one position to the left of the old cursor position.

In the presence of bidirectional text, the correspondence between logical and visual order will depend on the direction of the current run, and there may be jumps when the cursor is moved off of the end of a run.

Motion here is in cursor positions, not in characters, so a single call to the `move_cursor_visually()` method may move the cursor over multiple characters when multiple characters combine to form a single grapheme.

pango.Layout.xy_to_index

```
def xy_to_index(x, y)
```

x :	the X offset (in thousandths of a device unit) from the left edge of the layout.
y :	the Y offset (in thousandths of a device unit) from the top edge of the layout
Returns :	a 2−tuple containing the calculated byte index and an integer indicating where in the grapheme the user clicked (it will either be zero, or the number of characters in the grapheme − 0 represents the trailing edge of the grapheme).

The `xy_to_index()` method returns the byte index of the character at the specified *x* and *y* position within a layout. If the position is not inside the layout, the closest position is chosen (the (*x*, *y*) position will be clamped inside the layout).

pango.Layout.get_extents

```
def get_extents()
```

Returns :	a 2−tuple containing two 4−tuples representing the as drawn and logical extents rectangles of the layout

The `get_extents()` method returns a 2−tuple containing two 4−tuples representing the ink and logical extents rectangles of the layout in device units (one pixel = `pango.SCALE` device units). Logical extents are usually what you want for positioning things. The extents are given in layout coordinates which begin at the top left corner of the layout.

pango.Layout.get_pixel_extents

```
def get_pixel_extents()
```

Returns :	a 2−tuple containing two 4−tuples representing the as drawn (ink) and logical extents rectangles of the layout

The `get_pixel_extents()` method returns a 2−tuple containing two 4−tuples representing the logical and ink extents rectangles of the layout in pixel units. See the `get_extents()` method for more information. This method just calls the `get_extents()` and then converts the extents to pixels (one pixel = `pango.SCALE` device units).

pango.Layout.get_size

```
def get_size()
```

Returns :	a 2−tuple containing the logical width and height of the `pango.Layout`

The `get_size()` method returns a 2−tuple containing the logical width and height of the `pango.Layout` in pango device units (one pixel = `pango.SCALE` device units).

pango.Layout.get_pixel_size

```
def get_pixel_size()
```

Returns :	a 2–tuple containing the logical width height of the pango.Layout

The `get_pixel_size`() method returns a 2–tuple containing the logical width and height of the pango.Layout in pixels (one pixel = `pango.SCALE` device units). (The get_size() returns the width and height in device units.)

pango.Layout.get_line_count

```
def get_line_count()
```

Returns :	the line count

The `get_line_count`() method returns the count of lines in the layout.

pango.Layout.get_iter

```
def get_iter()
```

Returns :	a new pango.LayoutIter object

Note

This method is available in PyGTK 2.6 and above.

The `get_iter`() method returns a pango.LayoutIter object that can be used to iterate over the visual extents of the layout.

Prev	Up	Next
pango.Language	Home	pango.LayoutIter

pango.LayoutIter

Prev **The pango Class Reference** Next

pango.LayoutIter

pango.LayoutIter — an object used to iterate over the visual extents of a pango.Layout (new in PyGTK 2.6)

Synopsis

```
class pango.LayoutIter(gobject.GBoxed):
    def free()
    def next_char()
    def next_cluster()
    def next_line()
    def next_run()
    def at_last_line()
    def get_index()
    def get_baseline()
    def get_char_extents()
    def get_cluster_extents()
    def get_layout_extents()
    def get_line_extents()
    def get_run_extents()
    def get_line_yrange()
```

Ancestry

```
+-- gobject.GBoxed
  +-- pango.LayoutIter
```

Description

A `pango.LayoutIter` object can be used to iterate over the visual elements of a `pango.Layout`. A `pango.LayoutIter` is created using the `pango.Layout.get_iter()` method.

Methods

pango.LayoutIter.free

```
    def free()
```

Returns :	a new `pango.Layout` that is a copy of the layout

The `free()` method frees the `pango.LayoutIter` object.

pango.LayoutIter.next_char

```
    def next_char()
```

Returns :	`TRUE` if the iter was moved.

The `next_char()` method returns `TRUE` if the `pango.LayoutIter` is moved to the next character in visual order. If the iter was already at the end of the layout this method returns `FALSE`.

pango.LayoutIter.next_cluster

```
    def next_cluster()
```

Returns :	`TRUE` if the iter was moved.

The `next_cluster()` method returns `TRUE` if the `pango.LayoutIter` is moved to the next cluster in visual order. If the iter was already at the end of the layout this method returns `FALSE`.

pango.LayoutIter.next_line

```
    def next_line()
```

Returns :	`TRUE` if the iter was moved.

The `next_line()` method returns `TRUE` if the `pango.LayoutIter` is moved to the next line in visual order. If the iter was already at the end of the layout this method returns `FALSE`.

pango.LayoutIter.next_run

```
    def next_run()
```

Returns : `TRUE` if the iter was moved.

The `next_run`() method returns `TRUE` if the `pango.LayoutIter` is moved to the next run in visual order. If the iter was already at the end of the layout this method returns `FALSE`.

pango.LayoutIter.at_last_line

```
def at_last_line()
```

Returns : `TRUE` if the iter is in the last line.

The `at_last_line`() method returns `TRUE` if the `pango.LayoutIter` points to a position in the last line of the layout.

pango.LayoutIter.get_index

```
def get_index()
```

Returns : the current byte index

The `get_index`() method returns the current byte index. Note that iterating forward by char moves in visual order, not logical order, so indexes may not be sequential. Also, the index may be equal to the length of the text in the layout.

pango.LayoutIter.get_baseline

```
def get_baseline()
```

Returns : the baseline of the current line.

The `get_baseline`() method returns the y position of the current line's baseline, in layout coordinates (origin at top left of the entire layout).

pango.LayoutIter.get_char_extents

```
def get_char_extents()
```

Returns : a 4−tuple containing the logical extents of the character at the iter position.

The `get_char_extents`() method returns a 4−tuple (x, y, width, height) containing the logical extents of the current character, in layout coordinates (origin is the top left of the entire layout). Only logical extents can sensibly be obtained for characters; ink extents make sense only down to the level of clusters.

pango.LayoutIter.get_cluster_extents

```
def get_cluster_extents()
```

Returns : a 2−tuple containing containing the ink and logical extents as 4−tuples.

The `get_cluster_extents`() method returns a 2−tuple containing the ink and logical extents (as 4−tuples: x, y, width, height) of the cluster at the iter position.

pango.LayoutIter.get_layout_extents

```
def get_layout_extents()
```

Returns : a 2−tuple containing containing the ink and logical extents as 4−tuples.

The `get_layout_extents`() method returns a 2−tuple containing the ink and logical extents (as 4−tuples: x, y, width, height) of the layout at the iter position.

pango.LayoutIter.get_line_extents

```
    def get_line_extents()
```

Returns :	a 2−tuple containing containing the ink and logical extents as 4−tuples.

The `get_line_extents`() method returns a 2−tuple containing the ink and logical extents (as 4−tuples: x, y, width, height) of the line at the iter position.

pango.LayoutIter.get_run_extents

```
    def get_run_extents()
```

Returns :	a 2−tuple containing containing the ink and logical extents as 4−tuples.

The `get_run_extents`() method returns a 2−tuple containing the ink and logical extents (as 4−tuples: x, y, width, height) of the run at the iter position.

pango.LayoutIter.get_line_yrange

```
    def get_line_yrange()
```

Returns :	a 2−tuple containing the start and end of the layout line.

The `get_line_yrange`() method returns a 2−tuple containing the start and end y positions of the layout line. The vertical space in the `pango.Layout` associated with the iter is devided between the lines in the layout, the space belonging to the current line is returned in the 2−tuple. A line's range includes the line's logical extents, plus half of the spacing above and below the line, if the `pango.Layout.set_spacing()` method has been called to set the layout spacing. The y positions are in layout coordinates (origin at top left of the entire layout).

Prev	Up	Next
pango.Layout	Home	pango.TabArray

pango.TabArray

pango.TabArray — an object containing an array of tab stops.

Synopsis

```
class pango.TabArray(gobject.GBoxed):
    pango.TabArray(initial_size, positions_in_pixels)
    def copy()
    def get_size()
    def resize(new_size)
    def set_tab(tab_index, alignment, location)
    def get_tab(tab_index)
    def get_tabs()
    def get_positions_in_pixels()
```

Description

A pango.TabArray object contains an array of tab stops. Each tab stop has an alignment and a position.

Constructor

```
pango.TabArray(initial_size, positions_in_pixels)
```

initial_size :	Initial number of tab stops to allocate, can be 0
positions_in_pixels :	if `TRUE` the tab positions are in pixel units
Returns :	a pango.TabArray

Creates a new pango.TabArray object with the number of tab stops specified by *initial_size*. If *positions_in_pixels* is `TRUE`, the tab stop positions are specified in pixel units otherwise in pango units (one pixel = `pango.SCALE` pango units). All tab stops are initially at position 0.

Methods

pango.TabArray.copy

```
def copy()
```

Returns :	a new pango.TabArray object

The `copy`() method returns a new pango.TabArray that is copy of this pango.TabArray.

pango.TabArray.get_size

```
def get_size()
```

Returns :	the number of tab stops in the array.

The `get_size`() method returns the number of tab stops in the tab array.

pango.TabArray.resize

```
def resize(new_size)
```

new_size :	the new size of the array

The `resize`() method sets the size of the tab array to the value specified by *new_size*. You must subsequently initialize any tabs that were added to the array.

pango.TabArray.set_tab

```
def set_tab(tab_index, alignment, location)
```

tab_index :	the index of a tab stop
alignment :	the tab alignment
location :	the tab location in pango units

The `set_tab`() method sets the specified *alignment* and *location* of the tab stop specified by *tab_index*. The value of *alignment* must always be `pango.TAB_LEFT` in the current implementation.

pango.TabArray.get_tab

```
    def get_tab(tab_index)
```

`tab_index` :	the tab stop index
Returns :	a 2−tuple containing the tab alignment and position

The `get_tab`() method returns a 2−tuple containing the alignment and position of the tab stop specified by *`tab_index`*.

pango.TabArray.get_tabs

```
    def get_tabs()
```

Returns :	a tuple containing a list of 2−tuples (each holding the alignment and position of a tab stop).

The `get_tabs`() method returns a tuple containing a list of 2−tuples (each holding the alignment and position of a tab stop)

pango.TabArray.get_positions_in_pixels

```
    def get_positions_in_pixels()
```

Returns :	`TRUE` if tab stop positions are specified in pixels

The `get_positions_in_pixels`() method returns *`TRUE`* if the tab positions are specified in pixels and `FALSE` if they are in pango units.

Prev	Up	Next
pango.LayoutIter	Home	The Pango Markup Language

gtk.GenericCellRenderer

gtk.GenericCellRenderer — a TreeView cell renderer that helps create cell renderers in Python

Synopsis

```
class gtk.GenericCellRenderer(gtk.CellRenderer):
    gtk.GenericCellRenderer()
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.CellRenderer
      +-- gtk.GenericCellRenderer
```

Description

The `gtk.GenericCellRenderer` helps in the creation of `gtk.TreeView` cell renderers in Python. The `gtk.GenericCellRenderer` is subclassed to provide a new cell renderer that provides cell renderer behavior using methods with predefined names that are called by the `gtk.GenericCellRenderer` methods as required to provide the various cell renderer operations. The methods that need to be defined by the programmer in Python are:

```
def on_get_size(widget, cell_area)
 def on_render(window, widget, background_area, cell_area, expose_area,
flags)
 def on_activate(event, widget, path, background_area, cell_area, flags)
 def on_start_editing(event, widget, path, background_area, cell_area,
flags)
```

See the `gtk.CellRenderer` description for details of the above methods.

Constructor

```
  gtk.GenericCellRenderer()
```

Returns :	a new `gtk.GenericCellRenderer` object

Creates a new `gtk.GenericCellRenderer`

Prev	Up	Next
gtk.GammaCurve	Home	gtk.GenericTreeModel

gtk.GenericTreeModel

Prev	**The gtk Class Reference**	Next

gtk.GenericTreeModel

gtk.GenericTreeModel   a TreeView model that helps create tree models in Python

Synopsis

```
class gtk.GenericTreeModel(gobject.GObject, gtk.TreeModel):
    gtk.GenericTreeModel()
    def invalidate_iters()
    def iter_is_valid(iter)
```

Ancestry

```
+-- gobject.GObject
  +-- gtk.GenericTreeModel (implements gtk.TreeModel)
```

Properties

"leak−references"	Read−Write	If `TRUE`, creating a `gtk.TreeIter` will bump the reference count of the object used as the internal row reference. This may cause a memory leak but will prevent problems with objects being destroyed while still in use in a `gtk.TreeIter`. Set this to `FALSE` only if the model saves the objects used in tree iters.

Description

The gtk.GenericTreeModel helps in the creation of gtk.TreeView tree models in Python. The gtk.GenericTreeModel is subclassed to provide a new tree model that provides the tree model behavior using methods with predefined names that are called by the gtk.GenericTreeModel methods as required to provide the various tree model operations. The methods that need to be defined by the programmer in Python are:

```
def on_get_flags(self)
 def on_get_n_columns(self)
 def on_get_column_type(self, index)
 def on_get_iter(self, path)
 def on_get_path(self, rowref)
 def on_get_value(self, rowref, column)
 def on_iter_next(self, rowref)
 def on_iter_children(self, parent)
 def on_iter_has_child(self, rowref)
 def on_iter_n_children(self, rowref)
 def on_iter_nth_child(self, parent, n)
 def on_iter_parent(self, child)
```

See the gtk.TreeModel description for details of the above methods.

In PyGTK 2.4 and above the invalidate_iters() and iter_is_valid() methods are available to help manage the gtk.TreeIter objects and their Python object references. These are particularly useful when the "leak−references" property is set to FALSE.

The tree models derived from gtk.GenericTreeModel are protected from problems with out of date gtk.TreeIter objects because gtk.TreeIter objects are automatically checked for validity with the tree model.

If a custom tree model doesn't support persistent iters (i.e. gtk.TREE_MODEL_ITERS_PERSIST is not set in the return from the gtk.TreeModel.get_flags() method), it can call the invalidate_iters() method to invalidate all its outstanding gtk.TreeIter objects when it changes the model (e.g. after inserting a new row). The tree model can also dispose of any Python objects that it passed references to gtk.TreeIter objects after calling the invalidate_iters() method.

Applications can use the iter_is_valid() method to determine if a gtk.TreeIter is still valid for the custom tree model.

Constructor

```
gtk.GenericTreeModel()
```

Returns :	a new gtk.GenericTreeModel object

Creates a new gtk.GenericTreeModel object

Methods

gtk.GenericTreeModel.invalidate_iters

```
    def invalidate_iters()
```

Note

This method is available in PyGTK 2.4 and above.

The `invalidate_iters`() method invalidates all the `gtk.TreeIter` objects for the custom tree model.

gtk.GenericTreeModel.iter_is_valid

```
    def iter_is_valid(iter)
```

`iter` :	a `gtk.TreeIter`
Returns :	`TRUE` if *`iter`* is valid for the tree model; otherwise, `FALSE` is returned.

Note

This method is available in PyGTK 2.4 and above.

The `iter_is_valid`() method returns `TRUE` if the `gtk.TreeIter` specified by *`iter`* is valid for the custom tree model.

Prev	Up	Next
gtk.GenericCellRenderer	Home	gtk.HandleBox

	gtk.TreeModelRow	
Prev	**The gtk Class Reference**	Next

gtk.TreeModelRow

gtk.TreeModelRow — an object representing a row in a `gtk.TreeModel`

Synopsis

```
class gtk.TreeModelRow:
    def iterchildren()
```

Ancestry

```
+-- gobject.GBoxed
    +-- gtk.TreeModelRow
```

Attributes

"next"	Read	The next gtk.TreeModelRow or None
"parent"	Read	The parent gtk.TreeModelRow of this row or None
"model"	Read	The gtk.TreeModel that the row is part of.
"path"	Read	The tree path of the row
"iter"	Read	A gtk.TreeIter pointing at the row.

Description

A gtk.TreeModelRow object represents a row in a gtk.TreeModel. A gtk.TreeModelRow is created by taking the mapping of a gtk.TreeModel. For example:

```
  treemodelrow = liststore[0]
  treemodelrow = liststore[(0,)]
  treemodelrow = liststore['0']
```

all create a gtk.TreeModelRow for the first row in *liststore*. The gtk.TreeModelRow implements some of the Python sequence protocol that makes the row behave like a sequence of objects. Specifically a tree model row has the capability of:

- getting and setting column values,
- returning a tuple or list containing the column values, and
- getting the number of values in the row i.e. the number of columns

For example to get and set the value in the second column of a row, you could do the following:

```
  value = treemodelrow[1]
  treemodelrow[1] = value
```

You can use the Python `len`() function to get the number of columns in the row and you can retrieve all the column values as a list (tuple) using the Python `list`() (`tuple`()) function.

The gtk.TreeModelRow supports one method: the iterchildren() method that returns a gtk.TreeModelRowIter for iterating over the children of the row.

Methods

gtk.TreeModelRow.iterchildren

```
   def iterchildren()
```

Returns :	a gtk.TreeModelRowIter for the row's children or None

The `iterchildren`() method returns a gtk.TreeModelRowIter for iterating over the children of the row or None if the row has no children.

Prev Up Next
gtk.TreeModelSort Home gtk.TreeModelRowIter

gtk.TreeModelRowIter

Prev **The gtk Class Reference** Next

gtk.TreeModelRowIter

gtk.TreeModelRowIter an object for iterating over a set of gtk.TreeModelRow objects.

Synopsis

```
class gtk.TreeModelRowIter:
    def next()
```

Ancestry

```
+-- gobject.GBoxed
    +-- gtk.TreeModelRowIter
```

Description

A gtk.TreeModelRowIter is an object that implements the Python Iterator protocol. It provides the means to iterate over a set of gtk.TreeModelRow objects in a gtk.TreeModel. A gtk.TreeModelRowIter is created by calling the Python iter() function on a gtk.TreeModel object:

```
  treemodelrowiter = iter(treestore)
```

or, calling the gtk.TreeModelRow.iterchildren() method to iterate over its child rows.

Each time you call the next() method it returns the next sibling gtk.TreeModelRow . When there are no rows left the StopIteration exception is raised. Note that a gtk.TreeModelRowIter does not iterate over the child rows of the rows it is iterating over. You'll have to use the gtk.TreeModelRow.iterchildren() method to retrieve an iterator for the child rows.

Methods

gtk.TreeModelRowIter.next

```
  def next()
```

Returns :	the next gtk.TreeModelRow

The next() method returns the next gtk.TreeModelRow in the set of rows it is iterating over. When there are no more rows left the StopIteration exception is raised.

Prev Up Next
gtk.TreeModelRow Home gtk.TreeRowReference
The gtk.gdk Class Reference
Prev Next

The gtk.gdk Class Reference

Table of Contents

Prev Up Next
gtk Constants Home gtk.gdk.Atom
gtk.gdk Constants
Prev **The gtk.gdk Class Reference** Next

gtk.gdk Constants

gtk.gdk Constants the built–in constants of the gtk.gdk module

Synopsis

```
GDK Drag Action Constants
GDK Device Axis Use Constants
GDK Byte Order Constants
GDK Cap Style Constants
GDK Crossing Mode Constants
GDK Cursor Type Constants
GDK Drag Protocol Constants
GDK Event Mask Flag Constants
GDK Event Type Constants
GDK Extension Mode Constants
GDK Fill Constants
GDK Fill Rule Constants
GDK Filter Return Constants
GDK Function Constants
GDK GC Values Mask Flag Constants
GDK Gravity Constants
GDK Image Type Constants
GDK Input Condition Flag Constants
GDK Input Mode Constants
```

Description

GDK Drag Action Constants

The Drag Action constants are used by `gtk.gdk.DragContext` objects to indicate what the destination should do with the dropped data.

`gtk.gdk.ACTION_DEFAULT`	
`gtk.gdk.ACTION_COPY`	Copy the data.
`gtk.gdk.ACTION_MOVE`	Move the data, i.e. first copy it, then delete it from the source using the DELETE target of the X selection protocol.
`gtk.gdk.ACTION_LINK`	Add a link to the data. Note that this is only useful if source and destination agree on what it means.
`gtk.gdk.ACTION_PRIVATE`	Special action which tells the source that the destination will do something that the source doesn't understand.
`gtk.gdk.ACTION_ASK`	Ask the user what to do with the data.

GDK Device Axis Use Constants

The Device Axis constants describing the way in which a device axis (valuator) maps onto predefined valuator types.

`gtk.gdk.AXIS_IGNORE`	the axis is ignored.
`gtk.gdk.AXIS_X`	the axis is used as the x axis.
`gtk.gdk.AXIS_Y`	the axis is used as the y axis.
`gtk.gdk.AXIS_PRESSURE`	the axis is used for pressure information.
`gtk.gdk.AXIS_XTILT`	the axis is used for x tilt information.
`gtk.gdk.AXIS_YTILT`	the axis is used for y tilt information.
`gtk.gdk.AXIS_WHEEL`	the axis is used for wheel information.
`gtk.gdk.AXIS_LAST`	a constant equal to the numerically highest axis value.

GDK Byte Order Constants

The Byte Order constants specify a set of values describing the possible byte−orders for storing pixel values in memory.

`gtk.gdk.LSB_FIRST`	The values are stored with the least−significant byte first. For instance, the 32−bit value 0xffeecc would be stored in memory as 0xcc, 0xee, 0xff, 0x00.
`gtk.gdk.MSB_FIRST`	The values are stored with the most−significant byte first. For instance, the 32−bit value 0xffeecc would be stored in memory as 0x00, 0xcc, 0xee, 0xff.

GDK Cap Style Constants

The Cap Style constants specify how the end of lines are drawn.

`gtk.gdk.CAP_NOT_LAST`	The same as `gtk.gdk.CAP_BUTT` for lines of non−zero width but for zero width lines, the final point on the line will not be drawn.
`gtk.gdk.CAP_BUTT`	The ends of the lines are drawn squared off and extending to the coordinates of the end point.
`gtk.gdk.CAP_ROUND`	The ends of the lines are drawn as semicircles with the diameter equal to the line width and centered at the end point.
`gtk.gdk.CAP_PROJECTING`	The ends of the lines are drawn squared off and extending half the width of the line beyond the end point.

GDK Crossing Mode Constants

The Crossing Mode constants specify the crossing mode for the Crossing `gtk.gdk.Event`

GDK Cursor Type Constants

The Cursor Type constants specify the set of standard cursors available.

`gtk.gdk.X_CURSOR`	✖
`gtk.gdk.ARROW`	➶
`gtk.gdk.BASED_ARROW_DOWN`	⤓
`gtk.gdk.BASED_ARROW_UP`	⤒
`gtk.gdk.BOAT`	⛵
`gtk.gdk.BOGOSITY`	#
`gtk.gdk.BOTTOM_LEFT_CORNER`	↙
`gtk.gdk.BOTTOM_RIGHT_CORNER`	↘
`gtk.gdk.BOTTOM_SIDE`	⤓
`gtk.gdk.BOTTOM_TEE`	⊥
`gtk.gdk.BOX_SPIRAL`	▣
`gtk.gdk.CENTER_PTR`	↑
`gtk.gdk.CIRCLE`	○
`gtk.gdk.CLOCK`	🕒
`gtk.gdk.COFFEE_MUG`	☕

`gtk.gdk.CROSS`	
`gtk.gdk.CROSS_REVERSE`	
`gtk.gdk.CROSSHAIR`	
`gtk.gdk.DIAMOND_CROSS`	
`gtk.gdk.DOT`	
`gtk.gdk.DOTBOX`	
`gtk.gdk.DOUBLE_ARROW`	
`gtk.gdk.DRAFT_LARGE`	
`gtk.gdk.DRAFT_SMALL`	
`gtk.gdk.DRAPED_BOX`	
`gtk.gdk.EXCHANGE`	
`gtk.gdk.FLEUR`	
`gtk.gdk.GOBBLER`	
`gtk.gdk.GUMBY`	
`gtk.gdk.HAND1`	
`gtk.gdk.HAND2`	
`gtk.gdk.HEART`	
`gtk.gdk.ICON`	
`gtk.gdk.IRON_CROSS`	
`gtk.gdk.LEFT_PTR`	
`gtk.gdk.LEFT_SIDE`	
`gtk.gdk.LEFT_TEE`	
`gtk.gdk.LEFTBUTTON`	
`gtk.gdk.LL_ANGLE`	
`gtk.gdk.LR_ANGLE`	
`gtk.gdk.MAN`	
`gtk.gdk.MIDDLEBUTTON`	
`gtk.gdk.MOUSE`	
`gtk.gdk.PENCIL`	
`gtk.gdk.PIRATE`	
`gtk.gdk.PLUS`	
`gtk.gdk.QUESTION_ARROW`	
`gtk.gdk.RIGHT_PTR`	
`gtk.gdk.RIGHT_SIDE`	
`gtk.gdk.RIGHT_TEE`	
`gtk.gdk.RIGHTBUTTON`	
`gtk.gdk.RTL_LOGO`	
`gtk.gdk.SAILBOAT`	
`gtk.gdk.SB_DOWN_ARROW`	
`gtk.gdk.SB_H_DOUBLE_ARROW`	
`gtk.gdk.SB_LEFT_ARROW`	
`gtk.gdk.SB_RIGHT_ARROW`	
`gtk.gdk.SB_UP_ARROW`	
`gtk.gdk.SB_V_DOUBLE_ARROW`	
`gtk.gdk.SHUTTLE`	

`gtk.gdk.SIZING`	⧉
`gtk.gdk.SPIDER`	✳
`gtk.gdk.SPRAYCAN`	🧴
`gtk.gdk.STAR`	☆
`gtk.gdk.TARGET`	◎
`gtk.gdk.TCROSS`	+
`gtk.gdk.TOP_LEFT_ARROW`	↖
`gtk.gdk.TOP_LEFT_CORNER`	⇱
`gtk.gdk.TOP_RIGHT_CORNER`	↗
`gtk.gdk.TOP_SIDE`	⤒
`gtk.gdk.TOP_TEE`	┬
`gtk.gdk.TREK`	♙
`gtk.gdk.UL_ANGLE`	┌
`gtk.gdk.UMBRELLA`	☂
`gtk.gdk.UR_ANGLE`	┐
`gtk.gdk.WATCH`	⌚
`gtk.gdk.XTERM`	⌶

GDK Drag Protocol Constants

The Drag Protocol constants specify the protocol for a `gtk.gdk.DragContext` according to which `DND` is done.

`gtk.gdk.DRAG_PROTO_MOTIF`	The Motif DND protocol.
`gtk.gdk.DRAG_PROTO_XDND`	The Xdnd protocol.
`gtk.gdk.DRAG_PROTO_ROOTWIN`	An extension to the Xdnd protocol for unclaimed root window drops.
`gtk.gdk.DRAG_PROTO_NONE`	no protocol.
`gtk.gdk.DRAG_PROTO_WIN32_DROPFILES`	The simple WM_DROPFILES protocol.
`gtk.gdk.DRAG_PROTO_OLE2`	The complex OLE2 DND protocol (not implemented).
`gtk.gdk.DRAG_PROTO_LOCAL`	Intra–application DND.

GDK Event Mask Flag Constants

The Event Mask flag constants are a set of bit–flags that specify the events a window is to receive. Most of these masks map onto one or more of the Event Type Constants.

`gtk.gdk.EXPOSURE_MASK`
`gtk.gdk.POINTER_MOTION_MASK`
`gtk.gdk.POINTER_MOTION_HINT_MASK`
`gtk.gdk.BUTTON_MOTION_MASK`
`gtk.gdk.BUTTON1_MOTION_MASK`
`gtk.gdk.BUTTON2_MOTION_MASK`
`gtk.gdk.BUTTON3_MOTION_MASK`
`gtk.gdk.BUTTON_PRESS_MASK`
`gtk.gdk.BUTTON_RELEASE_MASK`
`gtk.gdk.KEY_PRESS_MASK`

	gtk.gdk.KEY_RELEASE_MASK
	gtk.gdk.ENTER_NOTIFY_MASK
	gtk.gdk.LEAVE_NOTIFY_MASK
	gtk.gdk.FOCUS_CHANGE_MASK
	gtk.gdk.STRUCTURE_MASK
	gtk.gdk.PROPERTY_CHANGE_MASK
	gtk.gdk.VISIBILITY_NOTIFY_MASK
	gtk.gdk.PROXIMITY_IN_MASK
	gtk.gdk.PROXIMITY_OUT_MASK
	gtk.gdk.SUBSTRUCTURE_MASK
	gtk.gdk.SCROLL_MASK
	gtk.gdk.ALL_EVENTS_MASK

GDK Event Type Constants

The Event Type constants specify the type of an event.

`gtk.gdk.NOTHING`	a special code to indicate a null event.
`gtk.gdk.DELETE`	the window manager has requested that the toplevel window be hidden or destroyed, usually when the user clicks on a special icon in the title bar.
`gtk.gdk.DESTROY`	the window has been destroyed.
`gtk.gdk.EXPOSE`	all or part of the window has become visible and needs to be redrawn.
`gtk.gdk.MOTION_NOTIFY`	the pointer (usually a mouse) has moved.
`gtk.gdk.BUTTON_PRESS`	a mouse button has been pressed.
`gtk.gdk._2BUTTON_PRESS`	a mouse button has been double−clicked (clicked twice within a short period of time). Note that each click also generates a `gtk.gdk.BUTTON_PRESS` event.
`gtk.gdk._3BUTTON_PRESS`	a mouse button has been clicked 3 times in a short period of time. Note that each click also generates a `gtk.gdk.BUTTON_PRESS` event.
`gtk.gdk.BUTTON_RELEASE`	a mouse button has been released.
`gtk.gdk.KEY_PRESS`	a key has been pressed.
`gtk.gdk.KEY_RELEASE`	a key has been released.
`gtk.gdk.ENTER_NOTIFY`	the pointer has entered the window.
`gtk.gdk.LEAVE_NOTIFY`	the pointer has left the window.
`gtk.gdk.FOCUS_CHANGE`	the keyboard focus has entered or left the window.
`gtk.gdk.CONFIGURE`	the size, position or stacking order of the window has changed. Note that `PyGTK` discards these events for `gtk.gdk.WINDOW_CHILD` windows.
`gtk.gdk.MAP`	the window has been mapped.
`gtk.gdk.UNMAP`	the window has been unmapped.
`gtk.gdk.PROPERTY_NOTIFY`	a property on the window has been changed or deleted.
`gtk.gdk.SELECTION_CLEAR`	the application has lost ownership of a selection.
`gtk.gdk.SELECTION_REQUEST`	another application has requested a selection.
`gtk.gdk.SELECTION_NOTIFY`	a selection has been received.
`gtk.gdk.PROXIMITY_IN`	an input device has moved into contact with a sensing surface (e.g. a touchscreen or graphics tablet).

`gtk.gdk.PROXIMITY_OUT`	an input device has moved out of contact with a sensing surface.
`gtk.gdk.DRAG_ENTER`	the mouse has entered the window while a drag is in progress.
`gtk.gdk.DRAG_LEAVE`	the mouse has left the window while a drag is in progress
`gtk.gdk.DRAG_MOTION`	the mouse has moved in the window while a drag is in progress.
`gtk.gdk.DRAG_STATUS`	the status of the drag operation initiated by the window has changed.
`gtk.gdk.DROP_START`	a drop operation onto the window has started.
`gtk.gdk.DROP_FINISHED`	the drop operation initiated by the window has completed.
`gtk.gdk.CLIENT_EVENT`	a message has been received from another application.
`gtk.gdk.VISIBILITY_NOTIFY`	the window visibility status has changed.
`gtk.gdk.NO_EXPOSE`	indicates that the source region was completely available when parts of a drawable were copied. This is not very useful.
`gtk.gdk.SCROLL`	a scroll had occurred for a window
`gtk.gdk.WINDOW_STATE`	the window state has changed
`gtk.gdk.SETTING`	a setting has changed

GDK Extension Mode Constants

The Extension Mode constants specify which extension events are desired for a particular widget.

`gtk.gdk.EXTENSION_EVENTS_NONE`	No extension events are desired.
`gtk.gdk.EXTENSION_EVENTS_ALL`	All extension events are desired.
`gtk.gdk.EXTENSION_EVENTS_CURSOR`	Extension events are desired only if a cursor will be displayed for the device.

GDK Fill Constants

The Fill constants specify how primitives are drawn.

`gtk.gdk.SOLID`	draw with the foreground color.
`gtk.gdk.TILED`	draw with a tiled pixmap.
`gtk.gdk.STIPPLED`	draw using the stipple bitmap. Pixels corresponding to bits in the stipple bitmap that are set will be drawn in the foreground color; pixels corresponding to bits that are not set will be left untouched.
`gtk.gdk.OPAQUE_STIPPLED`	draw using the stipple bitmap. Pixels corresponding to bits in the stipple bitmap that are set will be drawn in the foreground color; pixels corresponding to bits that are not set will be drawn with the background color.

GDK Fill Rule Constants

The Fill Rule constants specify the method for determining which pixels are included in a region, when creating a GdkRegion from a polygon. The fill rule is only relevant for polygons which overlap themselves. Not used in PyGTK.

`gtk.gdk.EVEN_ODD_RULE`	Areas which are overlapped an odd number of times are included in the region, while areas overlapped an even number of times are not.
`gtk.gdk.WINDING_RULE`	Overlapping areas are always included.

GDK Filter Return Constants

The Filter Return constants specify the result of filtering a native event. See the `gtk.gdk.Window.add_filter()` method for more information.

`gtk.gdk.FILTER_CONTINUE`	Event not handled, continue processing.
`gtk.gdk.FILTER_TRANSLATE`	Native event translated and stored into the `gtk.gdk.Event` passed in.
`gtk.gdk.FILTER_REMOVE`	Event handled, terminate processing.

GDK Function Constants

The Function constants specify how the bit values for the source pixels are combined with the bit values for destination pixels to produce the final result. The sixteen values here correspond to the 16 different possible 2x2 truth tables. Only a couple of these values are usually useful; for colored images, only `gtk.gdk.COPY`, `gtk.gdk.XOR` and `gtk.gdk.INVERT` are generally useful. For bitmaps, `gtk.gdk.AND` and `gtk.gdk.OR` are also useful.

	`gtk.gdk.COPY`
	`gtk.gdk.INVERT`
	`gtk.gdk.XOR`
	`gtk.gdk.CLEAR`
	`gtk.gdk.AND`
	`gtk.gdk.AND_REVERSE.`
	`gtk.gdk.AND_INVERT`
	`gtk.gdk.NOOP`
	`gtk.gdk.OR`
	`gtk.gdk.EQUIV`
	`gtk.gdk.OR_REVERSE`
	`gtk.gdk.COPY_INVERT`
	`gtk.gdk.OR_INVERT`
	`gtk.gdk.NAND`
	`gtk.gdk.NOR`
	`gtk.gdk.SET`

GDK GC Values Mask Flag Constants

The GC Values Mask flag constants are a set of bit flags used to specify which fields GdkGCValues structure are set. These are only used internally by PyGTK.

`gtk.gdk.GC_FOREGROUND`	the foreground is set.
`gtk.gdk.GC_BACKGROUND`	the background is set.
`gtk.gdk.GC_FONT`	the font is set.
`gtk.gdk.GC_FUNCTION`	the function is set.
`gtk.gdk.GC_FILL`	the fill is set.
`gtk.gdk.GC_TILE`	the tile is set.
`gtk.gdk.GC_STIPPLE`	the stipple is set.
`gtk.gdk.GC_CLIP_MASK`	the clip_mask is set.
`gtk.gdk.GC_SUBWINDOW`	the subwindow_mode is set.

`gtk.gdk.GC_TS_X_ORIGIN`	the ts_x_origin is set.
`gtk.gdk.GC_TS_Y_ORIGIN`	the ts_y_origin is set.
`gtk.gdk.GC_CLIP_X_ORIGIN`	the clip_x_origin is set.
`gtk.gdk.GC_CLIP_Y_ORIGIN`	the clip_y_origin is set.
`gtk.gdk.GC_EXPOSURES`	the graphics_exposures is set.
`gtk.gdk.GC_LINE_WIDTH`	the line_width is set.
`gtk.gdk.GC_LINE_STYLE`	the line_style is set.
`gtk.gdk.GC_CAP_STYLE`	the cap_style is set.
`gtk.gdk.GC_JOIN_STYLE`	the join_style is set.

GDK Gravity Constants

The Gravity constants specify the reference point of a window and the meaning of coordinates passed to the `gtk.Window.move()` method.

`gtk.gdk.GRAVITY_NORTH_WEST`	The reference point is at the top left corner.
`gtk.gdk.GRAVITY_NORTH`	The reference point is in the middle of the top edge.
`gtk.gdk.GRAVITY_NORTH_EAST`	The reference point is at the top right corner.
`gtk.gdk.GRAVITY_WEST`	The reference point is at the middle of the left edge.
`gtk.gdk.GRAVITY_CENTER`	The reference point is at the center of the window.
`gtk.gdk.GRAVITY_EAST`	The reference point is at the middle of the right edge.
`gtk.gdk.GRAVITY_SOUTH_WEST`	The reference point is at the lower left corner.
`gtk.gdk.GRAVITY_SOUTH`	The reference point is at the middle of the lower edge.
`gtk.gdk.GRAVITY_SOUTH_EAST`	The reference point is at the lower right corner.
`gtk.gdk.GRAVITY_STATIC`	The reference point is at the top left corner of the window itself, ignoring window manager decorations.

GDK Image Type Constants

The Image Type constants specify the type of a `gtk.gdk.Image`.

`gtk.gdk.IMAGE_NORMAL`	The original X image type, which is quite slow since the image has to be transferred from the client to the server to display it.
`gtk.gdk.IMAGE_SHARED`	A faster image type, which uses shared memory to transfer the image data between client and server. However this will only be available if client and server are on the same machine and the shared memory extension is supported by the server.
`gtk.gdk.IMAGE_FASTEST`	Specifies that gtk.gdk.IMAGE_SHARED should be tried first, and if that fails then gtk.gdk.IMAGE_NORMAL will be used.

GDK Input Condition Flag Constants

The Input Condition constants are a set of bit–flags that specify conditions for which an input callback will be triggered. The three members of this enumeration correspond to the *`readfds`*, *`writefds`*, and *`exceptfds`* arguments to the `select` system call.

`gtk.gdk.INPUT_READ`	The file descriptor has become available for reading. (Or, as is standard in Unix, a socket or pipe was closed at the other end; this is the case if a subsequent read on the file descriptor returns a count of zero.)

`gtk.gdk.INPUT_WRITE`	The file descriptor has become available for writing.
`gtk.gdk.INPUT_EXCEPTION`	An exception was raised on the file descriptor.

GDK Input Mode Constants

The Input Mode constants specify the mode of an input device.

`gtk.gdk.MODE_DISABLED`	the device is disabled and will not report any events.
`gtk.gdk.MODE_SCREEN`	the device is enabled. The device's coordinate space maps to the entire screen.
`gtk.gdk.MODE_WINDOW`	the device is enabled. The device's coordinate space is mapped to a single window. The manner in which this window is chosen is undefined, but it will typically be the same way in which the focus window for key events is determined.

GDK Input Source Constants

The Input Source constants specify the type of an input device in general terms.

`gtk.gdk.SOURCE_MOUSE`	the device is a mouse. (This will be reported for the core pointer, even if it is something else, such as a trackball.)
`gtk.gdk.SOURCE_PEN`	the device is a stylus of a graphics tablet or similar device.
`gtk.gdk.SOURCE_ERASER`	the device is an eraser. Typically, this would be the other end of a stylus on a graphics tablet.
`gtk.gdk.SOURCE_CURSOR`	the device is a graphics tablet "puck" or similar device.

GDK Join Style Constants

The Join Style constants specify how the joins between segments of a polygon are drawn.

`gtk.gdk.JOIN_MITER`	The sides of each line are extended to meet at an angle.
`gtk.gdk.JOIN_ROUND`	The sides of the two lines are joined by a circular arc.
`gtk.gdk.JOIN_BEVEL`	The sides of the two lines are joined by a straight line which makes an equal angle with each line.

GDK Line Style Constants

The Line Style constants specify how lines are drawn.

`gtk.gdk.LINE_SOLID`	Lines are drawn solid.
`gtk.gdk.LINE_ON_OFF_DASH`	Lines are drawn dashed where even segments are drawn but odd segments are not drawn.
`gtk.gdk.LINE_DOUBLE_DASH`	Lines are drawn dashed where even segments are drawn normally but odd segments are drawn in the background color if the fill style is `gtk.gdk.SOLID`, or in the background color masked by the stipple if the fill style is `gtk.gdk.STIPPLED`.

GDK Modifier Constants

The Modifier constants are a set of bit−flags to indicate the state of modifier keys and mouse buttons in various event types. Typical modifier keys are **Shift**, **Control**, **Meta**, **Super**, **Hyper**, **Alt**, **Compose**, **Apple**, **CapsLock** or **ShiftLock**.

`gtk.gdk.SHIFT_MASK`	The Shift key.
`gtk.gdk.LOCK_MASK`	A Lock key (depending on the modifier mapping of the X server this may either be CapsLock or ShiftLock).
`gtk.gdk.CONTROL_MASK`	The Control key.
`gtk.gdk.MOD1_MASK`	The fourth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier, but normally it is the Alt key).
`gtk.gdk.MOD2_MASK`	The fifth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
`gtk.gdk.MOD3_MASK`	The sixth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
`gtk.gdk.MOD4_MASK`	The seventh modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
`gtk.gdk.MOD5_MASK`	The eighth modifier key (it depends on the modifier mapping of the X server which key is interpreted as this modifier).
`gtk.gdk.BUTTON1_MASK`	The first mouse button.
`gtk.gdk.BUTTON2_MASK`	The second mouse button.
`gtk.gdk.BUTTON3_MASK`	The third mouse button.
`gtk.gdk.BUTTON4_MASK`	The fourth mouse button.
`gtk.gdk.BUTTON5_MASK`	The fifth mouse button.
`gtk.gdk.RELEASE_MASK`	Differentiates between (keyval, modifiers) pairs from key press and release events.
`gtk.gdk.MODIFIER_MASK`	all of the above

GDK Notify Type Constants

The Notify Type constants specify the kind of crossing for the Crossing `gtk.gdk.Event`

`gtk.gdk.NOTIFY_ANCESTOR`	the window is entered from an ancestor or left toward an ancestor.
`gtk.gdk.NOTIFY_VIRTUAL`	the pointer moves between an ancestor and an inferior of the window.
`gtk.gdk.NOTIFY_INFERIOR`	the window is entered from an inferior or left toward an inferior.
`gtk.gdk.NOTIFY_NONLINEAR`	the window is entered from or left toward a window which is neither an ancestor nor an inferior.
`gtk.gdk.NOTIFY_NONLINEAR_VIRTUAL`	the pointer moves between two windows which are not ancestors of each other and the window is part of the ancestor chain between one of these windows and their least common ancestor.
`gtk.gdk.NOTIFY_UNKNOWN`	???

GDK Overlap Type Constants

The Overlap Type constants are not currently useful by PyGTK.

`gtk.gdk.OVERLAP_RECTANGLE_IN`	The rectangle is inside the GdkRegion.
`gtk.gdk.OVERLAP_RECTANGLE_OUT`	The rectangle is outside the GdkRegion.
`gtk.gdk.OVERLAP_RECTANGLE_PART`	The rectangle is partly inside the GdkRegion.

GDK Property Mode Constants

The Property Mode constants specify how existing data is combined with new data when using the the `gtk.gdk.Window.property_change()` method.

`gtk.gdk.PROP_MODE_REPLACE`	The new data replaces the existing data.
`gtk.gdk.PROP_MODE_PREPEND`	The new data is prepended to the existing data.
`gtk.gdk.PROP_MODE_APPEND`	The new data is appended to the existing data.

GDK Property State Constants

The Property State constants specify the type of a property change for a Property `gtk.gdk.Event`

`gtk.gdk.PROPERTY_NEW_VALUE`	the property value was changed.
`gtk.gdk.PROPERTY_DELETE`	the property was deleted.

GDK RGB Dither Constants

The RGB Dither constants specify the type of dithering to apply to the image on display.

`gtk.gdk.RGB_DITHER_NONE`	Never use dithering.
`gtk.gdk.RGB_DITHER_NORMAL`	Use dithering in 8 bits per pixel (and below) only.
`gtk.gdk.RGB_DITHER_MAX`	Use dithering in 16 bits per pixel and below.

GDK Scroll Direction Constants

The Scroll Direction constants specify the direction for the Scroll.`gtk.gdk.Event`

`gtk.gdk.SCROLL_UP`	the window is scrolled up.
`gtk.gdk.SCROLL_DOWN`	the window is scrolled down.
`gtk.gdk.SCROLL_LEFT`	the window is scrolled to the left.
`gtk.gdk.SCROLL_RIGHT`	the window is scrolled to the right.

GDK Setting Action Constants

The Setting Action constants specify the kind of modification applied to a setting in a Setting `gtk.gdk.Event`

`gtk.gdk.SETTING_ACTION_NEW`	A setting was added.
`gtk.gdk.SETTING_ACTION_CHANGED`	A setting was changed.
`gtk.gdk.SETTING_ACTION_DELETED`	A setting was deleted.

GDK Subwindow Mode Constants

The Subwindow Mode constants specify how drawing onto a window will affect child windows of that window.

`gtk.gdk.CLIP_BY_CHILDREN`	only draw onto the window itself not the subwindows.
`gtk.gdk.INCLUDE_INFERIORS`	draw onto the window and child windows.

GDK Visibility State Constants

The Visibility State constants specify the visibility status of a window for the Visibility `gtk.gdk.Event`

`gtk.gdk.VISIBILITY_UNOBSCURED`	the window is completely visible.
`gtk.gdk.VISIBILITY_PARTIAL`	the window is partially visible.
`gtk.gdk.VISIBILITY_FULLY_OBSCURED`	the window is not visible at all.

GDK Visual Type Constants

The Visual Type constants specify a set of values that describe the manner in which the pixel values for a visual are converted into RGB values for display.

`gtk.gdk.VISUAL_STATIC_GRAY`	Each pixel value indexes a grayscale value directly.
`gtk.gdk.VISUAL_GRAYSCALE`	Each pixel is an index into a color map that maps pixel values into grayscale values. The color map can be changed by an application.
`gtk.gdk.VISUAL_STATIC_COLOR`	Each pixel value is an index into a predefined, unmodifiable color map that maps pixel values into RGB values.
`gtk.gdk.VISUAL_PSEUDO_COLOR`	Each pixel is an index into a color map that maps pixel values into rgb values. The color map can be changed by an application.
`gtk.gdk.VISUAL_TRUE_COLOR`	Each pixel value directly contains red, green, and blue components. The red_mask, green_mask, and blue_mask fields of the `gtk.gdk.Visual` structure describe how the components are assembled into a pixel value.
`gtk.gdk.VISUAL_DIRECT_COLOR`	Each pixel value contains red, green, and blue components as for `gtk.gdk.TRUE_COLOR`, but the components are mapped via a color table into the final output table instead of being converted directly.

GDK Window Class Constants

The Window Class constants specify the class of window. `gtk.gdk.INPUT_OUTPUT` windows are the standard kind of window you might expect. `gtk.gdk.INPUT_ONLY` windows are invisible; they are used to trap events, but you can't draw on them.

`gtk.gdk.INPUT_OUTPUT`	A window for graphics and events.
`gtk.gdk.INPUT_ONLY`	A window for events only.

GDK Window Edge Constants

The Window Edge constants specify window edge or corner.

`gtk.gdk.WINDOW_EDGE_NORTH_WEST`	The top left corner.
`gtk.gdk.WINDOW_EDGE_NORTH`	The top edge.
`gtk.gdk.WINDOW_EDGE_NORTH_EAST`	The top right corner.
`gtk.gdk.WINDOW_EDGE_WEST`	The left edge.
`gtk.gdk.WINDOW_EDGE_EAST`	The right edge.
`gtk.gdk.WINDOW_EDGE_SOUTH_WEST`	The lower left corner.
`gtk.gdk.WINDOW_EDGE_SOUTH`	The lower edge.
`gtk.gdk.WINDOW_EDGE_SOUTH_EAST`	The lower right corner.

GDK Window Hints Constants

The Window Hints constants specify the fields of a GdkGeometry struct should be paid attention to. Also, the presence/absence of `gtk.gdk.HINT_POS`, `gtk.gdk.HINT_USER_POS`, and `gtk.gdk.HINT_USER_SIZE` is significant, though they don't directly refer to GdkGeometry fields. `gtk.gdk.HINT_USER_POS` will be set automatically by `gtk.Window` if you call the `gtk.Window.move()` method. `gtk.gdk.HINT_USER_POS` and `gtk.gdk.HINT_USER_SIZE` should be set if the user specified a size/position using a −−geometry command−line argument; the `gtk.Window.parse_geometry()` method automatically sets these flags. These constants aren't useful in PyGTK.

`gtk.gdk.HINT_POS`	Indicates that the program has positioned the window
`gtk.gdk.HINT_MIN_SIZE`	Min size fields are set
`gtk.gdk.HINT_MAX_SIZE`	Max size fields are set
`gtk.gdk.HINT_BASE_SIZE`	Base size fields are set
`gtk.gdk.HINT_ASPECT`	Aspect ratio fields are set
`gtk.gdk.HINT_RESIZE_INC`	Resize increment fields are set
`gtk.gdk.HINT_WIN_GRAVITY`	Window gravity field is set
`gtk.gdk.HINT_USER_POS`	Indicates that the window's position was explicitly set by the user
`gtk.gdk.HINT_USER_SIZE`	Indicates that the window's size was explicitly set by the user

GDK Window State Flag Constants

The Window State flag constants are a set of bit−flags that specify the state of a toplevel window.

`gtk.gdk.WINDOW_STATE_WITHDRAWN`	The window is not shown.
`gtk.gdk.WINDOW_STATE_ICONIFIED`	The window is minimized.
`gtk.gdk.WINDOW_STATE_MAXIMIZED`	The window is maximized.
`gtk.gdk.WINDOW_STATE_STICKY`	The window is sticky.
`gtk.gdk.WINDOW_STATE_FULLSCREEN`	The window is maximized without decorations. Available in PyGTK 2.2 and above.
`gtk.gdk.WINDOW_STATE_ABOVE`	The window is kept above other windows. Available in PyGTK 2.4 and above.
`gtk.gdk.WINDOW_STATE_BELOW`	The window is kept below other windows. Available in PyGTK 2.4 and above.

GDK Window Type Constants

The Window Type constants specify the type of window.

`gtk.gdk.WINDOW_ROOT`	The root window; this window has no parent, covers the entire screen, and is created by the window system.
`gtk.gdk.WINDOW_TOPLEVEL`	A toplevel window (used to implement `gtk.Window`).
`gtk.gdk.WINDOW_CHILD`	A child window (used to implement widgets e.g. `gtk.Entry`).
`gtk.gdk.WINDOW_DIALOG`	A useless/deprecated compatibility type.
`gtk.gdk.WINDOW_TEMP`	An override redirect temporary window (used to implement `gtk.Menu`).
`gtk.gdk.WINDOW_FOREIGN`	A foreign window (see the `gtk.gdk.window_foreign_new()` function).

GDK Window Type Hint Constants

The Window Type Hint constants specify hints for the window manager that indicate what type of function the window has. The window manager can use this when determining decoration and behavior of the window. The hint must be set before mapping the window.

`gtk.gdk.WINDOW_TYPE_HINT_NORMAL`	A normal toplevel window.
`gtk.gdk.WINDOW_TYPE_HINT_DIALOG`	A dialog window.
`gtk.gdk.WINDOW_TYPE_HINT_MENU`	A window used to implement a menu.
`gtk.gdk.WINDOW_TYPE_HINT_TOOLBAR`	A window used to implement a toolbar.
`gtk.gdk.WINDOW_TYPE_HINT_SPLASHSCREEN`	A window used to implement a splash screen
`gtk.gdk.WINDOW_TYPE_HINT_UTILITY`	
`gtk.gdk.WINDOW_TYPE_HINT_DOCK`	A window used to implement a docking bar.
`gtk.gdk.WINDOW_TYPE_HINT_DESKTOP`	A window used to implement a desktop.

GDK WM Decoration Constants

The WM Decoration constants are bit−flags that specify the hints that the window manager can use when determining how to decorate the window. The hints must be set before mapping the window.

`gtk.gdk.DECOR_ALL`	All decorations should be applied.
`gtk.gdk.DECOR_BORDER`	A frame should be drawn around the window.
`gtk.gdk.DECOR_RESIZEH`	The frame should have resize handles.
`gtk.gdk.DECOR_TITLE`	A titlebar should be placed above the window.
`gtk.gdk.DECOR_MENU`	A button for opening a menu should be included.
`gtk.gdk.DECOR_MINIMIZE`	A minimize button should be included.
`gtk.gdk.DECOR_MAXIMIZE`	A maximize button should be included.

GDK WM Function Constants

The WM Function constants specify hints originally defined by the Motif toolkit. The window manager can use them when determining the functions to offer for the window. The hint must be set before mapping the window.

`gtk.gdk.FUNC_ALL`	All functions should be offered.
`gtk.gdk.FUNC_RESIZE`	The window should be resizable.
`gtk.gdk.FUNC_MOVE`	The window should be movable.
`gtk.gdk.FUNC_MINIMIZE`	The window should be minimizable.
`gtk.gdk.FUNC_MAXIMIZE`	The window should be maximizable.

`gtk.gdk.FUNC_CLOSE`	The window should be closeable.

gtk.gdk Functions

gtk.gdk Functions — the gtk.gdk module functions

Synopsis

gtk.gdk.Atom Functions

```
    def gtk.gdk.atom_intern(name, only_if_exists=FALSE)
```

gtk.gdk.Color Functions

```
    def gtk.gdk.color_parse(spec)
```

gtk.gdk.Colormap Functions

```
    def gtk.gdk.colormap_get_system()
```

gtk.gdk.Device Functions

```
    def gtk.gdk.devices_list()
     def gtk.gdk.device_get_core_pointer()
```

gtk.gdk.Display Functions

```
    def gtk.gdk.display_get_default()
```

gtk.gdk.DisplayManager Functions

```
    def gtk.gdk.display_manager_get()
```

gtk.gdk.Event Functions

```
    def gtk.gdk.events_pending()
     def gtk.gdk.event_peek()
     def gtk.gdk.event_get()
     def gtk.gdk.event_get_graphics_expose()
     def gtk.gdk.set_show_events()
     def gtk.gdk.get_show_events()
```

gtk.gdk.Keymap Functions

```
   def gtk.gdk.keymap_get_default()
    def gtk.gdk.keymap_get_for_display(display)
    def gtk.gdk.keyval_name(keyval)
    def gtk.gdk.keyval_from_name(keyval_name)
    def gtk.gdk.keyval_convert_case(symbol)
    def gtk.gdk.keyval_to_upper(keyval)
    def gtk.gdk.keyval_to_lower(keyval)
    def gtk.gdk.keyval_is_upper(keyval)
    def gtk.gdk.keyval_is_lower(keyval)
    def gtk.gdk.keyval_to_unicode(keyval)
    def gtk.gdk.unicode_to_keyval(wc)
```

gtk.gdk.Pixbuf Functions

```
   def gtk.gdk.pixbuf_new_from_file(filename)
    def gtk.gdk.pixbuf_new_from_file_at_size(filename, width, height)
    def gtk.gdk.pixbuf_new_from_data(data, colorspace, has_alpha, bits_per_sample, width, heigh
    def gtk.gdk.pixbuf_new_from_array(array, colorspace, bits_per_sample)
    def gtk.gdk.pixbuf_new_from_xpm_data(data)
    def gtk.gdk.pixbuf_new_from_inline(data_length, data, copy_pixels)
    def gtk.gdk.pixbuf_get_formats()
    def gtk.gdk.pixbuf_get_file_info(filename)
```

gtk.gdk.PixbufLoader Functions

```
   def gtk.gdk.pixbuf_loader_new_with_mime_type(mime_type)
```

gtk.gdk.Pixmap Functions

```
   def gtk.gdk.bitmap_create_from_data(drawable, data, width, height)
    def gtk.gdk.pixmap_create_from_data(drawable, data, width, height, depth, fg, bg)
    def gtk.gdk.pixmap_create_from_xpm(window, transparent_color, filename)
    def gtk.gdk.pixmap_colormap_create_from_xpm(window, colormap, transparent_color, filename)
    def gtk.gdk.pixmap_create_from_xpm_d(window, transparent_color, data)
    def gtk.gdk.pixmap_colormap_create_from_xpm_d(window, colormap, transparent_color, data)
    def gtk.gdk.pixmap_foreign_new(anid)
    def gtk.gdk.pixmap_lookup(anid)
    def gtk.gdk.pixmap_foreign_new_for_display(display, anid)
    def gtk.gdk.pixmap_lookup_for_display(display, anid)
```

gtk.gdk.Screen Functions

```
   def gtk.gdk.screen_width()
    def gtk.gdk.screen_height()
    def gtk.gdk.screen_width_mm()
    def gtk.gdk.screen_height_mm()
    def gtk.gdk.screen_get_default()
```

gtk.gdk.Visual Functions

```
   def gtk.gdk.list_visuals()
    def gtk.gdk.visual_get_best()
    def gtk.gdk.visual_get_best_depth()
    def gtk.gdk.visual_get_best_type()
    def gtk.gdk.visual_get_best_with_depth(depth)
    def gtk.gdk.visual_get_best_with_type(type)
    def gtk.gdk.visual_get_system()
    def gtk.gdk.query_depths()
    def gtk.gdk.query_visual_types()
```

gtk.gdk.Window Functions

```
   def gtk.gdk.window_foreign_new(anid)
    def gtk.gdk.window_foreign_new_for_display(display, anid)
    def gtk.gdk.get_default_root_window()
    def gtk.gdk.window_get_toplevels()
    def gtk.gdk.window_lookup(anid)
    def gtk.gdkwindow_lookup_for_display.(display, anid)
    def gtk.gdk.window_process_all_updates()
    def gtk.gdk.gdk_window_set_debug_updates()
    def gtk.gdk.window_at_pointer()
```

Miscellaneous Functions

```
   def gtk.gdk.pointer_grab(window, owner_events=FALSE, event_mask=0, confine_to=None, cursor=N
    def gtk.gdk.pointer_ungrab(time=0L)
    def gtk.gdk.keyboard_grab(window, owner_events=FALSE, time=0L)
    def gtk.gdk.keyboard_ungrab(time=0L)
    def gtk.gdk.pointer_is_grabbed()
    def gtk.gdk.flush()
    def gtk.gdk.beep()
    def gtk.gdk.set_double_click_time(msec)
    def gtk.gdk.threads_enter()
    def gtk.gdk.threads_leave()
    def gtk.gdk.threads_init()
    def gtk.gdk.rgb_ditherable()
    def gtk.gdk.rgb_get_colormap()
    def gtk.gdk.rgb_set_verbose(verbose)
    def gtk.gdk.rgb_set_install(install)
    def gtk.gdk.rgb_set_min_colors(min_colors)
    def gtk.gdk.rgb_get_visual()
    def gtk.gdk.selection_owner_get(selection)
    def gtk.gdk.selection_send_notify(requestor, selection, target, property, time)
    def gtk.gdk.set_sm_client_id(sm_client_id)
```

Description

These functions are part of the `PyGTK` gtk.gdk module. All the functions are listed above with links to the function description. Most functions are associated with a specific object class and their descriptions are part of the class reference. Those functions that are not directly associated with a specific class have their descriptions below.

Functions

gtk.gdk.pointer_grab

```
   def gtk.gdk.pointer_grab(window, owner_events=FALSE, event_mask=0, confine_to=None, cursor=N
```

window :	the `gtk.gdk.Window` that will own the grab (the grab window).
owner_events :	if `FALSE` then all pointer events are reported with respect to *window* and are only reported if selected by *event_mask*. If `TRUE` then pointer events for this application are reported as normal, but pointer events outside this application are reported with respect to *window* and only if selected by *event_mask*. In

	either mode, unreported events are discarded.
event_mask :	specifies the event mask, that is used in accordance with *owner_events*.
confine_to :	If not `None`, the pointer will be confined to this `gtk.gdk.Window` during the grab. If the pointer is outside *confine_to*, it will automatically be moved to the closest edge of *confine_to* and enter and leave events will be generated as necessary.
cursor :	the `gtk.gdk.Cursor` to display while the grab is active. If this is None then the normal cursors are used for *window* and its descendants, and the cursor for *window* is used for all other windows.
time :	the timestamp of the event that led to this pointer grab. This usually comes from a `gtk.gdk.Event`, though)l can be used to use the current time if the time isn't known.
Returns :	a grab status value

The `gtk.gdk.pointer_grab()` function grabs the pointer (usually a mouse) so that all events are passed to this application until the pointer is ungrabbed with the `gtk.gdk.pointer_ungrab()`, or the grab window becomes unviewable. This overrides any previous pointer grab by this client. This function returns a grab status value:

`gtk.gdk.GRAB_SUCCESS`	The resource was successfully grabbed.
`gtk.gdk.GRAB_ALREADY_GRABBED`	The resource is actively grabbed by another client.
`gtk.gdk.GRAB_INVALID_TIME`	The resource was grabbed more recently than the specified time.
`gtk.gdk.GRAB_NOT_VIEWABLE`	The grab window or the *confine_to* window are not viewable.
`gtk.gdk.GRAB_FROZEN`	The resource is frozen by an active grab of another client.

Pointer grabs are used for operations that need complete control over mouse events, even if the mouse leaves the application. For example it is used for drag and drop, for dragging the handle in the `gtk.HPaned` and `gtk.VPaned` widgets. Note that if the event mask of an X window has selected both button press and button release events, then a button press event will cause an automatic pointer grab until the button is released. X does this automatically since most applications expect to receive button press and release events in pairs. It is equivalent to a pointer grab on the window with *owner_events* set to `TRUE`.

gtk.gdk.pointer_ungrab

```
    def gtk.gdk.pointer_ungrab(time=0L)
```

time :	a timestamp from a `gtk.gdk.Event` or 0L to use the current time

The `gtk.gdk.pointer_ungrab()` function ungrabs the pointer if it is grabbed by this application.

gtk.gdk.keyboard_grab

```
    def gtk.gdk.keyboard_grab(window, owner_events=FALSE, time=0L)
```

window :	the `gtk.gdk.Window` that will own the grab (the grab window).
owner_events :	if `FALSE` then all keyboard events are reported with respect to *window*. If `TRUE` then keyboard events for this application are reported as normal, but keyboard events outside this application are reported with respect to *window*. Both key press and key release events are always reported, independent of the event mask set by the application.
time :	a timestamp from a `gtk.gdk.Event` or 0L to use the current time
Returns :	a grab status value

The `gtk.gdk.keyboard_grab()` function grabs the keyboard so that all events are passed to this application until the keyboard is ungrabbed with the `gtk.gdk.keyboard_ungrab()`) function. This overrides any previous keyboard grab by this client.

gtk.gdk.keyboard_ungrab

```
    def gtk.gdk.keyboard_ungrab(time=0L)
```

`time` :	a timestamp from a `gtk.gdk.Event` or 0L to use the current time

The `gtk.gdk.keyboard_ungrab()` function ungrabs the keyboard if it is grabbed by this application.

gtk.gdk.pointer_is_grabbed

```
    def gtk.gdk.pointer_is_grabbed()
```

Returns :	`TRUE` if the pointer is currently grabbed by this application.

The `gtk.gdk.pointer_is_grabbed()` function returns `TRUE` if the pointer is currently grabbed by this application. Note that this does not take the implicit pointer grab on button presses into account.

gtk.gdk.flush

```
    def gtk.gdk.flush()
```

The `gtk.gdk.flush()` function flushes the X output buffer and waits until all requests have been processed by the server. This is rarely needed by applications.

gtk.gdk.beep

```
    def gtk.gdk.beep()
```

The `gtk.gdk.beep()` function emits a short beep.

gtk.gdk.set_double_click_time

```
    def gtk.gdk.set_double_click_time(msec)
```

`msec` :	the double click time in milliseconds

The `gtk.gdk.set_double_click_time()` function set the double click time for the default display. Applications should NOT set this, it is a global user–configured setting.

gtk.gdk.threads_enter

```
    def gtk.gdk.threads_enter()
```

The `gtk.gdk.threads_enter()` function marks the beginning of a critical section that only one thread can operate within at a time. The critical section is guarded by a `GDK` mutual exclusion lock. Python threads are enabled while waiting for the `GDK` lock. See the `gtk.gtk.threads_init()` function for more information about threading.

gtk.gdk.threads_leave

```
    def gtk.gdk.threads_leave()
```

The `gtk.gdk.threads_leave()` function marks the end of a critical section started by the `gtk.gdk.threads_enter()` function. See the `gtk.gtk.threads_init()` function for more information about threading.

gtk.gdk.threads_init

```
    def gtk.gdk.threads_init()
```

The `gtk.gdk.threads_init()` function initializes `PyGTK` to use the Python macros that allow multiple threads to serialize access to the Python interpreter (using the Python Global Interpreter Lock (GIL)). In addition, the `gtk.gdk.threads_init()` function initializes the `GDK` global lock (mutex) that serializes thread access to the `GTK` and `GDK` libraries. Thus there are two different global locking mechanisms at work that are initialized by the `gtk.gdk.threads_init()` function: Python and `GDK`.

The `gtk.gdk.threads_init()` function must be called before the `gtk.main()` function. At this point in the application the Python GIL is held by the main application thread. (Usually the main thread calls the `gtk.gdk.threads_init()` function though any thread could call it instead.) When the `gtk.main()` function is called the GIL is released and other threads may use the Python interpreter. When `PyGTK` signal handlers are invoked in the main thread the GIL is reacquired so that the Python interpreter can run the handler code.

The assumptions behind `PyGTK` thread support were originally:

- A GIL count is initialized for each thread to indicate that it holds the GIL − the assumption being that the thread will be a Python thread and will hold the GIL when it starts because it is running in the Python interpreter. `PyGTK` adds the GIL count for each thread to provide a recursive lock mechanism. That is, a Python thread may attempt to gain exclusive access to the Python interpreter multiple times without fear of deadlock.
- The Python thread that calls the `gtk.main()` function releases the GIL allowing other Python threads to run.
- `PyGTK` does not release the GIL when calling a `GTK` or `GDK` function. Also it does not acquire the `GDK` global lock (GGL). This means that, in effect, Python threads can use the GIL alone to serialize access to the `GTK` and `GDK` libraries. Of course, if there are non−Python threads calling `GTK` or `GDK` functions the GGL must be used.
- Signal, timeout and idle handlers written in Python that are invoked from the `GTK` mainloop acquire the GIL automatically.

The `gtk.gdk.threads_enter()` and `gtk.gdk.threads_leave()` functions use the `GDK` global lock (GGL) to manage access to the `GTK` and `GDK` libraries. The intention is to allow only one thread to execute within the `GTK` and `GDK` code at a time. Theoretically any time a thread calls a `PyGTK` method or function it should bracket the call with the `gtk.gdk.threads_enter()` and `gtk.gdk.threads_leave()` functions. If your application only uses Python threads then this is not necessary since only the main thread can safely call `PyGTK` methods or functions. However if your application has foreign threads that call `GTK` or `GDK` functions you should use the `gtk.gdk.threads_enter()` and `gtk.gdk.threads_leave()` functions to serialize access.

Note

Signal handlers are automatically invoked within a `gdk_threads_enter()` and `gdk_threads_leave()` function pair by `GTK` so the `gtk.gdk.threads_enter()` and `gtk.gdk.threads_leave()` functions should not be called within a Python signal handler or the application will deadlock. However, idle, timeout and input handlers are executed outside the GGL so these should use the gtk.gdk.threads_enter() and gtk.gdk.threads_leave() functions if PyGTK methods or functions are called.

In general the safest strategy is to only call PyGTK methods and functions from the main Python thread and use idle or timeout handlers (which run in the main thread) to invoke PyGTK calls from other threads.

gtk.gdk.rgb_ditherable

```
def gtk.gdk.rgb_ditherable()
```

Returns :	TRUE if the gtk.gdk.Visual is ditherable

The gtk.gdk.rgb_ditherable() function returns TRUE if the gtk.gdk.Visual is ditherable. This function may be useful for presenting a user interface choice to the user about which dither mode is desired; if the display is not ditherable, it may make sense to gray out or hide the corresponding UI widget.

gtk.gdk.rgb_get_colormap

```
def gtk.gdk.rgb_get_colormap()
```

Returns :	the preferred gtk.gdk.Colormap for rendering image data.

The gtk.gdk.rgb_get_colormap() function returns the preferred gtk.gdk.Colormap.

gtk.gdk.rgb_set_verbose

```
def gtk.gdk.rgb_set_verbose(verbose)
```

verbose :	If TRUE messages should be verbose

The gtk.gdk.rgb_set_verbose() function sets the "verbose" flag to the value specified by *verbose*. If *verbose* is TRUE messages will be verbose. This is generally only useful for debugging.

gtk.gdk.rgb_set_install

```
def gtk.gdk.rgb_set_install(install)
```

install :	if TRUE set install mode

The gtk.gdk.rgb_set_install() function sets the "install" mode to the value of *install*. If *install* is TRUE, a new "private" colormap is always installed rather than trying to find a best fit with the colors already allocated. Ordinarily, a colormap only be installed if a sufficient cube cannot be allocated. A private colormap has more colors, leading to better quality display, but also leads to the dreaded "colormap flashing" effect.

gtk.gdk.rgb_set_min_colors

```
def gtk.gdk.rgb_set_min_colors(min_colors)
```

min_colors :	the minimum number of colors.

The gtk.gdk.rgb_set_min_colors() function sets the minimum number of colors for the color cube to the value specified by *min_colors*. Generally, the largest color cube is allocated. If a color cube at least as large as *min_colors* can't be allocated, a private colormap is installed.

gtk.gdk.rgb_get_visual

```
def gtk.gdk.rgb_get_visual()
```

Returns :	the `gtk.gdk.Visual` being used

The `gtk.gdk.rgb_get_visual`() function returns the `gtk.gdk.Visual` being used to render image data on the default screen.

gtk.gdk.selection_owner_get

```
def gtk.gdk.selection_owner_get(selection)
```

`selection` :	an atom indentifying a selection.
Returns :	the `gtk.gdk.Window` that owns the selection or None.

The `gtk.gdk.selection_owner_get`() function returns the `gtk.gdk.Window` that owns the selection specified by *`selection`* if there is a selection owner for this window, and if it is a window known to the current application. Note that the return value may be owned by a different process if a foreign window was previously created for that window, but a new foreign window will never be created by this call.

gtk.gdk.selection_send_notify

```
def gtk.gdk.selection_send_notify(requestor, selection, target, property, time)
```

`requestor` :	the integer ID of the window to deliver the response to.
`selection` :	an atom representing the selection that was requested.
`target` :	an atom representing the target that was selected.
`property` :	an atom representing the property in which the selection owner stored the data.
`time` :	a timestamp

The `gtk.gdk.selection_send_notify`() function sends a response to a SelectionRequest event.

gtk.gdk.set_sm_client_id

```
def gtk.gdk.set_sm_client_id(sm_client_id)
```

`sm_client_id` :	the client id string assigned by the session manager when the connection was opened

The `gtk.gdk.set_sm_client_id`() function sets the `SM_CLIENT_ID` property on the application's leader window so that the window manager can save the application's state using the X11R6 ICCCM session management protocol. See the X Session Management Library documentation for more information on session management and the Inter−Client Communication Conventions Manual (ICCCM) for information on the `WM_CLIENT_LEADER` property. (Both documents are part of the X Window System distribution.)

Prev	Up	Next
gtk.gdk.Window	Home	gtk.gdk Constants

The gtk.glade Class Reference

Prev Next

The gtk.glade Class Reference

Table of Contents

The gobject Class Reference

Table of Contents

gobject Constants

gobject Constants the built–in constants of the gobject module

Synopsis

```
GObject IO Condition Constants
GObject Param Flag Constants
GObject Priority Constants
GObject Signal Flag Constants
GObject Built-in Type Constants
GObject Version Constants
```

Description

GObject IO Condition Constants

The IO Condition constants are a set of bit–flags that specify a condition to watch for on an event source.

`gobject.IO_IN`	There is data to read
`gobject.IO_OUT`	Data can be written (without blocking).
`gobject.IO_PRI`	There is urgent data to read.
`gobject.IO_ERR`	Error condition.
`gobject.IO_HUP`	Hung up (the connection has been broken, usually for pipes and sockets).

`gobject.IO_NVAL`	Invalid request. The file descriptor is not open.

GObject Param Flag Constants

The Param Flag constants are a set of bit–flags that specify certain aspects of parameters that can be configured.

`gobject.PARAM_READABLE`	The parameter is readable
`gobject.PARAM_WRITABLE`	The parameter is writable
`gobject.PARAM_CONSTRUCT`	The parameter will be set upon object construction
`gobject.PARAM_CONSTRUCT_ONLY`	The parameter will only be set upon object construction
`gobject.PARAM_LAX_VALIDATION`	Upon parameter conversion strict validation is not required

GObject Priority Constants

The Priority constants specify

`gobject.PRIORITY_HIGH`	Use this for high priority event sources.
`gobject.PRIORITY_DEFAULT`	Use this for default priority event sources. This priority is used when adding timeout functions with the `gobject.timeout_add()` function. This priority is also used for events from the X server.
`gobject.PRIORITY_HIGH_IDLE`	Use this for high priority idle functions. For example, gobject.PRIORITY_HIGH_IDLE + 10 is used for resizing operations; and, gobject.PRIORITY_HIGH_IDLE + 20, for redrawing operations. (This is done to ensure that any pending resizes are processed before any pending redraws, so that widgets are not redrawn twice unnecessarily.)
`gobject.PRIORITY_DEFAULT_IDLE`	Use this for default priority idle functions. This priority is used when adding idle functions with the `gobject.idle_add()` function.
`gobject.PRIORITY_LOW`	Use this for very low priority background tasks.

GObject Signal Flag Constants

The Signal Flag constants are a set of bit–flags that specify a signal's behavior. The overall signal description outlines how especially the `RUN` flags control the stages of a signal emission.

`gobject.SIGNAL_RUN_FIRST`	Invoke the object method handler in the first emission stage.
`gobject.SIGNAL_RUN_LAST`	Invoke the object method handler in the third emission stage.
`gobject.SIGNAL_RUN_CLEANUP`	Invoke the object method handler in the last emission stage.
`gobject.SIGNAL_NO_RECURSE`	Signals being emitted for an object while currently being in emission for this very object will not be emitted recursively, but instead cause the first emission to be restarted.
`gobject.SIGNAL_DETAILED`	This signal supports "::detail" appendices to the signal name upon handler connections and emissions.
`gobject.SIGNAL_ACTION`	Action signals are signals that may freely be emitted on alive objects from user code via the `gobject.emit()` method and friends, without the need of being embedded into extra code that performs pre or post emission adjustments on the

	object. They can also be thought of as object methods which can be called generically by third−party code.
`gobject.SIGNAL_NO_HOOKS`	No emissions hooks are supported for this signal.

GObject Built−in Type Constants

The Built−in Type constants specify the pre−defined types used by gobject.

`gobject.TYPE_INVALID`	An invalid type, used as error return value in some functions.
`gobject.TYPE_NONE`	A fundamental type indicating no type.
`gobject.TYPE_INTERFACE`	The fundamental type from which all interfaces are derived.
`gobject.TYPE_CHAR`	The fundamental type corresponding to a character. This maps to a string in Python.
`gobject.TYPE_UCHAR`	The fundamental type corresponding to an unsigned character. This maps to a string in Python.
`gobject.TYPE_BOOLEAN`	The fundamental type corresponding to a True or False value. This maps to an integer in Python.
`gobject.TYPE_INT`	The fundamental type corresponding to an integer. This maps to an integer in Python.
`gobject.TYPE_UINT`	he fundamental type corresponding to an unsigned integer. This maps to an integer in Python.
`gobject.TYPE_LONG`	The fundamental type corresponding to a long integer. This maps to an integer in Python.
`gobject.TYPE_ULONG`	The fundamental type corresponding to an unsigned integer. This maps to an integer in Python.
`gobject.TYPE_INT64`	The fundamental type corresponding to an long long integer. This maps to a long integer in Python.
`gobject.TYPE_UINT64`	The fundamental type corresponding to an unsigned long long integer. This maps to a long integer in Python.
`gobject.TYPE_ENUM`	The fundamental type corresponding to an enumeration type. This maps to an integer in Python.
`gobject.TYPE_FLAGS`	The fundamental type corresponding to a flag type. This maps to an integer in Python.
`gobject.TYPE_FLOAT`	The fundamental type corresponding to a floating point number. This maps to a float in Python.
`gobject.TYPE_DOUBLE`	The fundamental type corresponding to a double floating point number. This maps to a float in Python.
`gobject.TYPE_STRING`	The fundamental type corresponding to a string.
`gobject.TYPE_POINTER`	The fundamental type corresponding to a pointer to an anonymous type. This has no corresponding Python type.
`gobject.TYPE_BOXED`	The fundamental type corresponding to a boxed object type.
`gobject.TYPE_PARAM`	The fundamental type corresponding to a GParamSpec type.
`gobject.TYPE_OBJECT`	The fundamental type corresponding to a GObject type.

GObject Version Constants

The Version constants specify the version of `GLIB` used by PyGTK as a 3−tuple containing the major, minor and patch release numbers.

`gobject.glib_version`	A 3−tuple containing (major, minor, patch) release numbers.

gobject Functions

gobject Functions miscellaneous functions

Synopsis

```
    def gobject.type_name(type)
    def gobject.type_from_name(type_name)
    def gobject.type_parent(type)
    def gobject.type_is_a(type, parent_type)
    def gobject.type_children(type)
    def gobject.type_interfaces(type)
    def gobject.type_register(class)
    def gobject.signal_new(signal_name, type, flags, return_type, param_types)
    def gobject.signal_list_names(type)
    def gobject.signal_list_ids(type)
    def gobject.signal_lookup(name, type)
    def gobject.signal_name(signal_id)
    def gobject.signal_query(name, type)
    def gobject.signal_query(signal_id)
    def gobject.list_properties(type)
    def gobject.new(type, ...)
    def gobject.idle_add(callback, ...)
    def gobject.timeout_add(interval, callback, ...)
    def gobject.io_add_watch(fd, condition, callback, ...)
    def gobject.source_remove(tag)
    def gobject.main_context_default()
```

Description

These functions are part of the `PyGTK` gobject module but are not directly associated with a specific class.

Functions

gobject.type_name

```
  def gobject.type_name(type)
```

`type` :	a GObject type, type ID or instance
Returns :	

The `gobject.type_name()` function returns the unique name that is assigned to the specified *`type`*. *`type`* can be a GObject type, type ID or instance. This function raises a TypeError exception if *`type`* isn't a `PyGTK` type.

gobject.type_from_name

```
def gobject.type_from_name(type_name)
```

type_name :	a string containing the name of a type
Returns :	the type ID named *type_name*

The `gobject.type_from_name()` function returns the type ID of the `PyGTK` type with the name specified by *type_name*. This function raises a RuntimeError exception if no type matches *type_name*.

gobject.type_parent

```
def gobject.type_parent(type)
```

type :	a GObject type, type ID or instance
Returns :	the parent type ID

The `gobject.type_parent()` function returns the direct parent type ID of the specified *type*. *type* can be a GObject type, type ID or instance. If *type* has no parent, i.e. is a fundamental type, the RuntimeError exception is raised.

gobject.type_is_a

```
def gobject.type_is_a(type, parent_type)
```

type :	a GObject type, type ID or instance
parent_type :	a GObject type, type ID or instance
Returns :	`TRUE` if *parent_type* is an ancestor of *type*

The `gobject.type_is_a()` function returns `TRUE` if the specified *type* is a descendant of the type specified by *parent_type*. This function also returns `TRUE` if *parent_type* is an interface and *type* conforms to it.

gobject.type_children

```
def gobject.type_children(type)
```

type :	a GObject type, type ID or instance
Returns :	a list of the child types of *type*

The `gobject.type_children()` function returns a list containing the child types of the specified *type*.

gobject.type_interfaces

```
def gobject.type_interfaces(type)
```

type :	a GObject type, type ID or instance
Returns :	a list of the interface types supported by *type*

The `gobject.type_interfaces()` function returns a list of the interface types supported by *type*. *type* can be a GObject type, type ID or instance. This function returns a RuntimeError exception if type is not a valid type or has no interfaces.

gobject.type_register

```
def gobject.type_register(class)
```

class :	a Python class that is a descendant of `gobject.GObject`

The `gobject.type_register()` function registers the specified Python *class* as a PyGTK type. class must be a descendant of `gobject.GObject`. The function generates a name for the new type.

gobject.signal_new

```
def gobject.signal_new(signal_name, type, flags, return_type, param_types)
```

signal_name :	a string containing the name of the signal
type :	the object type that the signal is associated with
flags :	the signal flags
return_type :	the return type of the signal handler
param_types :	the parameter types passed to the signal handler
Returns :	a unique integer signal ID

The `gobject.signal_new()` function registers a signal with the specified *signal_name* for the specified object *type*. The value of *flags* is a combination of:

`gobject.SIGNAL_RUN_FIRST`	Invoke the object method handler in the first emission stage.
`gobject.SIGNAL_RUN_LAST`	Invoke the object method handler in the third emission stage.
`gobject.SIGNAL_RUN_CLEANUP`	Invoke the object method handler in the last emission stage.
`gobject.SIGNAL_NO_RECURSE`	Signals being emitted for an object while currently being in emission for this very object will not be emitted recursively, but instead cause the first emission to be restarted.
`gobject.SIGNAL_DETAILED`	This signal supports "::detail" appendixes to the signal name upon handler connections and emissions.
`gobject.SIGNAL_ACTION`	Action signals are signals that may freely be emitted on alive objects from user code via `gobject.emit()()` and friends, without the need of being embedded into extra code that performs pre or post emission adjustments on the object. They can also be thought of as generically callable object methods.
`gobject.SIGNAL_NO_HOOKS`	No emissions hooks are supported for this signal.

return_type is the type of the return value from a signal handler and may be a gobject type, type ID or instance. The *param_types* parameter is a list of additional types that are passed to the signal handler. Each parameter type may be specified as a gobject type, type ID or instance. For example, to add a signal to the gtk.Window type called "my−signal" that calls a handler with a gtk.Button widget and an integer value and a return value that is a boolean, use:

```
  gobject.signal_new("my_signal", gtk.Window, gobject.SIGNAL_RUN_LAST, gobject.TYPE_BOOLEAN, (g
```

gobject.signal_list_names

```
def gobject.signal_list_names(type)
```

type :	a GObject type, type ID or instance
Returns :	a list of the signal names supported by *type*

The `gobject.signal_list_names()` function returns a list of the names of the signals that are supported by the specified GObject *type*

Note

The type keyword is available in PyGTK 2.6 and above.

gobject.signal_list_ids

```
def gobject.signal_list_ids(type)
```

`type` :	a GObject type, type ID or instance
Returns :	a list of the signal ids supported by *`type`*

Note

This method is available in PyGTK 2.6 and above.

The `gobject.signal_list_ids()` function returns a list of the integer ids of the signals that are supported by the GObject specified by *`type`*

gobject.signal_lookup

```
def gobject.signal_lookup(name, type)
```

`name` :	the name of a signal for *`type`*
`type` :	a GObject type, type ID or instance
Returns :	the integer id of a signal supported by *`type`* or 0.

Note

This method is available in PyGTK 2.6 and above.

The `gobject.signal_lookup()` function returns the id of the signal with the name specified by *`name`* that is supported by the GObject specified specified by*`type`*. 0 is returned if the signal is not found.

gobject.signal_name

```
def gobject.signal_name(signal_id)
```

`signal_id` :	an integer signal id
Returns :	the name of the signal or `None`.

Note

This method is available in PyGTK 2.6 and above.

The `gobject.signal_name()` function returns the name of the signal that has the signal id specified by *`id`*.

gobject.signal_query

```
def gobject.signal_query(name, type)
```

`name` :	the name of a signal for *`type`*
`type` :	a GObject type, type ID or instance
Returns :	a 6–tuple containing signal information or `None`

Note

This method is available in PyGTK 2.6 and above.

The `gobject.signal_query()` function returns a 6–tuple containing information about the signal with the name specified by *`name`* that is supported by the GObject specified by *`type`*. If the signal is not found `None` is returned.

The signal information 6–tuple contains:

- the integer signal id
- the signal name
- the GType that the signal is registered for
- the signal flags (see the GObject Signal Flag Constants)
- the GType of the return from the signal callback function
- a tuple containing the GTypes of the parameters that are passed to the signal callback function. Note that these may not correspond exactly to the `PyGTK` signal callback parameters.

gobject.signal_query

```
    def gobject.signal_query(signal_id)
```

`signal_id` :	the integer id of a signal
Returns :	a 6–tuple containing signal information or `None`

Note

This method is available in PyGTK 2.6 and above.

The `gobject.signal_query()` function returns a 6–tuple containing information about the signal with the id specified by *`signal_id`*. If the signal is not found `None` is returned.

The signal information 6–tuple contains:

- the integer signal id
- the signal name
- the GType that the signal is registered for
- the signal flags (see the GObject Signal Flag Constants)
- the GType of the return from the signal callback function
- a tuple containing the GTypes of the parameters that are passed to the signal callback function. Note that these may not correspond exactly to the `PyGTK` signal callback parameters.

gobject.list_properties

```
    def gobject.list_properties(type)
```

`type` :	a GObject type, type ID or instance
Returns :	a list of the properties (as GParam objects) supported by *`type`*

The `gobject.list_properties()` function returns a list of the properties (as GParam objects) supported by *`type`*.

gobject.new

```
def gobject.new(type, ...)
```

type :	a GObject type, type ID or instance
... :	zero or more property−value pairs
Returns :	a new object if the specified *type*

The `gobject.new()` function returns a new object of the specified *type*. type must specify a type that is a descendant of `gobject.GObject`. A TypeError exception is raised if *type* specifies an abstract class or a type that is not a descendant of `gobject.GObject`. A set of property−value pairs may be specified to set the value of the object's properties.

gobject.idle_add

```
def gobject.idle_add(callback, ...)
```

callback :	a function to call when `PyGTK` is idle
... :	optionals arguments to be passed to *callback*
Returns :	an integer ID

The `gobject.idle_add()` function adds a function (specified by *callback*) to be called whenever there are no higher priority events pending to the default main loop. The function is given the default idle priority, `gobject.PRIORITY_DEFAULT_IDLE`. Additional arguments to pass to *callback* can be specified after *callback*. The idle priority can be specified as a keyword−value pair with the keyword "priority". If *callback* returns `FALSE` it is automatically removed from the list of event sources and will not be called again.

gobject.timeout_add

```
def gobject.timeout_add(interval, callback, ...)
```

interval :	the time between calls to the function, in milliseconds
callback :	the function to call
... :	zero or more arguments that will be passed to *callback*
Returns :	an integer ID of the event source

The `gobject.timeout_add()` function sets a function (specified by *callback*) to be called at regular intervals (specified by *interval*, with the default priority, `gobject.PRIORITY_DEFAULT`. Additional arguments to pass to *callback* can be specified after *callback*. The idle priority may be specified as a keyword−value pair with the keyword "priority".

The function is called repeatedly until it returns `FALSE`, at which point the timeout is automatically destroyed and the function will not be called again. The first call to the function will be at the end of the first interval. Note that timeout functions may be delayed, due to the processing of other event sources. Thus they should not be relied on for precise timing. After each call to the timeout function, the time of the next timeout is recalculated based on the current time and the given interval (it does not try to 'catch up' time lost in delays).

gobject.io_add_watch

```
def gobject.io_add_watch(fd, condition, callback, ...)
```

fd :	a Python file object or an integer file descriptor ID
condition :	a condition mask
callback :	a function to call

`...` :	additional arguments to pass to *`callback`*
Returns :	an integer ID of the event source

The `gobject.io_add_watch()` function arranges for the file (specified by *`fd`*) to be monitored by the main loop for the specified *`condition`*. *`fd`* may be a Python file object or an integer file descriptor. The value of condition is a combination of:

`gobject.IO_IN`	There is data to read.
`gobject.IO_OUT`	Data can be written (without blocking).
`gobject.IO_PRI`	There is urgent data to read.
`gobject.IO_ERR`	Error condition.
`gobject.IO_HUP`	Hung up (the connection has been broken, usually for pipes and sockets).

Additional arguments to pass to *`callback`* can be specified after *`callback`*. The idle priority may be specified as a keyword−value pair with the keyword "priority". The signature of the callback function is:

```
  def callback(source, cb_condition, ...)
```

where *`source`* is *`fd`*, the file descriptor; *`cb_condition`* is the condition that triggered the signal; and, `...` are the zero or more arguments that were passed to the `gobject.io_add_watch()` function.

If the callback function returns `FALSE` it will be automatically removed from the list of event sources and will not be called again. If it returns `TRUE` it will be called again when the condition is matched.

gobject.source_remove

```
    def gobject.source_remove(tag)
```

`tag` :	an integer ID
Returns :	`TRUE` if the event source was removed

The `gobject.source_remove()` function removes the event source specified by tag (as returned by the `gobject.idle_add()`, `gobject.timeout_add()` and `gobject.io_add_watch()` functions)

gobject.main_context_default

```
    def gobject.main_context_default()
```

Returns :	the default gobject.MainContext object

The `gobject.main_context_default()` function returns the default gobject.MainContext object.

The gtk Class Reference

Table of Contents

gtk Constants

gtk Constants — the built–in constants of the gtk module

Synopsis

Description

GTK Accel Flags Constants

The Accel Flags constants are a set of bit–flags that specify characteristics of the accelerator.

`gtk.ACCEL_VISIBLE`	if set, the accelerator is visible in a label
`gtk.ACCEL_LOCKED`	If set the accelerator cannot be changed by the user.
`gtk.ACCEL_MASK`	A mask for the Accel Flags

GTK Anchor Type Constants

The Anchor Type constants specify the anchor point of a widget.

```
gtk.ANCHOR_CENTER
gtk.ANCHOR_NORTH
gtk.ANCHOR_NORTH_WEST
gtk.ANCHOR_NORTH_EAST
gtk.ANCHOR_SOUTH
gtk.ANCHOR_SOUTH_WEST
gtk.ANCHOR_SOUTH_EAST
gtk.ANCHOR_WEST
gtk.ANCHOR_EAST,
```

gtk.ANCHOR_N	Same as gtk.ANCHOR_NORTH
gtk.ANCHOR_NW	Same as gtk.ANCHOR_NORTH_WEST
gtk.ANCHOR_NE	Same as gtk.ANCHOR_NORTH_EAST
gtk.ANCHOR_S	Same as gtk.ANCHOR_SOUTH
gtk.ANCHOR_SW	Same as gtk.ANCHOR_SOUTH_WEST
gtk.ANCHOR_SE	Same as gtk.ANCHOR_SOUTH_EAST
gtk.ANCHOR_W	Same as gtk.ANCHOR_WEST
gtk.ANCHOR_E	Same as gtk.ANCHOR_EAST

GTK Arrow Type Constants

The Arrow Type constants specify the direction a gtk.Arrow should point.

gtk.ARROW_UP	Represents an upward pointing arrow.
gtk.ARROW_DOWN	Represents a downward pointing arrow.
gtk.ARROW_LEFT	Represents a left pointing arrow.
gtk.ARROW_RIGHT	Represents a right pointing arrow.

GTK Attach Flag Options Constants

The Attach Flag Options constants are a set of bit–flags that specify the expansion properties that a widget will have when it (or its parent) is resized.

gtk.EXPAND	The widget should expand to take up any extra space in its container that has been allocated.
gtk.SHRINK	The widget should shrink as and when possible.
gtk.FILL	The widget should fill the space allocated to it.

GTK ButtonBox Style Constants

The ButtonBox Style constants specify the style that a gtk.ButtonBox uses to layout the buttons it contains. (See also: gtk.VButtonBox and gtk.HButtonBox).

gtk.BUTTONBOX_DEFAULT_STYLE	Default packing.
gtk.BUTTONBOX_SPREAD	Buttons are evenly spread across the ButtonBox.
gtk.BUTTONBOX_EDGE	Buttons are placed at the edges of the ButtonBox.
gtk.BUTTONBOX_START	Buttons are grouped toward the start of box, (on the left for a HBox, or the top for a VBox).
gtk.BUTTONBOX_END	Buttons are grouped toward the end of a box, (on the right for a HBox, or the bottom for a VBox).

GTK Buttons Type Constants

The Buttons Type constants specify the pre–defined sets of buttons for the dialog. If none of these choices are appropriate, simply use gtk.BUTTONS_NONE then call the add_buttons() method.

gtk.BUTTONS_NONE	no buttons at all
gtk.BUTTONS_OK	an OK button
gtk.BUTTONS_CLOSE	a Close button

`gtk.BUTTONS_CANCEL`	a Cancel button
`gtk.BUTTONS_YES_NO`	Yes and No buttons
`gtk.BUTTONS_OK_CANCEL`	OK and Cancel buttons

GTK Calendar Display Options Constants

The Calendar Display Options constants are a set of bit–flags that specify the display and behavior of a `gtk.Calendar`.

`gtk.CALENDAR_SHOW_HEADING`	Specifies that the month and year should be displayed.
`gtk.CALENDAR_SHOW_DAY_NAMES`	Specifies that three letter day descriptions should be present.
`gtk.CALENDAR_NO_MONTH_CHANGE`	Prevents the user from switching months with the calendar.
`gtk.CALENDAR_SHOW_WEEK_NUMBERS`	Displays each week numbers of the current year, down the left side of the calendar.
`gtk.CALENDAR_WEEK_START_MONDAY`	Since GTK+ 2.4, this option is deprecated and ignored by GTK+. The information on which day the calendar week starts is derived from the locale.

GTK CellRenderer Mode Constants

The CellRenderer Mode constants specify how the user can interact with a particular cell.

`gtk.CELL_RENDERER_MODE_INERT`	The cell is just for display and cannot be interacted with. Note that this doesn't mean that e.g. the row being drawn can't be selected –– just that a particular element of it cannot be individually modified.
`gtk.CELL_RENDERER_MODE_ACTIVATABLE`	The cell can be clicked.
`gtk.CELL_RENDERER_MODE_EDITABLE`	The cell can be edited or otherwise modified.

GTK CellRenderer State Constants

The CellRenderer State constants specify how a cell is to be rendered.

`gtk.CELL_RENDERER_SELECTED`	The cell is currently selected, and probably has a selection colored background to render to.
`gtk.CELL_RENDERER_PRELIT`	The mouse is hovering over the cell.
`gtk.CELL_RENDERER_INSENSITIVE`	The cell is drawn in an insensitive manner
`gtk.CELL_RENDERER_SORTED`	The cell is in a sorted row
`gtk.CELL_RENDERER_FOCUSED`	The cell has the focus.

GTK Corner Type Constants

The Corner Type constants specify the corner a child widget should be placed in when packed into a `gtk.ScrolledWindow`. This is effectively the opposite of where the scroll bars are placed.

`gtk.CORNER_TOP_LEFT`	Place the scrollbars on the right and bottom of the widget (default behavior).
`gtk.CORNER_BOTTOM_LEFT`	Place the scrollbars on the top and right of the widget.
`gtk.CORNER_TOP_RIGHT`	Place the scrollbars on the left and bottom of the widget.

`gtk.CORNER_BOTTOM_RIGHT`	Place the scrollbars on the top and left of the widget.

GTK Curve Type Constants

The Curve Type constants specify the type of curve to use for a `gtk.Curve`.

`gtk.CURVE_TYPE_LINEAR`	Linear interpolation
`gtk.CURVE_TYPE_SPLINE`	Spline interpolation
`gtk.CURVE_TYPE_FREE`	Free form curve

GTK Debug Flag Constants

The Debug Flag constants are a set of bit–flags that specify the debug options.

`gtk.DEBUG_MISC`
`gtk.DEBUG_PLUGSOCKET`
`gtk.DEBUG_TEXT`
`gtk.DEBUG_TREE`
`gtk.DEBUG_UPDATES`
`gtk.DEBUG_KEYBINDINGS`
`gtk.DEBUG_MULTIHEAD`

GTK Delete Type Constants

The Delete Type constants specify the deletion type.

`gtk.DELETE_CHARS`	Delete a character at the cursor
`gtk.DELETE_WORD_ENDS`	Delete from the cursor to the end of a word
`gtk.DELETE_WORDS`	Delete a number of words
`gtk.DELETE_DISPLAY_LINES`	Delete a single line at the cursor
`gtk.DELETE_DISPLAY_LINE_ENDS,`	Delete from the cursor to the end of the line.
`gtk.DELETE_PARAGRAPH_ENDS`	Delete from the cursor to a paragraph end (usually to the period)
`gtk.DELETE_PARAGRAPHS`	Delete several complete paragraphs at the cursor
`gtk.DELETE_WHITESPACE`	Delete the whitespace at the cursor.

GTK Dest Defaults Constants

The Dest Defaults constants are a set of bit–flags that specify the various types of action that will be taken on behalf of the user for a drag destination site.

`gtk.DEST_DEFAULT_MOTION`	If set for a widget, during a drag over this widget will check if the drag matches this widget's list of possible targets and actions. The `gtk.gdk.DragContext.drag_status()` method will be called as appropriate.
`gtk.DEST_DEFAULT_HIGHLIGHT`	If set for a widget, draw a highlight on this widget as long as a drag is over this widget and the widget drag format and action are acceptable.
`gtk.DEST_DEFAULT_DROP`	If set for a widget, when a drop occurs, check if the drag matches this widget's list of possible targets and actions. If so, call the `gtk.Widget.drag_get_data()` method on behalf of the widget.

	Whether or not the drop is successful, call the gtk.gdk.DragContext.finish() method. If the action was a move, then if the drag was successful, then `TRUE` will be passed for the delete parameter to the gtk.gdk.DragContext.finish() method.
`gtk.DEST_DEFAULT_ALL`	If set, specifies that all default actions should be taken.

GTK Dialog Flag Constants

The Dialog Flag constants are a set of bit–flags that specify characteristics of a dialog.

`gtk.DIALOG_MODAL`	If set, the dialog grabs all keyboard events
`gtk.DIALOG_DESTROY_WITH_PARENT`	If set, the dialog is destroyed when its parent is.
`gtk.DIALOG_NO_SEPARATOR`	If set, there is no separator bar above the buttons.

GTK Direction Type Constants

The Direction Type constants specify a direction for moving a cursor or focus.

`gtk.DIR_TAB_FORWARD`	Tab forward.
`gtk.DIR_TAB_BACKWARD`	Tab backward.
`gtk.DIR_UP`	Up.
`gtk.DIR_DOWN`	Down.
`gtk.DIR_LEFT`	Left.
`gtk.DIR_RIGHT`	Right.

GTK Expander Style Constants

The Expander Style constants specify the style of the expanders drawn by a `gtk.TreeView`

`gtk.EXPANDER_COLLAPSED`	The style used for a collapsed subtree.
`gtk.EXPANDER_SEMI_COLLAPSED`	Intermediate style used during animation.
`gtk.EXPANDER_SEMI_EXPANDED`	Intermediate style used during animation.
`gtk.EXPANDER_EXPANDED`	The style used for an expanded subtree.

GTK FileChooser Action Constants

The FileChooser Action constants specify the mode of a `gtk.FileChooser` i.e. whether it is being used to open existing files or to save to a possibly new file.

`gtk.FILE_CHOOSER_ACTION_OPEN`	Indicates open mode. The file chooser will only let the user pick an existing file.
`gtk.FILE_CHOOSER_ACTION_SAVE`	Indicates save mode. The file chooser will let the user pick an existing file, or type in a new filename.
`gtk.FILE_CHOOSER_ACTION_SELECT_FOLDER`	Indicates an Open mode for selecting folders. The file chooser will let the user pick an existing folder.
`gtk.FILE_CHOOSER_ACTION_CREATE_FOLDER`	Indicates a mode for creating a new folder. The file chooser will let the user name an existing or

	new folder.

GTK FileChooser Error Constants

The FileChooser Error constants specify the various errors that can occur while calling `gtk.FileChooser` functions.

`gtk.FILE_CHOOSER_ERROR_NONEXISTENT`	Indicates that a file does not exist.
`gtk.FILE_CHOOSER_ERROR_BAD_FILENAME`	Indicates a malformed filename.

GTK FileFilter Flags Constants

The FileFilter Flags constants are a set of bit–flags that specify the file types to filter the files against.

`gtk.FILE_FILTER_FILENAME`	The full pathname of the file e.g. /tmp/junk.
`gtk.FILE_FILTER_URI`	The full URI of the file e.g. file:///tmp/junk.
`gtk.FILE_FILTER_DISPLAY_NAME`	The simple name of the file e.g. junk.
`gtk.FILE_FILTER_MIME_TYPE`	The MIME type of the file e.g. text/html.

GTK Icon Lookup Flags Constants

The Icon Lookup Flags constants are a set of bit–flags that specify options for the `gtk.IconTheme.lookup_icon()` method

`gtk.ICON_LOOKUP_NO_SVG`	Never return SVG (Scalable Vector Graphics) icons, even if gdk–pixbuf supports them. Cannot be used together with `gtk.ICON_LOOKUP_FORCE_SVG`.
`gtk.ICON_LOOKUP_FORCE_SVG`	Return SVG icons, even if gdk–pixbuf doesn't support them. Cannot be used together with `gtk.ICON_LOOKUP_NO_SVG`.
`gtk.ICON_LOOKUP_USE_BUILTIN`	When passed to the `gtk.IconTheme.lookup_icon()` method includes builtin icons as well as files. For a builtin icon, the `gtk.IconInfo.get_filename()` method returns `None` and you need to call the `gtk.IconInfo.get_builtin_pixbuf()` method.

GTK Icon Size Constants

The Icon Size constants specify the pre–defined sizes of icons for various application uses.

`gtk.ICON_SIZE_INVALID`
`gtk.ICON_SIZE_MENU`
`gtk.ICON_SIZE_SMALL_TOOLBAR`
`gtk.ICON_SIZE_LARGE_TOOLBAR`
`gtk.ICON_SIZE_BUTTON`
`gtk.ICON_SIZE_DND`
`gtk.ICON_SIZE_DIALOG`

GTK IconTheme Error Constants

The IconTheme Error constants specify error codes for gtk.IconTheme operations.

gtk.ICON_THEME_NOT_FOUND	The icon specified does not exist in the theme
gtk.ICON_THEME_FAILED	An unspecified error occurred.

GTK IM Pre–edit Style Constants

The IM Pre–edit Style constants specify the style of input method pre–edit display.

```
gtk.IM_PREEDIT_NOTHING
gtk.IM_PREEDIT_CALLBACK
gtk.IM_PREEDIT_NONE
```

GTK IM Status Style Constants

The IM Status Style constants specify the style of input method display.

```
gtk.IM_STATUS_NOTHING
gtk.IM_STATUS_CALLBACK
gtk.IM_STATUS_NONE
```

GTK Image Type Constants

The Image Type constants specify the type of image in a gtk.Image.

gtk.IMAGE_EMPTY	There is no image displayed by the widget
gtk.IMAGE_PIXMAP	The widget contains a gtk.gdk.Pixmap
gtk.IMAGE_IMAGE	The widget contains a gtk.gdk.Image
gtk.IMAGE_PIXBUF	The widget contains a gtk.gdk.Pixbuf
gtk.IMAGE_STOCK	The widget contains a stock icon name (see the Stock Items reference)
gtk.IMAGE_ICON_SET	The widget contains a gtk.IconSet
gtk.IMAGE_ANIMATION	The widget contains a gtk.gdk.PixbufAnimation

GTK Justification Constants

The Justification constants specify the justification of the text inside a gtk.Label widget. (See also gtk.Alignment).

gtk.JUSTIFY_LEFT	The text is placed at the left edge of the label.
gtk.JUSTIFY_RIGHT	The text is placed at the right edge of the label.
gtk.JUSTIFY_CENTER	The text is placed in the center of the label.
gtk.JUSTIFY_FILL	The text is placed is distributed across the label.

GTK Menu Direction Type Constants

The Menu Direction Type constants specify directional movements within a menu.

`gtk.MENU_DIR_PARENT`	To the parent menu shell.
`gtk.MENU_DIR_CHILD`	To the submenu, if any, associated with the item.
`gtk.MENU_DIR_NEXT`	To the next menu item.
`gtk.MENU_DIR_PREV`	To the previous menu item.

GTK Message Type Constants

The Message Type constants specify the type of message being displayed in the message dialog.

`gtk.MESSAGE_INFO`	Informational message
`gtk.MESSAGE_WARNING`	Nonfatal warning message
`gtk.MESSAGE_QUESTION`	Question requiring a choice
`gtk.MESSAGE_ERROR`	Fatal error message

GTK Metric Type Constants

The Metric Type constants specify the metric used by a `gtk.Ruler`.

`gtk.PIXELS`	Pixels.
`gtk.INCHES`	Inches.
`gtk.CENTIMETERS`	Centimeters.

GTK Movement Step Constants

The Movement Step constants specify the steps used in movement through text.

`gtk.MOVEMENT_LOGICAL_POSITIONS`	move by graphemes
`gtk.MOVEMENT_VISUAL_POSITIONS`	move by graphemes
`gtk.MOVEMENT_WORDS`	move by words
`gtk.MOVEMENT_DISPLAY_LINES`	move by lines(wrapped lines)
`gtk.MOVEMENT_DISPLAY_LINE_ENDS`	move to line ends(wrapped lines)
`gtk.MOVEMENT_PARAGRAPHS`	move by paragraphs(newline–ended lines)
`gtk.MOVEMENT_PARAGRAPH_ENDS`	move to ends of a paragraph
`gtk.MOVEMENT_PAGES`	move by pages
`gtk.MOVEMENT_BUFFER_ENDS`	move to ends of the buffer

GTK Notebook Tab Constants

The Notebook Tab constants specify the tab position to receive focus.

`gtk.NOTEBOOK_TAB_FIRST`	The first `gtk.Notebook` tab
`gtk.NOTEBOOK_TAB_LAST`	The last `gtk.Notebook` tab

GTK Object Flags Constants

The Object Flags constants are a set of bit–flags that specify the state of the `gtk.Object`.

`gtk.IN_DESTRUCTION`	The object is currently being destroyed. This is used internally to prevent reinvocations during destruction.

`gtk.FLOATING`	The object is orphaned.

GTK Orientation Constants

The Orientation constants specify the orientation of widgets which can be switched between horizontal and vertical orientation on the fly, like `gtk.Toolbar`.

`gtk.ORIENTATION_HORIZONTAL`	The widget is in horizontal orientation.
`gtk.ORIENTATION_VERTICAL`	The widget is in vertical orientation.

GTK Pack Type Constants

The Pack Type constants specify the packing location `gtk.Box` children. (See: `gtk.VBox`, `gtk.HBox`, and `gtk.ButtonBox`).

`gtk.PACK_START`	The child is packed into the start of the box
`gtk.PACK_END`	The child is packed into the end of the box

GTK Path Priority Type Constants

The Path Priority Type constants are a set of bit−flags that specify the priority of path lookup.

`gtk.PATH_PRIO_LOWEST`
`gtk.PATH_PRIO_GTK`
`gtk.PATH_PRIO_APPLICATION`
`gtk.PATH_PRIO_THEME`
`gtk.PATH_PRIO_RC`
`gtk.PATH_PRIO_HIGHEST`

GTK Path Type Constants

The Path Type constants specify

`gtk.PATH_WIDGET`
`gtk.PATH_WIDGET_CLASS`
`gtk.PATH_CLASS`

GTK Policy Type Constants

The Policy Type constants specify when a scroll bar will be visible.

gtk.POLICY_ALWAYS	the scrollbar is always present
gtk.POLICY_AUTOMATIC	the scrollbar is present only if needed i.e. the contents are larger than the window
gtk.POLICY_NEVER	the scrollbar is never present

GTK Position Type Constants

The Position Type constants specify

`gtk.POS_LEFT`	The feature is at the left edge.
`gtk.POS_RIGHT`	The feature is at the right edge.
`gtk.POS_TOP`	The feature is at the top edge.
`gtk.POS_BOTTOM`	The feature is at the bottom edge

GTK ProgressBar Orientation Constants

The ProgressBar Orientation constants specify the orientation and growth direction for a visible progress bar.

`gtk.PROGRESS_LEFT_TO_RIGHT`	A horizontal progress bar growing from left to right.
`gtk.PROGRESS_RIGHT_TO_LEFT`	A horizontal progress bar growing from right to left.
`gtk.PROGRESS_BOTTOM_TO_TOP`	A vertical progress bar growing from bottom to top.
`gtk.PROGRESS_TOP_TO_BOTTOM`	A vertical progress bar growing from top to bottom.

GTK ProgressBar Style Constants

The ProgressBar Style constants specify the style of the `gtk.ProgressBar` display.

`gtk.PROGRESS_CONTINUOUS`	The progress bar grows in a smooth, continuous manner.
`gtk.PROGRESS_DISCRETE`	The progress bar grows in discrete, visible blocks.

GTK RC Flags Constants

The>RC Flags constants are a set of bit–flags that specify which fields of a `gtk.RcStyle` have been set for each state.

`gtk.RC_FG`	If present, the foreground color has been set for this state.
`gtk.RC_BG`	If present, the background color has been set for this state.
`gtk.RC_TEXT`	If present, the text color has been set for this state.
`gtk.RC_BASE`	If present, the base color has been set for this state.

GTK RC Token Type Constants

The RC Token Type constants specify the tokens in the RC file. It is exposed so that theme engines can reuse these tokens when parsing the theme–engine specific portions of a RC file.

```
gtk.RC_TOKEN_INVALID
gtk.RC_TOKEN_INCLUDE
gtk.RC_TOKEN_NORMAL
gtk.RC_TOKEN_ACTIVE
gtk.RC_TOKEN_PRELIGHT
gtk.RC_TOKEN_SELECTED
gtk.RC_TOKEN_INSENSITIVE
gtk.RC_TOKEN_FG
gtk.RC_TOKEN_BG
gtk.RC_TOKEN_TEXT
gtk.RC_TOKEN_BASE
gtk.RC_TOKEN_XTHICKNESS
```

`gtk.RC_TOKEN_YTHICKNESS`
`gtk.RC_TOKEN_FONT`
`gtk.RC_TOKEN_FONTSET`
`gtk.RC_TOKEN_FONT_NAME`
`gtk.RC_TOKEN_BG_PIXMAP`
`gtk.RC_TOKEN_PIXMAP_PATH`
`gtk.RC_TOKEN_STYLE`
`gtk.RC_TOKEN_BINDING`
`gtk.RC_TOKEN_BIND`
`gtk.RC_TOKEN_WIDGET`
`gtk.RC_TOKEN_WIDGET_CLASS`
`gtk.RC_TOKEN_CLASS`
`gtk.RC_TOKEN_LOWEST`
`gtk.RC_TOKEN_GTK`
`gtk.RC_TOKEN_APPLICATION`
`gtk.RC_TOKEN_THEME`
`gtk.RC_TOKEN_RC`
`gtk.RC_TOKEN_HIGHEST`
`gtk.RC_TOKEN_ENGINE`
`gtk.RC_TOKEN_MODULE_PATH`
`gtk.RC_TOKEN_IM_MODULE_PATH`
`gtk.RC_TOKEN_IM_MODULE_FILE`
`gtk.RC_TOKEN_STOCK`
`gtk.RC_TOKEN_LTR`
`gtk.RC_TOKEN_RTL`
`gtk.RC_TOKEN_LAST`

GTK Relief Style Constants

The Relief Style constants specify

`gtk.RELIEF_NORMAL`	Draw a normal relief.
`gtk.RELIEF_HALF`	Draw a half relief.
`gtk.RELIEF_NONE`	Draw no relief.

GTK Resize Mode Constants

The Resize Mode constants specify how resize requests are handled by a widget.

`gtk.RESIZE_PARENT`	Pass resize request to the parent
`gtk.RESIZE_QUEUE`	Queue resizes on this widget
`gtk.RESIZE_IMMEDIATE`	Perform the resizes now

GTK Response Type Constants

The Response Type constants specify pre−defined response values.

	gtk.RESPONSE_NONE
	gtk.RESPONSE_REJECT
	gtk.RESPONSE_ACCEPT
	gtk.RESPONSE_DELETE_EVENT
	gtk.RESPONSE_OK
	gtk.RESPONSE_CANCEL
	gtk.RESPONSE_CLOSE
	gtk.RESPONSE_YES
	gtk.RESPONSE_NO
	gtk.RESPONSE_APPLY
	gtk.RESPONSE_HELP

GTK Scroll Step Constants

The Scroll Step constants specify the size of the scroll movements.

gtk.SCROLL_STEPS	Scroll up or down in step increments
gtk.SCROLL_PAGES,	Scroll up or down in page increments
gtk.SCROLL_ENDS	Scroll to the beginning or end
gtk.SCROLL_HORIZONTAL_STEPS	Scroll left or right in step increments
gtk.SCROLL_HORIZONTAL_PAGES	Scroll left or right in step increments
gtk.SCROLL_HORIZONTAL_ENDS	Scroll to the far left end or far right end.

GTK Scroll Type Constants

The Scroll Type constants specify the type of scroll \movement.

```
gtk.SCROLL_NONE
gtk.SCROLL_JUMP
gtk.SCROLL_STEP_BACKWARD
gtk.SCROLL_STEP_FORWARD
gtk.SCROLL_PAGE_BACKWARD
gtk.SCROLL_PAGE_FORWARD
gtk.SCROLL_STEP_UP
gtk.SCROLL_STEP_DOWN
gtk.SCROLL_PAGE_UP
gtk.SCROLL_PAGE_DOWN
gtk.SCROLL_STEP_LEFT,
gtk.SCROLL_STEP_RIGHT
gtk.SCROLL_PAGE_LEFT
gtk.SCROLL_PAGE_RIGHT
gtk.SCROLL_START
gtk.SCROLL_END
```

GTK Selection Mode Constants

The Selection Mode constants specify the mode of selection in a gtk.Treeview

gtk.SELECTION_NONE	No selection allowed.
gtk.SELECTION_SINGLE	A single selection allowed by clicking.
gtk.SELECTION_BROWSE	A single selection allowed by browsing with the pointer.
gtk.SELECTION_MULTIPLE	Multiple items can be selected at once.
gtk.SELECTION_EXTENDED	Deprecated.

GTK Shadow Type Constants

The Shadow Type constants specify the appearance of an outline typically provided by a gtk.Frame.

gtk.SHADOW_NONE	No outline.
gtk.SHADOW_IN	The outline is beveled inward.
gtk.SHADOW_OUT	The outline is beveled outward like a button.
gtk.SHADOW_ETCHED_IN	The outline itself is an inward bevel, but the frame bevels outward
gtk.SHADOW_ETCHED_OUT	The outline itself is an outward bevel, but the frame bevels inward

GTK SizeGroup Mode Constants

The SizeGroup Mode constants specify the directions in which the size group affects the requested sizes of its component widgets.

gtk.SIZE_GROUP_NONE	The group has no affect
gtk.SIZE_GROUP_HORIZONTAL	The group affects horizontal requisition
gtk.SIZE_GROUP_VERTICAL	The group affects vertical requisition
gtk.SIZE_GROUP_BOTH	The group affects both horizontal and vertical requisition

GTK Sort Type Constants

The Sort Type constants specify he direction of a sort.

gtk.SORT_ASCENDING	Sorting is in ascending order.
gtk.SORT_DESCENDING	Sorting is in descending order.

GTK SpinButton Update Policy Constants

The SpinButton Update Policy constants specify the update policy for a gtk.SpinButton.

gtk.UPDATE_ALWAYS	When refreshing a gtk.SpinButton, the value is always displayed.
gtk.UPDATE_IF_VALID	When refreshing a gtk.SpinButton, the value is only displayed if it is valid within the bounds of the spin button's gtk.Adjustment.

GTK Spin Type Constants

The Spin Type constants specify the step movement of a gtk.SpinButton.

`gtk.SPIN_STEP_FORWARD`	Spin a `gtk.SpinButton` forward by the step value of the spin button's `gtk.Adjustment`.
`gtk.SPIN_STEP_BACKWARD`	Spin a `gtk.SpinButton` backward by the step value of the spin button's `gtk.Adjustment`.
`gtk.SPIN_PAGE_FORWARD`	Spin a `gtk.SpinButton` forward by the page value of the spin button's `gtk.Adjustment`.
`gtk.SPIN_PAGE_BACKWARD`	Spin a `gtk.SpinButton` backward by the page value of the spin button's `gtk.Adjustment`.
`gtk.SPIN_HOME`	Set the spin button's value to the minimum possible value specified by its `gtk.Adjustment`
`gtk.SPIN_END`	Set the spin button's value to the maximum possible value specified by its `gtk.Adjustment`
`gtk.SPIN_USER_DEFINED`	The programmer must specify the exact amount to spin the `gtk.SpinButton`.

GTK State Type Constants

The State Type constants specify the current state of a widget; the state determines how the widget is drawn. The State Type constants are also used to identify different colors in a `gtk.Style` for drawing, so states can be used for subparts of a widget as well as entire widgets.

`gtk.STATE_NORMAL`	State during normal operation.
`gtk.STATE_ACTIVE`	State of a currently active widget, such as a depressed button.
`gtk.STATE_PRELIGHT`	State indicating that the mouse pointer is over the widget and the widget will respond to mouse clicks.
`gtk.STATE_SELECTED`	State of a selected item, such the selected row in a list.
`gtk.STATE_INSENSITIVE`	State indicating that the widget is unresponsive to user actions.

GTK Target Flags Constants

The Target Flags constants are a set of bit–flags that specify constraints on the target of a drag operation.

`gtk.TARGET_SAME_APP`	If this is set, the target will only be selected for drags within a single application.
`gtk.TARGET_SAME_WIDGET`	f this is set, the target will only be selected for drags within a single widget.

GTK Text Direction Constants

The Text Direction constants specify the direction of the text.

`gtk.TEXT_DIR_NONE`	Text direction not specified.
`gtk.TEXT_DIR_LTR,`	Left to right direction.
`gtk.TEXT_DIR_RTL`	Right to left direction

GTK Text Search Flags Constants Constants

The Text Search Flags constants are a set of bit–flags that specify what types of text are suitable for search matches in a `gtk.TextView`.

`gtk.TEXT_SEARCH_VISIBLE_ONLY`	Only visible text can match the search criteria.

`gtk.TEXT_SEARCH_TEXT_ONLY`	Both visible and invisible text can match the search criteria.

GTK Text Window Type Constants

The Text Window Type constants specify the `gtk.gdk.Window` objects that make up a `gtk.TextView`. See the `gtk.TextView.get_window()` method for more detail.

`gtk.TEXT_WINDOW_WIDGET`	The `gtk.gdk.Window` of the `gtk.TextView` widget.
`gtk.TEXT_WINDOW_TEXT`	The `gtk.gdk.Window` that contains the text in the `gtk.TextView`.
`gtk.TEXT_WINDOW_LEFT`	The left child `gtk.gdk.Window` of the `gtk.TextView`.
`gtk.TEXT_WINDOW_RIGHT`	The right child `gtk.gdk.Window` of the `gtk.TextView`.
`gtk.TEXT_WINDOW_TOP`	The top child `gtk.gdk.Window` of the `gtk.TextView`.
`gtk.TEXT_WINDOW_BOTTOM`	The bottom child `gtk.gdk.Window` of the `gtk.TextView`.

GTK Toolbar Space Style Constants

The Toolbar Space Style constants specify whether a spacer is displayed as a vertical line or space.

`gtk.TOOLBAR_SPACE_EMPTY,`	Show as an empty space
`gtk.TOOLBAR_SPACE_LINE`	Show as a vertical line.

GTK Toolbar Style Constants

The Toolbar Style constants specify the appearance of a `gtk.Toolbar`. Note that setting the toolbar style overrides the user's preferences for the default toolbar style.

`gtk.TOOLBAR_ICONS`	Buttons display only icons in the toolbar.
`gtk.TOOLBAR_TEXT`	Buttons display only text labels in the toolbar.
`gtk.TOOLBAR_BOTH`	Buttons display text and icons in the toolbar.
`gtk.TOOLBAR_BOTH_HORIZ`	Buttons display icons and text alongside each other, rather than vertically stacked

GTK TreeModel Flags Constants

The TreeModel Flags constants are a set of bit−flags that specify various properties of a `gtk.TreeModel`. They are returned by the `gtk.TreeModel.get_flags()` method, and must be static for the lifetime of the object. A more complete description of `gtk.TREE_MODEL_ITERS_PERSIST` can be found in the `gtk.TreeView` reference description.

`gtk.TREE_MODEL_ITERS_PERSIST`	Iterators survive all signals emitted by the tree.
`gtk.TREE_MODEL_LIST_ONLY`	The model is a list only, and never has children

GTK TreeView Drop Position Constants

The TreeView Drop Position constants specify where a dropped row goes.

`gtk.TREE_VIEW_DROP_BEFORE`	The dropped item goes before the row it's dropped on.
`gtk.TREE_VIEW_DROP_AFTER`	The dropped item goes after the row it's dropped on.
`gtk.TREE_VIEW_DROP_INTO_OR_BEFORE`	The dropped item becomes a child of the row it's dropped on. Fallback to goes before.

`gtk.TREE_VIEW_DROP_INTO_OR_AFTER`	The dropped item becomes a child of the row it's dropped on. Fallback to goes after.

GTK TreeViewColumn Sizing Constants

The TreeViewColumn Sizing constants specify the sizing method the column uses to determine its width. Please note that `gtk.TREE_VIEW_COLUMN_AUTOSIZE` are inefficient for large views, and can make columns appear choppy.

`gtk.TREE_VIEW_COLUMN_GROW_ONLY`	Columns only get bigger in reaction to changes in the model
`gtk.TREE_VIEW_COLUMN_AUTOSIZE`	Columns resize to be the optimal size every time the model changes.
`gtk.TREE_VIEW_COLUMN_FIXED`	Columns are a fixed numbers of pixels wide.

GTK UIManager Item Type Constants

The UIManager Item Type constants specify what UI element to create.

`gtk.UI_MANAGER_AUTO`	Pick the type of the UI element according to context.
`gtk.UI_MANAGER_MENUBAR`	Create a menubar.
`gtk.UI_MANAGER_MENU`	Create a menu.
`gtk.UI_MANAGER_TOOLBAR`	Create a toolbar.
`gtk.UI_MANAGER_PLACEHOLDER`	Insert a placeholder.
`gtk.UI_MANAGER_POPUP`	Create a popup menu.
`gtk.UI_MANAGER_MENUITEM`	Create a menuitem.
`gtk.UI_MANAGER_TOOLITEM`	Create a toolitem.
`gtk.UI_MANAGER_SEPARATOR`	Create a separator.
`gtk.UI_MANAGER_ACCELERATOR`	Install an accelerator.

GTK Update Type Constants

The Update Type constants specify the update policy of a `gtk.Range` and `gtk.SpinButton`.

`gtk.UPDATE_CONTINUOUS`	Update the display continuously as the pointer is moved.
`gtk.UPDATE_DISCONTINUOUS`	Update the display at intervals while the pointer is being moved.
`gtk.UPDATE_DELAYED`	Update the display after the pointer has finished moving.

GTK Version Constants

The Version constants specify the versions of `GTK+` and `PyGTK` as a 3–tuple containing the major, minor and patch release numbers.

`gtk.gtk_version`	A 3–tuple containing the `GTK+` (major, minor, patch) release numbers.
`gtk.pygtk_version`	A 3–tuple containing the `PyGTK` (major, minor, patch) release numbers.

GTK Widget Flags Constants

The Widget Flags constants are a set of bit–flags that specify certain properties of the widget.

gtk.TOPLEVEL	Widgets without a real parent, as there are gtk.Window and gtk.Menu objects that have this flag set throughout their lifetime. Toplevel widgets always contain their own gtk.gdk.Window.
gtk.NO_WINDOW	Indicative for a widget that does not provide its own gtk.gdk.Window. Visible action (e.g. drawing) is performed on the parent's gtk.gdk.Window.
gtk.REALIZED	Set by the gtk.Widget.realize() method , unset by the gtk.Widget.unrealize() method. A realized widget has an associated gtk.gdk.Window.
gtk.MAPPED	Set by the gtk.Widget.map() method, unset by the gtk.Widget.unmap() method. Only realized widgets can be mapped. It means that the gtk.Window.show() method has been called on the widgets window(s).
gtk.VISIBLE	Set by the gtk.Widget.show() method, unset by the gtk.Widget.hide() method. Implies that a widget will be mapped as soon as its parent is mapped.
gtk.SENSITIVE	Set and unset by the gtk.Widget.set_sensitive() method. The sensitivity of a widget determines whether it will receive certain events (e.g. button or key presses). One premise for the widgets sensitivity is to have this flag set.
gtk.PARENT_SENSITIVE	Set and unset by the gtk.Widget.set_sensitive() method operations on the parents of the widget. This is the second premise for the widgets sensitivity. Once it has gtk.SENSITIVE and gtk.PARENT_SENSITIVE set, its state is effectively sensitive.
gtk.CAN_FOCUS	Determines if a widget is able to handle focus grabs.
gtk.HAS_FOCUS	Set by the gtk.Widget.grab_focus() method for widgets that also have gtk.CAN_FOCUS set. The flag will be unset once another widget grabs the focus.
gtk.CAN_DEFAULT	The widget is allowed to receive the default action via the gtk.Widget.grab_default() method.
gtk.HAS_DEFAULT	The widget currently is receiving the default action.
gtk.HAS_GRAB	Set by the gtk.Widget.grab_add() method, unset by the gtk.Widget.grab_remove() method. It means that the widget is in the grab_widgets stack, and will be the preferred one for receiving events other than ones of cosmetic value.
gtk.RC_STYLE	Indicates that the widgets style has been looked up through the rc mechanism. It does not imply that the widget actually had a style defined through the rc mechanism.
gtk.COMPOSITE_CHILD	Indicates that the widget is a composite child of its parent; see the gtk.widget_push_composite_child() and gtk.widget_pop_composite_child() functions.
gtk.NO_REPARENT	Unused.
gtk.APP_PAINTABLE	Set and unset by the gtk.Widget.set_app_paintable() method. Must be set on widgets whose window the application directly draws on, in order to keep PyGTK and GTK+ from overwriting the drawn stuff.
gtk.RECEIVES_DEFAULT	The widget when focused will receive the default action and have gtk.HAS_DEFAULT set even if there is a different widget set as default.
gtk.DOUBLE_BUFFERED	Set and unset by the gtk.Widget.set_double_buffered() method. Indicates that exposes done on the widget should be double−buffered.
gtk.NO_SHOW_ALL	If TRUE, the show_all() and hide_all() methods do not affect the

	widget.

GTK Widget Help Type Constants

The Widget Help Type constants specify the help type of the widget.

`gtk.WIDGET_HELP_TOOLTIP`	Tooltip help.
`gtk.WIDGET_HELP_WHATS_THIS`	What's this help.

GTK Window Position Constants

The Window Position constants specify hints for initial window placement.

`gtk.WIN_POS_NONE`	No influence is made on placement.
`gtk.WIN_POS_CENTER`	Windows should be placed in the center of the screen.
`gtk.WIN_POS_MOUSE`	Windows should be placed at the current mouse position.
`gtk.WIN_POS_CENTER_ALWAYS`	Keep window centered as it changes size, etc.
`gtk.WIN_POS_CENTER_ON_PARENT`	Center the window on its transient parent (see the `gtk.Window.set_transient_for()`) method.

GTK Window Type Constants

The Window Type constants specify the type of a `gtk.Window`. Most things you'd consider a "window" should have type `gtk.WINDOW_TOPLEVEL`; windows with this type are managed by the window manager and have a frame by default (call the `set_decorated()` method to toggle the frame). Windows with type `gtk.WINDOW_POPUP` are ignored by the window manager; window manager keybindings won't work on them, the window manager won't decorate the window with a frame, many GTK+ features that rely on the window manager will not work (e.g. resize grips and maximization/minimization). `gtk.WINDOW_POPUP` is used to implement widgets such as `gtk.Menu` or tooltips that you normally don't think of as windows per se. Nearly all windows should be `gtk.WINDOW_TOPLEVEL`. In particular, do not use `gtk.WINDOW_POPUP` just to turn off the window borders; use the `gtk.Window.set_decorated()` method for that.

`gtk.WINDOW_TOPLEVEL`	A regular window, such as a dialog.
`gtk.WINDOW_POPUP`	A special window such as a tooltip.

GTK Wrap Mode Constants

The Wrap Mode constants specify the type of line wrapping in a `gtk.TextView`.

`gtk.WRAP_NONE`	Do not wrap lines – just make the text area wider
`gtk.WRAP_CHAR`	Wrap text, breaking lines anywhere the cursor can appear (usually between characters)
`gtk.WRAP_WORD`	Wrap text, breaking lines in between words
`gtk.WRAP_WORD_CHAR`	Wrap text, breaking lines in between words, or if that is not enough, also between graphemes.

Prev | Up | Next
Stock Items | Home | The gtk.gdk Class Reference

gtk Functions

gtk Functions

gtk Functions — miscellaneous functions

Synopsis

gtk.AboutDialog Functions

```
    def gtk.about_dialog_set_email_hook(func, data)
     def gtk.about_dialog_set_url_hook(func, data)
```

gtk.AccelGroup Functions

```
    def gtk.accelerator_valid(keyval, modifiers)
     def gtk.accelerator_parse(accelerator)
     def gtk.accelerator_name(accelerator_key, accelerator_mods)
     def gtk.accelerator_set_default_mod_mask(default_mod_mask)
     def gtk.accelerator_get_default_mod_mask()
     def gtk.accelerator_get_label(accelerator_key, accelerator_mods)
     def gtk.accel_map_add_entry(accel_path, accel_key, accel_mods)
     def gtk.accel_map_lookup_entry(accel_path)
     def gtk.accel_map_change_entry(accel_path, accel_key, accel_mods, replace)
     def gtk.accel_map_load(file_name)
     def gtk.accel_map_save(file_name)
     def gtk.accel_map_load_fd(fd)
     def gtk.accel_map_save_fd(fd)
     def gtk.accel_map_lock_path(accel_path)
     def gtk.accel_map_unlock_path(accel_path)
     def gtk.accel_map_add_filter(filter_pattern)
     def gtk.accel_groups_from_object(object)
```

gtk.CellView Functions

```
    def gtk.cell_view_new_with_text(text)
     def gtk.cell_view_new_with_markup(markup)
     def gtk.cell_view_new_with_pixbuf(pixbuf)
```

gtk.Clipboard Functions

```
    def gtk.clipboard_get()
```

gtk.ColorSelection Functions

```
    def gtk.color_selection_palette_from_string(str)
     def gtk.color_selection_palette_to_string(colors)
```

gtk.ComboBox Functions

```
    def gtk.combo_box_new_text()
```

gtk.ComboBoxEntry Functions

```
def gtk.combo_box_entry_new_text()
```

gtk.Container Functions

```
def gtk.container_class_install_child_property(klass, property_id, pspec)
 def gtk.container_class_list_child_properties(klass)
```

gtk.Expander Functions

```
def gtk.expander_new_with_mnemonic(label)
```

gtk.IconFactory Functions

```
def gtk.icon_factory_lookup_default(stock_id)
```

gtk.IconSource Functions

```
def gtk.icon_size_lookup(icon_size)
 def gtk.icon_size_lookup_for_settings(settings, icon_size)
 def gtk.icon_size_register(name, width, height)
 def gtk.icon_size_register_alias(alias, target)
 def gtk.icon_size_from_name(name)
 def gtk.icon_size_get_name(size)
```

gtk.IconTheme Functions

```
def gtk.icon_theme_get_default()
 def gtk.icon_theme_get_for_screen(screen)
 def gtk.icon_theme_add_builtin_icon(icon_name, size, pixbuf)
```

gtk.Image Functions

```
def gtk.image_new_from_stock(stock_id, size)
 def gtk.image_new_from_icon_set(icon_set, size)
 def gtk.image_new_from_animation(animation)
 def gtk.image_new_from_icon_name(icon_name, size)
```

gtk.ItemFactory Functions

```
def gtk.item_factory_from_widget(widget)
 def gtk.item_factory_path_from_widget(widget)
```

gtk.Object Functions

```
def gtk.bindings_activate(object, keyval, modifiers)
 def gtk.bindings_activate_event(object, event)
```

gtk.Plug Functions

```
def gtk.plug_new_for_display(display, socket_id)
```

gtk.RcStyle Functions

```
    def gtk.rc_add_default_file(filename)
     def gtk.rc_set_default_files(filenames)
     def gtk.rc_get_default_files()
     def gtk.rc_get_style_by_paths(settings, widget_path, class_path, type)
     def gtk.rc_reparse_all_for_settings(settings, force_load)
     def gtk.rc_reset_styles(settings)
     def gtk.rc_parse(filename)
     def gtk.rc_parse_string(rc_string)
     def gtk.rc_reparse_all()
     def gtk.rc_find_module_in_path(module_file)
     def gtk.rc_get_theme_dir()
     def gtk.rc_get_module_dir()
     def gtk.rc_get_im_module_path()
     def gtk.rc_get_im_module_file()
```

gtk.SelectionData Functions

```
    def gtk.selection_owner_set_for_display(display, widget, selection, time=0)
     def gtk.target_list_add_image_targets(list=None, info=0, writable=FALSE)
     def gtk.target_list_add_text_targets(list=None, info=0)
     def gtk.target_list_add_uri_targets(list=None, info=0)
```

gtk.Settings Functions

```
    def gtk.settings_get_default()
     def gtk.settings_get_for_screen(screen)
```

gtk.Tooltips Functions

```
    def gtk.tooltips_data_get(widget)
```

gtk.TreeModel Functions

```
    def gtk.tree_row_reference_inserted(proxy, path)
     def gtk.tree_row_reference_deleted(proxy, path)
```

gtk.Widget Functions

```
    def gtk.widget_push_colormap(cmap)
     def gtk.widget_push_composite_child()
     def gtk.widget_pop_composite_child()
     def gtk.widget_pop_colormap()
     def gtk.widget_get_default_style()
     def gtk.widget_set_default_colormap(colormap)
     def gtk.widget_get_default_colormap()
     def gtk.widget_get_default_visual()
     def gtk.widget_set_default_direction(dir)
     def gtk.widget_get_default_direction()
     def gtk.widget_list_style_properties(cmap)
     def gtk.widget_class_install_style_property(widget, pspec)
```

gtk.Window Functions

```
    def gtk.window_set_default_icon(icon)
     def gtk.window_set_default_icon_from_file(filename)
     def gtk.window_set_default_icon_list(...)
     def gtk.window_get_default_icon_list()
     def gtk.window_set_auto_startup_notification(setting)
     def gtk.window_list_toplevels()
     def gtk.window_set_default_icon_name(name)
```

Stock Item Functions

```
   def gtk.stock_add(items)
    def gtk.stock_lookup(stock_id)
    def gtk.stock_list_ids()
```

Miscellaneous Functions

```
   def gtk.binding_entry_add_signal(object, keyval, modifiers, signal_name, ...)
    def gtk.check_version(required_major, required_minor, required_micro)
    def gtk.draw_insertion_cursor(widget, drawable, area, location, is_primary, direction, draw
    def gtk.get_default_language()
    def gtk.events_pending()
    def gtk.main_do_event(event)
    def gtk.main()
    def gtk.main_level()
    def gtk.main_quit()
    def gtk.main_iteration(block)
    def gtk.main_iteration_do(block)
    def gtk.grab_get_current()
    def gtk.quit_add(level, func, ...)
    def gtk.quit_remove(quit_handler_id)
    def gtk.get_current_event()
    def gtk.get_current_event_state()
    def gtk.get_current_event_time()
```

Description

All `PyGTK` functions for the gtk module are listed above categorized by class. Those functions associated with a class have their descriptions included with the class reference. Those functions that are not associated with a specific class are described below.

Functions

gtk.binding_entry_add_signal

```
   def gtk.binding_entry_add_signal(class, keyval, modifiers, signal_name, ...)
```

`object` :	the `gtk.Object` class the binding entry will be associated with
`keyval` :	the key value
`modifiers` :	the modifier mask
`signal_name` :	the signal name
`...` :	zero or more pairs of value type−value pairs

The `gtk.binding_entry_add_signal`() function adds a binding (specified by *`keyval`* and *`modifiers`*) to the binding set of the specified object *`class`*. The signal specified by *`signal_name`* will be emitted with the optional arguments specified by the argument pairs denoted by ... that are value type and value. This function is used when creating a new widget class to set up key bindings.

gtk.check_version

```
    def gtk.check_version(required_major, required_minor, required_micro)
```

required_major :	the required major version number
required_minor :	the required minor version number
required_micro :	the required micro version number
Returns :	None if the underlying GTK+ library is compatible or a string describing the mismatch

The gtk.check_version() function checks the underlying GTK+ library version against the version specified by *required_major*, *required_minor* and *required_micro*. If the library is compatible this function returns None; otherwise it returns a string describing the mismatch.

gtk.draw_insertion_cursor

```
    def gtk.draw_insertion_cursor(widget, drawable, area, location, is_primary, direction, draw_
```

widget :	a gtk.Widget
drawable :	a gtk.gdk.Drawable
area :	the rectangle to which the output is clipped, or None if the output should not be clipped
location :	the location to draw the cursor (*location.width* is ignored)
is_primary :	if TRUE the cursor should be the primary cursor color.
direction :	the direction of the cursor; either gtk.TEXT_DIR_LTR or gtk.TEXT_DIR_RTL
draw_arrow :	if TRUE draw a directional arrow on the cursor. Should be FALSE unless the cursor is split.

Note

This function is available in PyGTK 2.4 and above.

The gtk.draw_insertion_cursor() function draws a text caret on the gtk.gdk.Drawable specified by *drawable* at the position specified by *location*. *area* specifies a clipping rectangle or is None if the output should not be clipped. If *is_primary* is TRUE the cursor should be the primary cursor color. *direction* specifies whether the cursor is right–to–left or left–to–right. This is a convenience function for drawing the standard cursor shape.

gtk.get_default_language

```
    def gtk.get_default_language()
```

Returns :	a pango.Language object for the default language

The gtk.get_default_language() function returns a pango.Language describing the default language.

gtk.events_pending

```
    def gtk.events_pending()
```

Returns :	TRUE if any events are pending

The gtk.events_pending() function returns TRUE if any events are pending. This can be used to update the user interface and invoke timeouts etc. while doing some time intensive computation.

gtk.main_do_event

```
def gtk.main_do_event()
```

event :	a gtk.gdk.Event to process

The `gtk.main_do_event`() function processes a single gtk.gdk.Event. This function is public only to allow filtering of events between GDK and GTK+. You will not usually need to call this function directly. While you should not call this function directly, you might want to know how exactly events are handled. So here is what this function does with the event:

1. Compress enter/leave notify events. If the event passed builds an enter–leave pair together with the next event (peeked from GDK) both events are thrown away. This is to avoid a backlog of (de–)highlighting widgets crossed by the pointer.
2. Find the widget which got the event. If the widget can't be determined the event is thrown away unless it belongs to a INCR transaction. In that case it is passed to `gtk_selection_incr_event`().
3. Then the event is passed on a stack so you can query the currently handled event with `gtk.get_current_event()`.
4. The event is sent to a widget. If a grab is active all events for widgets that are not in the contained in the grab widget are sent to the latter with a few exceptions:

    - Deletion and destruction events are still sent to the event widget for obvious reasons.
    - Events which directly relate to the visual representation of the event widget.
    - Leave events are delivered to the event widget if there was an enter event delivered to it before without the paired leave event.
    - Drag events are not redirected because it is unclear what the semantics of that would be.

    Another point of interest might be that all key events are first passed through the key snooper functions if there are any. Read the description of `gtk_key_snooper_install`() if you need this feature.
5. After finishing the delivery the event is popped from the event stack.

gtk.main

```
def gtk.main()
```

The `gtk.main`() function runs the main loop until the `gtk.main_quit()` function is called. You can nest calls to `gtk.main`(). In that case the call to the `gtk.main_quit()` function will make the innermost invocation of the main loop return.

gtk.main_level

```
def gtk.main_level()
```

Returns :	the nesting level of the current invocation of the main loop

The `gtk.main_level`() function returns the current nesting level of the main loop. The nesting level is increased by calling the `gtk.main()` function and reduced by calling the `gtk.main_quit()` function.

gtk.main_quit

```
def gtk.main_quit()
```

The `gtk.main_quit()` function terminates the current main loop level started by the most recent call to the `gtk.main()` function. The nesting level of the main loop is reduced by calling this function.

gtk.main_iteration

```
def gtk.main_iteration(block=TRUE)
```

`block` :	if `TRUE` block if no events are pending
Returns :	`TRUE` if the `gtk.main_quit()` function has been called for the innermost main loop.

The `gtk.main_iteration()` function runs a single iteration of the mainloop. If no events are waiting to be processed `PyGTK` will block until the next event is noticed if *`block`* is `TRUE`. This function is identical to the `gtk.main_iteration_do()` function.

gtk.main_iteration_do

```
def gtk.main_iteration_do(block=TRUE)
```

`block` :	if `TRUE` block if no events are pending
Returns :	`TRUE` if the `gtk.main_quit()` function has been called for the innermost main loop.

The `gtk.main_iteration_do()` function runs a single iteration of the main loop. If *`block`* is `TRUE` block until an event occurs. This function is identical to the `gtk.main_iteration()` function.

gtk.grab_get_current

```
def gtk.grab_get_current()
```

Returns :	the `gtk.Widget` that has the grab currently or `None` if no grab is active

The `gtk.grab_get_current()` function returns the `gtk.Widget` that has the grab or `None` if no grab is active.

gtk.quit_add

```
def gtk.quit_add(level, func, ...)
```

`level` :	the level at which termination *`func`* shall be called. You can pass 0 here to have *`func`* run at the termination of the current main loop.
`func` :	the function to call – it should return 0 to be removed from the list of quit handlers
`...` :	optional parameter(s) to be passed to *`func`*
Returns :	a handle for this quit handler (you need this for the `gtk.quit_remove()` function).

The `gtk.quit_add()` function registers a function specified by *`func`* to be called when the specified main loop *`level`* is exited. *`func`* should return 0 to be removed from the list of quit handlers. This function returns a handler ID that is used when removing the handler with the `gtk.quit_remove()` function.

gtk.quit_remove

```
def gtk.quit_remove(quit_handler_id)
```

`quit_handler_id` :	the ID of a quit handler

The `gtk.quit_remove()` function removes the quit handler specified by *`quit_handler_id`* from the list of quit handlers.

gtk.get_current_event

```
    def gtk.get_current_event()
```

Returns :	a copy of the current event or `None`

Note

This function is available in PyGTK 2.6 and above.

The `gtk.get_current_event()` function returns a copy of the event currently being processed by GTK+. For example, if you get a "clicked" signal from `gtk.Button`, the current event will be the GdkEventButton that triggered the "clicked" signal. If there is no current event, the function returns `None`.

gtk.get_current_event_state

```
    def gtk.get_current_event_state()
```

Returns :	the state of the current event if any or `None` if there is no current event or state

Note

This function is available in PyGTK 2.6 and above.

The `gtk.get_current_event_state()` function returns the state of the current event or `None` if there is no current event or state.

gtk.get_current_event_time

```
    def gtk.get_current_event_time()
```

Returns :	the timestamp of the current event or 0L

The `gtk.get_current_event_time()` function returns the timestamp from the current event or 0L if there is no current event.

Prev Up Next
gtk.WindowGroup Home Stock Items

Stock Items
Prev **The gtk Class Reference** Next

Stock Items

Stock Items — prebuilt common menu/toolbar items and corresponding icons

Synopsis

```
Functions

    def gtk.stock_add(items)
    def gtk.stock_lookup(stock_id)
    def gtk.stock_list_ids()
```

Description

Stock items represent commonly–used menu or toolbar items such as "Open" or "Exit". Each stock item is identified by a stock ID; stock IDs are just strings, but macros such as `gtk.STOCK_OPEN` are provided to avoid typing mistakes in the strings. Applications can register their own stock items in addition to those built–in to `PyGTK`.

The stock items are:

`gtk.STOCK_ABOUT`	Available in GTK+ 2.6 and above.
`gtk.STOCK_ADD`	
`gtk.STOCK_APPLY`	
`gtk.STOCK_BOLD`	
`gtk.STOCK_CANCEL`	
`gtk.STOCK_CDROM`	
`gtk.STOCK_CLEAR`	
`gtk.STOCK_CLOSE`	
`gtk.STOCK_COLOR_PICKER`	Available in GTK+ 2.2 and above.
`gtk.STOCK_CONVERT`	
`gtk.STOCK_CONNECT`	Available in GTK+ 2.6 and above.
`gtk.STOCK_COPY`	
`gtk.STOCK_CUT`	
`gtk.STOCK_DELETE`	
`gtk.STOCK_DIALOG_AUTHENTICATION`	Available in GTK+ 2.4 and above.
`gtk.STOCK_DIALOG_ERROR`	
`gtk.STOCK_DIALOG_INFO`	
`gtk.STOCK_DIALOG_QUESTION`	
`gtk.STOCK_DIALOG_WARNING`	
`gtk.STOCK_DIRECTORY`	Available in GTK+ 2.6 and above.
`gtk.STOCK_DND`	
`gtk.STOCK_DND_MULTIPLE`	
`gtk.STOCK_EDIT`	Available in GTK+ 2.6 and above.

gtk.STOCK_EXECUTE	
gtk.STOCK_FILE	Available in GTK+ 2.6 and above.
gtk.STOCK_FIND	
gtk.STOCK_FIND_AND_REPLACE	
gtk.STOCK_FLOPPY	
gtk.STOCK_GOTO_BOTTOM	
gtk.STOCK_GOTO_FIRST	
gtk.STOCK_GOTO_LAST	
gtk.STOCK_GOTO_TOP	
gtk.STOCK_GO_BACK	
gtk.STOCK_GO_DOWN	
gtk.STOCK_GO_FORWARD	
gtk.STOCK_GO_UP	
gtk.STOCK_HARDDISK	Available in GTK+ 2.4 and above
gtk.STOCK_HELP	
gtk.STOCK_HOME	
gtk.STOCK_INDENT	Available in GTK+ 2.4 and above.
gtk.STOCK_INDEX	
gtk.STOCK_ITALIC	
gtk.STOCK_JUMP_TO	
gtk.STOCK_JUSTIFY_CENTER	
gtk.STOCK_JUSTIFY_FILL	
gtk.STOCK_JUSTIFY_LEFT	
gtk.STOCK_JUSTIFY_RIGHT	
gtk.STOCK_MEDIA_FORWARD	Available in GTK+ 2.6 and above.
gtk.STOCK_MEDIA_NEXT	Available in GTK+ 2.6 and above.
gtk.STOCK_MEDIA_PAUSE	Available in GTK+ 2.6 and above.
gtk.STOCK_MEDIA_PLAY	RTL version is ▷ Available in GTK+ 2.6 and above.
gtk.STOCK_MEDIA_PREVIOUS	Available in GTK+ 2.6 and above.
gtk.STOCK_MEDIA_RECORD	Available in GTK+ 2.6 and above.
gtk.STOCK_MEDIA_REWIND	Available in GTK+ 2.6 and above.
gtk.STOCK_MEDIA_STOP	Available in GTK+ 2.6 and above.
gtk.STOCK_MISSING_IMAGE	
gtk.STOCK_NETWORK	Available in GTK+ 2.4 and above.
gtk.STOCK_NEW	
gtk.STOCK_NO	

`gtk.STOCK_OK`	
`gtk.STOCK_OPEN`	
`gtk.STOCK_PASTE`	
`gtk.STOCK_PREFERENCES`	
`gtk.STOCK_PRINT`	
`gtk.STOCK_PRINT_PREVIEW`	
`gtk.STOCK_PROPERTIES`	
`gtk.STOCK_QUIT`	
`gtk.STOCK_REDO`	
`gtk.STOCK_REFRESH`	
`gtk.STOCK_REMOVE`	
`gtk.STOCK_REVERT_TO_SAVED`	RTL version is
`gtk.STOCK_SAVE`	
`gtk.STOCK_SAVE_AS`	
`gtk.STOCK_SELECT_COLOR`	
`gtk.STOCK_SELECT_FONT`	
`gtk.STOCK_SORT_ASCENDING`	
`gtk.STOCK_SORT_DESCENDING`	
`gtk.STOCK_SPELL_CHECK`	
`gtk.STOCK_STOP`	
`gtk.STOCK_STRIKETHROUGH`	
`gtk.STOCK_UNDELETE`	RTL version is
`gtk.STOCK_UNDERLINE`	
`gtk.STOCK_UNDO`	
`gtk.STOCK_UNINDENT`	Available in GTK+ 2.4 and above.
`gtk.STOCK_YES`	
`gtk.STOCK_ZOOM_100`	
`gtk.STOCK_ZOOM_FIT`	
`gtk.STOCK_ZOOM_IN`	
`gtk.STOCK_ZOOM_OUT`	

Functions

gtk.stock_add

```
    def gtk.stock_add(items)
```

`items` :	a list or tuple containing 5−tuples of stock items

The `gtk.stock_add()` function registers each of the stock items in the list or tuple specified by *`items`*. The stock items are specified by a 5−tuple containing:

- stock_id − a string identifier
- label − a string to use for a label
- modifier − a modifier mask (see the GDK Modifier Constants section for more detail on modifiers)
- keyval − an integer key value (see `gtk.gdk.Keymap`) Together with the modifiers specifies an accelerator.
- translation_domain − a string identifier of a translation domain

If an item already exists with the same stock ID as one of the items, the old item gets replaced.

gtk.stock_lookup

```
    def gtk.stock_lookup(stock_id)
```

`stock_id` :	a stock item name
Returns :	a 5−tuple containing the stock item info or `None` if *`stock_id`* is unknown

The `gtk.stock_lookup()` function looks up the stock item identified by *`stock_id`* and returns a 5−tuple containing its information. If *`stock_id`* is not known this function returns `None`. See the `gtk.stock_add()` function for more detail.

gtk.stock_list_ids

```
    def gtk.stock_list_ids()
```

Returns :	a list of known stock IDs

The `gtk.stock_list_ids()` function returns a list containing all of the known stock IDs added to a `gtk.IconFactory` or registered with the `gtk.stock_add()` function.

Prev Up Next
gtk Functions Home gtk Constants

The pango Class Reference

Table of Contents

pango Constants

pango Constants   the built–in constants of the pango module

Synopsis

```
Pango Alignment Constants
Pango Attribute Type Constants
Pango Coverage Level Constants
Pango Direction Constants
Pango Ellipsize Mode Constants
Pango Font Mask Flag Constants
Pango Scale Constants
Pango Stretch Constants
Pango Style Constants
Pango Tab Constants
Pango Underline Constants
Pango Variant Constants
Pango Weight Constants
Pango Wrap Mode Constants
```

Description

Pango Alignment Constants

The Alignment constants specify how to align the lines of a `pango.Layout` within the available space. If the `pango.Layout` is set to justify using the `pango.Layout.set_justify()` method, then this only has an effect for partial lines.

`pango.ALIGN_LEFT`	Put all available space on the right
`pango.ALIGN_CENTER`	Center the line within the available space
`pango.ALIGN_RIGHT`	Put all available space on the left

Pango Attribute Type Constants

The Attribute Type constants specify the type of a `pango.Attribute`.Along with the predefined values, it is possible to allocate additional values for custom attributes using the `pango.attr_type_register()` function. The predefined values are given below. The type of structure used to store the attribute is listed in parentheses after the description.

`pango.ATTR_LANGUAGE`	Specifies a `pango.Language`.
`pango.ATTR_FAMILY`	Specifies a font family name list as a string.
`pango.ATTR_STYLE`	Specifies a font slant style. See the `pango.AttrStyle()` function for more details.
`pango.ATTR_WEIGHT`	Specifies a font weight. See the `pango.AttrWeight()` function for more detail.
`pango.ATTR_VARIANT`	Specifies a font variant (normal or small caps). See the `pango.AttrVariant()` function for more detail.
`pango.ATTR_STRETCH`	Specifies a font stretch. See the `pango.AttrStretch()` function for more details.
`pango.ATTR_SIZE`	Specifies a font size in thousandths of a point.
`pango.ATTR_FONT_DESC`	Specifies a `pango.FontDescription`.
`pango.ATTR_FOREGROUND`	Specifies a foreground `pango.Color`.
`pango.ATTR_BACKGROUND`	Specifies a background `pango.Color`.
`pango.ATTR_UNDERLINE`	Specifies an underline style. See the `pango.AttrUnderline()` function for more details.
`pango.ATTR_STRIKETHROUGH`	If `TRUE` the text is struck through.
`pango.ATTR_RISE`	Specifies the displacement of the text from the baseline.
`pango.ATTR_SHAPE`	Specifies a shape. See the `pango.AttrShape()` function for more details.
`pango.ATTR_SCALE`	Specifies a font size scale factor.
`pango.ATTR_FALLBACK`	if `TRUE`, fallback to other fonts is enabled (

Pango Coverage Level Constants

`pango.COVERAGE_NONE`	The character is not representable with the font.
`pango.COVERAGE_FALLBACK`	The character is represented in a way that may be comprehensible but is not the correct graphical form. For instance, a Hangul character represented as a a sequence of Jamos, or a Latin transliteration of a Cyrillic word.
`pango.COVERAGE_APPROXIMATE`	The character is represented as basically the correct graphical form, but with a stylistic variant inappropriate for the current script.
`pango.COVERAGE_EXACT`	The character is represented as the correct graphical form.

Pango Direction Constants

The pango Direction constants specify a direction in the Unicode bidirectional algorithm. Not every value in this enumeration makes sense for every usage of pango Direction.

The `pango.DIRECTION_TTB_LTR`, `pango.DIRECTION_TTB_RTL` values come from an earlier interpretation of this enumeration as the writing direction of a block of text and are no longer used; See the Text module of the CSS3 spec for how vertical text is planned to be handled in a future version of Pango. The explanation of why `pango.DIRECTION_TTB_LTR` is treated as `pango.DIRECTION_RTL` can be found there as well.

`pango.DIRECTION_LTR`	A strong left–to–right direction
`pango.DIRECTION_RTL`	A strong right–to–left direction
`pango.DIRECTION_TTB_LTR`	Deprecated value; treated the same as `pango.DIRECTION_RTL`.
`pango.DIRECTION_TTB_RTL`	Deprecated value; treated the same as `pango.DIRECTION_LTR`
`pango.DIRECTION_WEAK_LTR`	A weak left–to–right direction
`pango.DIRECTION_WEAK_RTL`	A weak right–to–left direction
`pango.DIRECTION_NEUTRAL`	No direction specified

Pango Ellipsize Mode Constants

The Ellipsize Mode constants specify what sort of (if any) ellipsization should be applied to a line of text. In the ellipsization process characters are removed from the text in order to make it fit to a given width and replaced with an ellipsis (...).

`pango.ELLIPSIZE_NONE`	No ellipsization.
`pango.ELLIPSIZE_START`	Omit characters at the start of the text.
`pango.ELLIPSIZE_MIDDLE`	Omit characters in the middle of the text.
`pango.ELLIPSIZE_END`	Omit characters at the end of the text.

Pango Font Mask Flag Constants

The Font Mask flag constants are bit–flags that correspond to fields in a `pango.FontDescription` that have been set.

`pango.FONT_MASK_FAMILY`	the font family is specified.
`pango.FONT_MASK_STYLE`	the font style is specified.
`pango.FONT_MASK_VARIANT`	the font variant is specified.
`pango.FONT_MASK_WEIGHT`	the font weight is specified.
`pango.FONT_MASK_STRETCH`	the font stretch is specified.
`pango.FONT_MASK_SIZE`	the font size is specified.

Pango Scale Constants

`pango.SCALE`	The `pango.SCALE` constant represents the scale between dimensions used for Pango distances and device units. (The definition of device units is dependent on the output device; it will typically be pixels for a screen, and points for a printer.) `pango.SCALE` is currently 1024, but this may be changed in the future. When setting font sizes, device units are always considered to be points (as in "12 point font"), rather than pixels.
`pango.SCALE_XX_SMALL`	The scale factor for three shrinking steps (1 / (1.2 * 1.2 * 1.2)).
`pango.SCALE_X_SMALL`	he scale factor for two shrinking steps (1 / (1.2 * 1.2)).
`pango.SCALE_SMALL`	The scale factor for one shrinking step (1 / 1.2).
`pango.SCALE_MEDIUM`	The scale factor for normal size (1.0).
`pango.SCALE_LARGE`	The scale factor for one magnification step (1.2).
`pango.SCALE_X_LARGE`	he scale factor for two magnification steps (1.2 * 1.2).
`pango.SCALE_XX_LARGE`	The scale factor for three magnification steps (1.2 * 1.2 * 1.2).

Pango Stretch Constants

The Stretch constants specify the width of the font relative to other designs within a family.

`pango.STRETCH_ULTRA_CONDENSED`	The most narrow width
`pango.STRETCH_EXTRA_CONDENSED`	
`pango.STRETCH_CONDENSED`	
`pango.STRETCH_SEMI_CONDENSED`	
`pango.STRETCH_NORMAL`	The normal width.
`pango.STRETCH_SEMI_EXPANDED`	
`pango.STRETCH_EXPANDED`	
`pango.STRETCH_EXTRA_EXPANDED`	
`pango.STRETCH_ULTRA_EXPANDED`	The most expanded width

Pango Style Constants

The Style constants specify the various slant styles possible for a font.

`pango.STYLE_NORMAL`	The font is upright.
`pango.STYLE_OBLIQUE`	The font is slanted in a roman style.
`pango.STYLE_ITALIC`	The font is slanted in an italic style.

Pango Tab Constants

The Tab constants specify where a tab stop appears relative to the text.

`pango.TAB_LEFT`	the tab stop appears to the left of the text.

Pango Underline Constants

The Underline constants specify he type of underlining (if any) to be used.

`pango.UNDERLINE_NONE`	No underline should be drawn.
`pango.UNDERLINE_SINGLE`	A single underline should be drawn.
`pango.UNDERLINE_DOUBLE`	A double underline should be drawn.
`pango.UNDERLINE_LOW`	A single underline should be drawn at a position beneath the ink extents of the text being underlined. This should be used only for underlining single characters, such as for keyboard accelerators. `pango.UNDERLINE_SINGLE` should be used for extended portions of text.

Pango Variant Constants

The Variant constants specify the capitalization variant of the font.

`pango.VARIANT_NORMAL`	A normal font.
`pango.VARIANT_SMALL_CAPS`	A font with the lower case characters replaced by smaller variants of the capital characters.

Pango Weight Constants

The Weight constants specify the weight (boldness) of a font. This is a numerical value ranging from 100 to 900, but there are some predefined values:

`pango.WEIGHT_ULTRALIGHT`	The ultralight weight (= 200).
`pango.WEIGHT_LIGHT`	The light weight (=300).
`pango.WEIGHT_NORMAL`	The default weight (= 400).
`pango.WEIGHT_BOLD`	The bold weight (= 700).
`pango.WEIGHT_ULTRABOLD`	The ultrabold weight (= 800).
`pango.WEIGHT_HEAVY`	The heavy weight (= 900).

Pango Wrap Mode Constants

The Wrap Mode constants specify how to wrap the lines of a `pango.Layout` to the desired width.

`pango.WRAP_WORD`	wrap lines at word boundaries.
`pango.WRAP_CHAR`	wrap lines at character boundaries.
`pango.WRAP_WORD_CHAR`	wrap lines at word boundaries, but fall back to character boundaries if there is not enough space for a full word.

Prev Up Next
pango Functions Home Appendix A. ChangeLog

pango Functions
Prev **The pango Class Reference** Next

pango Functions

pango Functions — a list of all the pango functions

Synopsis

`pango.Attribute` Functions

```
  def pango.attr_type_register(name)
   def pango.AttrLanguage(language, start_index=0, end_index=1)
   def pango.AttrFamily(family, start_index=0, end_index=1)
   def pango.AttrForeground(red, green, blue, start_index=0, end_index=1)
   def pango.AttrBackground(red, green, blue, start_index=0, end_index=1)
   def pango.AttrSize(size, start_index=0, end_index=1)
   def pango.AttrStyle(style, start_index=0, end_index=1)
   def pango.AttrWeight(weight, start_index=0, end_index=1)
   def pango.AttrVariant(variant, start_index=0, end_index=1)
   def pango.AttrStretch(stretch, start_index=0, end_index=1)
   def pango.AttrFontDesc(desc, start_index=0, end_index=1)
   def pango.AttrUnderline(underline, start_index=0, end_index=1)
   def pango.AttrStrikethrough(strikethrough, start_index=0, end_index=1)
   def pango.AttrRise(rise, start_index=0, end_index=1)
   def pango.AttrShape(ink_rect, logical_rect, start_index=0, end_index=1)
   def pango.AttrScale(scale, start_index=0, end_index=1)
   def pango.AttrFallback(fallback, start_index=0, end_index=1)
```

pango.AttrList Functions

```
    def pango.parse_markup(markup_text, accel_marker)
```

pango.Font Functions

```
    def pango.PIXELS(size)
     def pango.ASCENT(rect)
     def pango.DESCENT(rect)
     def pango.RBEARING(rect)
     def pango.LBEARING(rect)
```

The Pango Markup Language

The Pango Markup Language a simple markup language for encoding attributes with text.

Description

The pango markup language is a very simple SGML–like language that allows you specify attributes with the text they are applied to by using a small set of markup tags. A simple example of a string using markup is:

```
<span foreground="blue" size="100">Blue text</span> is <i>cool</i>!
```

Using the pango markup language to markup text and parsing the result with the `pango.parse_markup()` function is a convenient way to generate the `pango.AttrList` and plain text that can be used in a `pango.Layout`.

The root tag of a marked–up document is `<markup>`, but the `pango.parse_markup()` function allows you to omit this tag, so you will most likely never need to use it. The most general markup tag is `<span>`. The `<span>` tag has the following attributes:

`font_desc`	A font description string, such as "Sans Italic 12"; note that any other span attributes will override this description. So if you have "Sans Italic" and also a style="normal" attribute, you will get Sans normal, not italic.
`font_family`	A font family name such as "normal", "sans", "serif" or "monospace".
`face`	A synonym for font_family
`size`	The font size in thousandths of a point, or one of the absolute sizes 'xx–small', 'x–small', 'small', 'medium', 'large', 'x–large', 'xx–large', or one of the relative sizes 'smaller' or 'larger'.
`style`	The slant style – one of 'normal', 'oblique', or 'italic'
`weight`	The font weight – one of 'ultralight', 'light', 'normal', 'bold', 'ultrabold', 'heavy', or a numeric weight.
`variant`	The font variant – either 'normal' or 'smallcaps'.
`stretch`	The font width – one of 'ultracondensed', 'extracondensed', 'condensed', 'semicondensed', 'normal', 'semiexpanded', 'expanded', 'extraexpanded', 'ultraexpanded'.

`foreground`	An RGB color specification such as '#00FF00' or a color name such as 'red'.
`background`	An RGB color specification such as '#00FF00' or a color name such as 'red'.
`underline`	The underline style – one of 'single', 'double', 'low', or 'none'.
`rise`	The vertical displacement from the baseline, in ten thousandths of an em. Can be negative for subscript, positive for superscript.
`strikethrough`	'true' or 'false' whether to strike through the text.
`fallback`	If `TRUE` enable fallback to other fonts of characters are missing from the current font. If disabled, then characters will only be used from the closest matching font on the system. No fallback will be done to other fonts on the system that might contain the characters in the text. Fallback is enabled by default. Most applications should not disable fallback.
`lang`	A language code, indicating the text language.

There are a number of convenience tags that encapsulate specific span options:

`b`	Make the text bold.
`big`	Makes font relatively larger, equivalent to <span size="larger">.
`i`	Make the text italic.
`s`	Strikethrough the text.
`sub`	Subscript the text.
`sup`	Superscript the text.
`small`	Makes font relatively smaller, equivalent to <span size="smaller">.
`tt`	Use a monospace font.
`u`	Underline the text.

Prev Up Next
pango.TabArray Home pango Functions

Introduction
Prev Next

Introduction

Table of Contents

This document describes most of the `PyGTK` version 2.0 through 2.6 classes and their methods and associated functions. Deprecated classes, functions and methods have been specifically left out of this reference though classes that have become deprecated since PyGTK 2.0 have been left in but annotated with a deprecation warning. I have attempted to document as much of the `PyGTK API` as I can but there are undoubtedly errors and omissions. If you discover any of these please send me email at <`finlay@moeraki.com`> or file a bug report at bugzilla.gnome.org for the `pygtk` project. Specific areas that have not been documented include:

- The ATK classes

This reference describes the API for `PyGTK` as of version 2.5.0+ and assumes that the additional API changes for version 2.6 may be significant. There will undoubtedly be changes that are not reflected in this reference. The differences in the API between version 2.0 and previous versions are denoted in this reference with a

Note that describes the availability of the object, constructor, method or function. Any of these that do not have a notation can be assumed to be available in all versions of PyGTK from 2.0 and up. In the case of properties and signals the availability is dependent on the version of the underlying GTK+, GDK or Pango libraries. These will be annotated appropriately in a similar fashion. The source code must be consulted if this reference and your version of `PyGTK` seem to differ. You are encouraged to use the latest version of `PyGTK` that is available. See the `PyGTK` homepage for more information and more resources on how to use PyGTK as well as help in its development.

The Reference contains a chapter for each `PyGTK` module (that corresponds to the underlying `GTK+` library) containing the class descriptions. The second chapter illustrates the `PyGTK` class hierarchy covering the gobject, gtk, gtk.gdk and pango modules.

The class descriptions are arranged alphabetically within the chapters. Currently there are five module chapters:

The `gobject` module	The classes that are included in the `gobject` module of `PyGTK` and are accessed similar to: gobject.GObject. These classes are the base object classes that the `gtk` and `gtk.gdk` module classes are built on.
The `gtk` module	The classes that are included in the `gtk` module of `PyGTK` and are accessed similar to: gtk.Widget. These classes are the "higher" level widget classes that provide most of the user interface widgets used for application development.
The `gtk.gdk` module	The classes that are included in the `gtk.gdk` module of `PyGTK`. These classes are "lower" level classes that provide more fundamental capabilities that the `gtk` module widgets are built upon. These classes provide an abstract interface to the underlying window system (either X Window System or Microsoft Windows).
The `gtk.glade` module	The classes that are included in the `gtk.glade` module of `PyGTK`. These classes provide access to the libglade functions that allow the dynamic loading of user interfaces from XML descriptions.
The `pango` module	The classes that are included in the `pango` module of `PyGTK`. These classes provide access to the Pango text layout and rendering engines. PyGTK supports a subset of the full Pango capability: mainly the high level layout capabilities exposed by the pango.Layout objects. The low level rendering capabilities have not been exposed mostly because there isn't a full GObject interface to the underlying Pango data structures. It's also likely that the rendering capabilities require more performance that Python can provide.

The `atk` module classes will be added in a later version of this Reference.

Major Changes since Version 1.9

The major changes in this document since version 1.9 include:

- descriptions of new GTK+ 2.2 and 2.4 classes:
    - `gobject.GBoxed`
    - `gobject.GPointer`
    - `gobject.GInterface`
    - `gobject.MainContext`
    - `gobject.MainLoop`
    - `gtk.Action`
    - `gtk.ActionGroup`
    - `gtk.Border`
    - `gtk.Clipboard`
    - `gtk.ColorButton`
    - `gtk.ComboBox`

    - `gtk.ComboBoxEntry`
    - `gtk.EntryCompletion`
    - `gtk.Expander`
    - `gtk.FileChooser`
    - `gtk.FileChooserDialog`
    - `gtk.FileChooserWidget`
    - `gtk.FileFilter`
    - `gtk.FontButton`
    - `gtk.IconInfo`
    - `gtk.IconTheme`
    - `gtk.RadioAction`
    - `gtk.RadioToolButton`
    - `gtk.SeparatorToolItem`
    - `gtk.ToggleAction`
    - `gtk.ToggleToolButton`
    - `gtk.ToolButton`
    - `gtk.ToolItem`
    - `gtk.TreeModelFilter`
    - `gtk.TreeRowReference`
    - `gtk.UIManager`
    - `gtk.gdk.Atom`
    - `gtk.gdk.Display`
    - `gtk.gdk.DisplayManager`
    - `gtk.gdk.Screen`
    - `pango.FontsetSimple`
- Descriptions of new GTK+ 2.6 classes:
    - `gtk.AboutDialog`
    - `gtk.CellRendererCombo`
    - `gtk.CellRendererProgress`
    - `gtk.CellView`
    - `gtk.FileChooserButton`
    - `gtk.IconView`
    - `gtk.MenuToolButton`
- a reference page in each module chapter containing descriptions of or links to all of the functions defined in a module.
- a reference page in each module chapter containing descriptions of all of the constants defined in a module.
- additional methods, functions, properties and signals added for GTK+ 2.2, GTK+ 2.4, GTK+ 2.6, PyGTK 2.2, PyGTK 2.4 and PyGTK 2.6.

Prev Up Next
PyGTK 2.0 Reference Manual Home Reference Page Format

Appendix A. ChangeLog
Prev

Appendix A. ChangeLog

======== 2.5.2 ============
2005−03−05 John Finlay <finlay@moeraki.com>

* `pygtk2−ref.xml` Bump version number and change date.

* `pygtk-introduction` (*Major Changes*) Update description with new
GTK+ 2.6 classes and add gtk.glade module.

2005−02−28 John Finlay <finlay@moeraki.com>

* `pygtk-gdkcursor.xml` (*gtk.gdk.Cursor*): Rephrase constructor
descriptions to clarify how to make an invisible cursor. Fixes #168755
(*get_display*) Fix mangled description.

2005−02−15 John Finlay <finlay@moeraki.com>

* `pygtk-gtkcellrenderertext.xml` (*Properties*): Add GTK+ 2.6
properties ellipsize, ellipsize−set, width−chars. Alphabetize entries.
Fixes #167504 (*Rafael Villar Burke*)

2005−02−03 John Finlay <finlay@moeraki.com>

* `pygtk-gtkcellrenderer.xml` (*stop_editing, "editing−started"*):
Add descriptions of these PyGTK 2.6 additions.

2005−01−18 John Finlay <finlay@moeraki.com>

* `pygtk2-ref-xml:` Add pygtk−glade−classes.xml. Change date.

* pygtk−glade−classes.xml
* `pygtk-gladexml.xml:` Add these files created by Johan Dahlin.

* `pygtk-gladexml-classes.xml:` Remove.

2005−01−15 John Finlay <finlay@moeraki.com>

* `pygtk-gtkentrycompletion.xml` (*get_text_column*): Remove unimplemented
method.
(*insert−prefix*): Add description of PyGTK2.6 method.

2005−01−13 John Finlay <finlay@moeraki.com>

* `pygtk-gtklabel.xml` (*get_angle, set_angle, set_max_width_chars*)
(*get_max_width_chars*): Add descriptions of these PyGTK 2.6 methods.
(*"angle", "max−width−chars"*): Add descriptions.

* `pygtk-gtk-functions.xml` (*gtk.get_current_event*)
(*gtk.get_current_event_state*): Add descriptions of these PyGTK2.6
functions.

* pygtk−gdkpixmap.xml
* `pygtk-gdk-functions.xml` (*gtk.pixmap_colormap_create_from_xpm*)
(*gtk.pixmap_colormap_create_from_xpm_d*)
(*gtk.pixmap_colormap_from_xpm_d*):
First arg can be a GdkDrawable.

* `pygtk-gtkcontainer.xml` (*gtk.container_class_install_child_property*):
Add description of this PyGTK2.4 function. Update Description.

* `pygtk-gtk-functions.xml` (*gtk.container_class_install_child_property*): Add link to.

* `pygtk-gtkwidget.xml` (*"child−notify", "client−event"*) (*"proximity−in−event", "proximity−out−event"*) Add descriptions. Patch by Gian Mario Tagliaretti.

2004−12−27 John Finlay <finlay@moeraki.com>

* `pygtk-gdk-constants.xml` (*event type constants*): Fix Typo (Gian Mario Tagliaretti)

2004−12−26 John Finlay <finlay@moeraki.com>

* `pygtk-gdkevent.xml` (*Attributes*): CLIENT_EVENT message_type is writeable.

2004−12−24 John Finlay <finlay@moeraki.com>

* `pygtk2-ref.xml`: Bump version number and pubdate.

* `pygtk-gtkuimanager.xml` (*"post−activate", "pre−activate"*): Add action param description.

======== 2.5.1 ============

2004−12−23 John Finlay <finlay@moeraki.com>

* `pygtk2-ref.xml` Update pubdate and revhistory.

* `pygtk-gdkpixbuf.xml` (*render_to_drawable, render_to_drawable_alpha*): Add deprecation warning.

* `pygtk-gtkcellview.xml` (*set_value, set_values, set_cell_data_func*): Remove descriptions of removed methods.

2004−12−21 John Finlay <finlay@moeraki.com>

* `Various` Use xref tags instead of link tags where possible.

* `pygtk2-ref.xml` Add revhistory, releaseinfo and edition tags. Remove version number from title tag and add to releaseinfo tag.

* `pygtk-gtktoolbar.xml` (*set_style*) Add a note about ToolItem label display when style is TOOLBAR_BOTH_HORIZ. (*Gian Mario Tagliaretti*)

2004−12−10 John Finlay <finlay@moeraki.com>

* `pygtk-gtksettings.xml` (*Description*): Change description to indicate that there is one Settings object per GdkScreen (*Rafael Villar Burke*)

2004−12−09 John Finlay <finlay@moeraki.com>

* `pygtk-gtk-stock-items.xml` (*gtk.stock_add*): Add reference to

Modifier Constants docs.

2004−12−07 John Finlay <finlay@moeraki.com>

* `pygtk-gtkwindow.xml` (*move*): Fix example code.
Fixes #160678. (*Gian Mario Tagliaretti*)

* `pygtk-gtkwindow.xml` (*set_type_hint*): Add reference to window type hint constants. Fixes #160669 (*Gian Mario Tagliaretti*)
(*set_gravity, begin_resize_drag*): Add reference to constants.

2004−12−06 John Finlay <finlay@moeraki.com>

* `pygtk-gtk-stock-items.xml` Remove duplicate STOCK_WARNING.
Fixes #160584 (*Erik Grinaker*).
Add GTK+ 2.6 stock icons.

2004−12−04 John Finlay <finlay@moeraki.com>

* `pygtk-gtkdialog.xml` (*set_alternative_button_order*): Add.

* `pygtk-pangolayoutiter.xml` Add.

* `pygtk-pangolayout.xml` (*get_iter*) Add description of this PyGTK 2.6 method.

* `pygtk-gtkcellview.xml` (*set_cell_data*) Remove.

* `pygtk-gtkfilechooserbutton.xml` (*set_active, get_active, "active"*): Remove.

2004−11−20 John Finlay <finlay@moeraki.com>

* `pygtk_gtkdialog.xml` (*gtk.Dialog*) Fix param list bug.

2004−11−18 John Finlay <finlay@moeraki.com>

* `pygtk-gtkliststore.xml` (*remove*): Document return value differences.
(*Doug Quale*)

2004−11−16 John Finlay <finlay@moeraki.com>

* `pygtk2-ref.xml` Set pubdate. Bump version number to 2.5.1

* `pygtk-gtkselectiondata.xml` (*gtk.target_list_add_image_targets*)
(*gtk.target_list_add_text_targets*)
(*gtk.target_list_add_uri_targets*): Add descriptions.

* `pygtk-gtkaboutdialog.xml` (*gtk.about_dialog_set_email_hook*)
(*gtk.about_dialog_set_url_hook*): Add descriptions.

* `pygtk-gtk-functions.xml:` Add new AboutDialog, CellView, Image and Window functions. Fix SelectionData and Settings functions.

======== 2.5.0 ============

2004−11−15 John Finlay <finlay@moeraki.com>

* `pygtk-introduction.xml` Update.

* `pygtk2-ref.xml` Set pubdate. Bump version number to 2.5.0

* `pygtk-gdkwindow.xml` (*set_focus_on_map*)
(*enable_synchronized_configure, configure_finished*): Add descriptions

* `pygtk-gdkdragcontext.xml` (*drag_drop_succeeded*): Add description.

* `pygtk-gdkdisplay.xml` (*store_clipboard*)
(request_selection_notification, supports_clipboard_persistence
(*supports_selection_notification*): Add descriptions.

* `pygtk-gtktreeselection.xml` (*get_selected_rows*): Remove model
arg − not needed. Fixes #158397 (*Ken Harris*)

* `pygtk-gtkwindow.xml` (*gtk.window_set_default_icon_name*)
(*get_icon_name, set_icon_name, get_focus_on_map, set_focus_on_map*):
Add descriptions.

* `pygtk-gtktreeview.xml` (*set_row_separator_func*)
(*set_hover_expand, get_hover_expand, set_hover_selection*)
(*get_hover_selection, set_fixed_height_mode, get_fixed_height_mode*)
(*"hover−selection", "hover−expand", "fixed−height−mode"*):
Add descriptions.

* `pygtk-gtktoolitem.xml` (*rebuild_menu*): Add description
(*"create−menu−proxy"*): Update description.

* `pygtk-gtktextbuffer.xml` (*backspace, "tag−table"*): Add descriptions.

2004−11−14 John Finlay <finlay@moeraki.com>

* `pygtk-gtkselectiondata.xml` (*data_targets_include_image*)
(*get_uris, set_uris, get_pixbuf, set_pixbuf*): Add descriptions.

* `pygtk-gtkmessagedialog.xml` (*format_secondary_text*)
(*format_secondary_markup*): Add descriptions.

* `pygtk-gtkmenutoolbutton.xml` Add.

* `pygtk-gtkwidget.xml` (*menu_get_for_attach_widget*): Add description.

* `pygtk-gtkmenu.xml` (*"tearoff−state"*) Add description.

* `pygtk-gtklabel.xml` (*Description*): Update.
(*set_ellipsize, get_ellipsize, set_width_chars, get_width_chars*)
(*get_single_line_mode, set_single_line_mode*): Add descriptions.
(*"ellipsize, "width−chars", "single−line−mode"*): Add descriptions.

* `pygtk-pango-constant.xml` Add Ellipsize Mode Constants.

* `pygtk-gtkimage.xml` (*gtk.image_new_from_icon_name*)
(*get_icon_name, set_from_icon_name, set_pixel_size, get_pixel_size*):
Add descriptions of these PyGTK 2.6 methods and function.
(*Description*): Update.

2004−11−13 John Finlay <finlay@moeraki.com>

* `pygtk-gtkiconview.xml` Add.
* `pygtk-gtk-classes.xml` Add pygtk−pygtk−gtkiconview.xml

* `pygtk-gtkicontheme.xml` (*get_icon_sizes*): Add description.

* `pygtk-gtkfilechooser.xml` (*set_show_hidden, get_show_hidden*):
Add description os these PyGTK 2.6 methods.
(*Description*) Update.

* `pygtk-gtkfilechooserbutton.xml` Add.

* `pygtk-gtk-classes.xml` Add pygtk−gtkfilechooserbutton.xml

2004−11−12 John Finlay <finlay@moeraki.com>

* `pygtk-gobject-functions.xml` (*signal_list_ids, signal_lookup*)
(*signal_name, signal_query*): Add descriptions of these PyGTK 2.6
methods.

* `pygtk-gtkwidget.xml` (*set_accelerator*): Change wording to clarify
argument value usage. (*stan@saticed.me.uk*)

2004−11−10 John Finlay <finlay@moeraki.com>

* `pygtk-gtkentrycompletion.xml` (*get_text_column*)
(*set_inline_completion, get_inline_completion*)
(*set_popup_completion, get_popup_completion*): Add descriptions of
these PyGTK 2.6 methods.

* `pygtk-gtkwidget.xml` (*drag_dest_add_image_targets*)
(*drag_dest_add_text_targets, drag_dest_add_uri_targets*)
(*drag_source_add_text_targets, drag_source_get_target_list*)
(*drag_source_set_target_list*): Add descriptions of these PyGTK 2.6
methods.

* `pygtk-gtkcombobox.xml` (*get_wrap_width, get_row_span_column*)
(*get_column_span_column, get_active_text, get_popup_accessible*)
(*get_row_separator_func, set_row_separator_func, get_add_tearoffs*)
(*set_add_tearoffs, get_focus_on_click, set_focus_on_click*):
Add descriptions of these PyGTK 2.6 methods.

* `pygtk-gtkclipboard.xml` (*wait_is_target_available*)
(*set_can_store, store*): Add documentation on these PyGTK 2.6 methods

* `pygtk-gtkcellrenderercombo.xml` Add.
* `pygtk-gtkcellrendererprogress.xml` Add.

* `pygtk-gtkcellview.xml` Add.
* `pygtk-gtk-classes.xml` Add above files.

2004−11−09 John Finlay <finlay@moeraki.com>

* `pygtk-gtkaboutdialog.xml` Add.

* `pygtk-gtk-classes.xml` Add pygtk−gtkaboutdialog.xml

2004−11−08 John Finlay <finlay@moeraki.com>

* `pygtk-gtkwindow.xml` (*Description, gtk.Window, set_position*): Add link to appropriate Constants description.

2004−11−01 John Finlay <finlay@moeraki.com>

* `pygtk-gtksettings.xml` Add gtk−alternative−button−order and gtk−modules properties. Fix typo.

2004−10−28 John Finlay <finlay@moeraki.com>

* `pygtk-gtkcombobox.xml` (*set_model*) model can be and default to None in PyGTK 2.4.1.

* `pygtk-gtkentrycompletion.xml` (*set_model*) model can be and default to None in PyGTK 2.4.1.

2004−10−21 John Finlay <finlay@moeraki.com>

* `pygtk-gtkalignment.xml` (*gtk.Alignment*) Undo changes since new default values are wrong.

2004−10−19 John Finlay <finlay@moeraki.com>

* `pygtk-gtkalignment.xml` (*gtk.Alignment*) Note change of default values in 2.4. (*Gustavo Niemeyer*)

2004−10−12 John Finlay <finlay@moeraki.com>

* `pygtk-gtkwidget.xml` (*"destroy−event"*) Add description per Christian Reis.

* `pygtk-gtkstatusbar.xml` (*push*) Note that the message id can be used with the remove() method. (*Rafael Villar Burke*)

* `pygtk-gtkwidget.xml` (*modify_bg*) Note that bg can only be modified on widgets with a gdkWindow. (*Rafael Villar Burke*)

* `pygtk-gtkadjustment.xml` (*Description*) Fix broken links reported by Antoon Pardon.

2004−10−05 John Finlay <finlay@moeraki.com>

* `pygtk2-ref.xml` Bump version number and set pubdate

* `pygtk-gdkgc.xml` (*set_clip_rectangle*) Note that the clip origin is also set to (*0, 0*) per Rafael Villar Burke.

================ 2.4.11 ==============

2004−10−03 John Finlay <finlay@moeraki.com>

* `pygtk2-ref.xml` Set pubdate.

* `pygtk-gtkuimanager.xml` (*insert_action_group*) Add details on pos param.
(*add_ui*) Additional info on path param. Add info on type values.
(*new_merge_id*) Add info on merge ids.

2004−09−28 John Finlay <finlay@moeraki.com>

* `pygtk-gtkalignment.xml` (*Description*) Fix example description.

2004−09−18 John Finlay <finlay@moeraki.com>

* `pygtk-gtkactiongroup.xml` (*add_actions*) (*add_toggle_actions*) (*add_radio_actions*) Clarify entry tuple field usage.

2004−09−15 John Finlay <finlay@moeraki.com>

* `pygtk-gtktreeview.xml` (*set_search_equal_func*) Note that the comparison function should return FALSE to indicate a match. (Thomas Mills Hinkle)

2004−09−06 John Finlay <finlay@moeraki.com>

* `pygtk-gtkaction.xml` (*Properties*) name property is construct only.

2004−08−11 John Finlay <finlay@moeraki.com>

* `pygtk2-ref.html` Bump version number.

================ 2.4.10 ==============

2004−08−11 John Finlay <finlay@moeraki.com>

* `pygtk2-ref.xml` Set pubdate.

* `pygtk-gdkdisplay.xml` pygtk−gdkdisplaymanager.xml
* `pygtk-gdkkeymap.xml` pygtk−gdkpixbufloader.xml
* `pygtk-gdkscreen.xml` pygtk−gtkaccelgroup.xml
* `pygtk-gtkaction.xml` pygtk−gtkactiongroup.xml
* `pygtk-gtkadjustment.xml` pygtk−gtkbutton.xml
* `pygtk-gtkcalendar.xml` pygtk−gtkcelleditable.xml
* `pygtk-gtkcellrenderer.xml` pygtk−gtkcellrenderertext.xml
* `pygtk-gtkcellrenderertoggle.xml` pygtk−gtkcheckmenuitem.xml
* `pygtk-gtkcolorbutton.xml` pygtk−gtkcolorselection.xml
* `pygtk-gtkcombobox.xml` pygtk−gtkcontainer.xml
* `pygtk-gtkcurve.xml` pygtk−gtkdialog.xml pygtk−gtkeditable.xml

* `pygtk-gtkentry.xml` pygtk–gtkentrycompletion.xml
* `pygtk-gtkexpander.xml` pygtk–gtkfilechooser.xml
* `pygtk-gtkfontbutton.xml` pygtk–gtkhandlebox.xml
* `pygtk-gtkicontheme.xml` pygtk–gtkimcontext.xml
* `pygtk-gtkinputdialog.xml` pygtk–gtkitem.xml pygtk–gtklabel.xml
* `pygtk-gtklayout.xml` pygtk–gtkmenu.xml pygtk–gtkmenuitem.xml
* `pygtk-gtkmenushell.xml` pygtk–gtknotebook.xml
* `pygtk-gtkobject.xml` pygtk–gtkoptionmenu.xml pygtk–gtkpaned.xml
* `pygtk-gtkplug.xml` pygtk–gtkradioaction.xml
* `pygtk-gtkradiobutton.xml` pygtk–gtkradiomenuitem.xml
* `pygtk-gtkrange.xml` pygtk–gtkscale.xml
* `pygtk-gtkscrolledwindow.xml` pygtk–gtksocket.xml
* `pygtk-gtkspinbutton.xml` pygtk–gtkstatusbar.xml
* `pygtk-gtktextbuffer.xml` pygtk–gtktexttag.xml
* `pygtk-gtktexttagtable.xml` pygtk–gtktextview.xml
* `pygtk-gtktoggleaction.xml` pygtk–gtktogglebutton.xml
* `pygtk-gtktoggletoolbutton.xml` pygtk–gtktoolbar.xml
* `pygtk-gtktoolbutton.xml` pygtk–gtktoolitem.xml
* `pygtk-gtktreemodel.xml` pygtk–gtktreeselection.xml
* `pygtk-gtktreesortable.xml` pygtk–gtktreeview.xml
* `pygtk-gtktreeviewcolumn.xml` pygtk–gtkuimanager.xml
* `pygtk-gtkviewport.xml` pygtk–gtkwidget.xml pygtk–gtkwindow.xml
Fix signal titles.

2004–08–10 John Finlay <finlay@moeraki.com>

* `pygtk2-ref.xml` Undo erroneous check–in

2004–08–06 John Finlay <finlay@moeraki.com>

* `pygtk2-ref.xml` Bump version number and pubdate

* `pygtk-gtkclipboard.xml` (*Synopsis*) Fix typo. (*Able Daniel*)

=============== 2.4.9 ==============
2004–08–03 John Finlay <finlay@moeraki.com>

* `pygtk2-ref.xml` Update pubdate.

2004–08–02 John Finlay <finlay@moeraki.com>

* `pygtk-gtkexpander.xml` (*gtk.expander_new_with_mnemonic*)
(*gtk.Expander*) Note that label is optional and defaults to None.
(*set_label*) (*set_label_widget*) Note that label and label_widget
can be None.

* `pygtk-gtkcomboboxentry.xml` (*gtk.ComboBoxEntry*) Default value
for column is –1.
(*Description*) (*set_text_column*)
Note that the text column can only be set once.

2004–07–31 John Finlay <finlay@moeraki.com>

* `pygtk-gtkmenu.xml` (*Properties*) Swap Child and Style property titles.

2004−07−29 John Finlay <finlay@moeraki.com>

* `pygtk−gtktreeview.xml` (*get_drag_dest_row*) None is a valid return value.

* `pygtk−gtkliststore.xml` (*insert_before*) (*insert_after*) sibling param can be None in PyGTK 2.4.

2004−07−28 John Finlay <finlay@moeraki.com>

* `pygtk−gtkicontheme.xml` (*Description*) Fix broken ulink tag.

* `pygtk−gtkexpander.xml` (*Description*) Fix bug in example code using "expanded" property signal. Add note about using "activate" signal.

2004−07−27 John Finlay <finlay@moeraki.com>

* `pygtk−gdkatom.xml` (*Description*) Add note on support for Atom and string comparsion support in PyGTK 2.4.

* `pygtk−gdkwindow.xml` (*property_get*) (*property_change*) Add links for gtk.gdk.Atom references. Fixes #148569. Thanks to Abel Daniel.

2004−07−24 John Finlay <finlay@moeraki.com>

* `pygtk−gdk−constants.xml` (*Filter Return Constants*) Add reference to gtk.gdk.Window.Add_filter() method.

* `pygtk−gdkwindow.xml` (*add_filter*) Add initializer for data. Describe return value for callback.

2004−07−23 John Finlay <finlay@moeraki.com>

* `pygtk−gtktreemodel.xml` (*"row−inserted"*) (*"row−changed"*) Add detail on when these are emitted.

2004−07−22 John Finlay <finlay@moeraki.com>

* `pygtk−gtktextbuffer.xml` (*add_selection_clipboard*)
(*remove_selection_clipboard*)
(*cut_clipboard*) (*copy_clipboard*) (*paste_clipboard*)
These methods are available in PyGTK 2.2.

* `pygtk−gtktextview.xml` (*Description*) Clipboard access is available in PyGTK 2.2.

* `pygtk−gtkwidget.xml` (*get_clipboard*) Available in PyGTK 2.2.

2004−07−21 John Finlay <finlay@moeraki.com>

* `pygtk−gtkclipboard.xml` (*gtk.Clipboard*) Add description of defaults for optional params.
(*gtk.clipboard_get*) Add description of this PyGTK 2.4 function.

* pygtk–gtk–functions.xml
(*gtk.clipboard_get*) Add link for this PyGTK 2.4 function.

* `pygtk-gtkclipboard.xml` (*set_with_data*) Add description of get_func
and clear_func signatures.

2004–07–20 John Finlay <finlay@moeraki.com>

* `pygtk-gtktreeviewcolumn.xml` (*set_sort_column_id*) Expand description
of this convenience method.
(*set_sort_indicator*) Add note re effect of set_sort_column_id() on
use of this method.

2004–07–19 John Finlay <finlay@moeraki.com>

* `pygtk-gtktreesortable.xml` (*set_sort_func*) Add description of
comaprison function return value. Thanks to Andrew Boie.

* `pygtk-gtktreeview.xml` (*get_path_at_pos*) Return value if no path
is None.

2004–07–18 John Finlay <finlay@moeraki.com>

* `pygtk-gtkactiongroup.xml` (*add_actions*)
(*add_toggle_actions*) (*add_radio_actions*) Document new user_data
param.

2004–07–15 John Finlay <finlay@moeraki.com>

* `pygtk-gtkbutton.xml` Add description of optional use_underline
param added in PyGTK 2.4
* `pygtk-gtktogglebutton.xml` Add description of optional use_underline
param added in PyGTK 2.4. Add missing constructor param description.
* `pygtk-gtkcheckbutton.xml` Add description of optional use_underline
param added in PyGTK 2.4.
* `pygtk-gtkradiobutton.xml` Add description of optional use_underline
param added in PyGTK 2.4.
* `pygtk-gtkcheckmenuitem.xml` Add description of optional use_underline
param added in PyGTK 2.4.
* `pygtk-gtkmenuitem.xml` Add description of optional use_underline
param added in PyGTK 2.4.
* `pygtk-gtkradiomenuitem.xml` Add description of optional use_underline
param added in PyGTK 2.4.

2004–07–12 John Finlay <finlay@moeraki.com>

* `pygtk-gdkwindow.xml` (*set_geometry_hints*) Add missing param names.
Thanks to Theo Reed in #147458.

2004–07–09 John Finlay <finlay@moeraki.com>

* `pygtk-gtkbin.xml` (*"child"*) Add description of this attribute.

* pygtk-gtkcheckmenuitem (*"active"*) Add description of this attribute.
(*"indicator−size"*) Add description of this style property.

* pygtk-gtkgammadialog.xml (*Attributes*) Add description of attributes.

2004−07−08 John Finlay <finlay@moeraki.com>

* pygtk-gdkcolor.xml (*gtk.gdk.parse_color*) Add info on exceptions.

* pygtk-gdkcolormap.xml (*alloc_color*) Modify description on
exceptions.

2004−07−04 John Finlay <finlay@moeraki.com>

* pygtk-pygtkgenerictreemodel.xml Fix broken links.

* pygtk-gtktreemodel.xml (*rows_reordered*) (*"rows−reordered"*)
Update these to indicate top level row reordering.

* pygtk-gdkcolormap.xml (*query_color*) Add description of this
PyGTK 2.4 method.

2004−07−03 John Finlay <finlay@moeraki.com>

* pygtk-gdkpixbuf.xml (*subpixbuf*) Add description of this new
PyGTK 2.4 method.

* pygtk-pygtkgenerictreemodel.xml (*Description*) Add info on new
methods invalidate_iters() and iter_is_valid().
(*invalidate_iters*) (*iter_is_valid*) Add description of these
PyGTK 2.4 methods.

* pygtk-gtktreeview.xml (*get_search_equal_func*) Remove. This wasn't
implemented.

* pygtk-gtktreeview.xml (*get_search_equal_func*)
(*set_search_equal_func*) Add description of these PyGTK 2.4 methods.
Fix some typos.

* pygtk-gobject-functions.xml (*io_add_watch*)

* pygtk2-ref.html Bump version number.

=============== 2.4.8 =============
2004−07−01 John Finlay <finlay@moeraki.com>

* pygtk2-ref.xml Change pubdate.

* pygtk-gdkwindow.xml (*set_user_data*) Add info about TypeError
exception.

* pygtk-gtknotebook.xml (*append_page*) (*append_page_menu*)
(*prepend_page*) (*prepend_page_menu*) (*insert_page*) (*insert_page_menu*)
(*set_tab_label*) (*set_menu_label*)

Describe default values for tab_label, menu_label and position params.

* `pygtk-gtkcombobox.xml` (*Description*) Fix typo.
* `pygtk-introduction.xml` Fix broken link
* `pygtk-gtkentrycompletion.xml` Fix typo.
Fixes #145239 thanks to Olav Vitters

2004−06−30 John Finlay <finlay@moeraki.com>

* `pygtk-gtktreesortable.xml` (*Description*)
(*get_sort_column_id*) (*set_sort_func_id*) Add more detail on the
meaning and use of sort column IDs.

* `pygtk-gtktreesortable.xml` (*set_default_sort_func*) Revise
description of sort_func param. Change
gtk.TREE_SORTABLE_DEFAULT_SORT_COLUMN_ID to −1.
(*set_sort_column_id*) Change gtk.TREE_SORTABLE_DEFAULT_SORT_COLUMN_ID
to −1.

2004−06−28 John Finlay <finlay@moeraki.com>

* `pygtk-pygtkgenerictreemodel.xml` (*Properties*) Add description
of the "leak−references" property.

2004−06−20 John Finlay <finlay@moeraki.com>

* `pygtk-gtkwidget.xml` (*allocation*) (*window*) These attributes are
writeable in PyGTK 2.4
* pygtk−gtkwidget.xml
* `pygtk-gtk-functions.xml` (*widget_class_install_style_property*)
Add description of this PyGTK 2.4 function.

* pygtk−gtkcontainer.xml
* `pygtk-gtk-functions.xml` (*container_class_list_child_properties*)
Add description of this PyGTK 2.4 function.

* `pygtk-gtkitemfactory.xml` (*Description*) Add link to gtk.UIManager
in the deprecation message. Thanks to Matthew Bull.

* `pygtk-gdkwindow.xml` (*set_user_data*) (*get_user_data*) Add
description of these PyGTK 2.4 methods.

* `pygtk-gtknotebook.xml` (*insert_page*) (*set_tab_label*) tab_label
parameter can be None in PyGTK 2.4 and above.

* `pygtk-gtkwidget.xml` (*"drag−drop"*) Fix broken link.

* pygtk−gtkaccelgroup.xml
* `pygtk-gtk-functions.xml` (*accel_groups_from_object*) Add
description of this PyGTK 2.4 function.

2004−06−14 John Finlay <finlay@moeraki.com>

* `pygtk-gtkwidget.xml` (*"drag−motion"*) (*"drag−leave"*) (*"drag−end"*)

(*"drag−drop"*) (*"drag−data−received"*) (*"drag−data−get"*)
(*"drag−data−delete"*) (*"drag−begin"*) Update the documentation on
these signals.

2004−06−13 John Finlay <finlay@moeraki.com>

* `pygtk−gtkicontheme.xml` (*Description*) Fix typo and add exception
handling to example. Thanks to Steve Chaplin.

* `pygtk−gtktextview.xml` (*Description*) Add info about popup menu and
selection clipboards.

2004−06−12 John Finlay <finlay@moeraki.com>

* `pygtk−gtkimage.xml` (*"pixbuf"*) Attribute contains a pixbuf not
a pixmap. Thanks to Christian Reis.

2004−06−06 Johan Dahlin <johan@gnome.org>

* `pygtk−gdkevent.xml` : Proper documentation of all attributes, in
sync with latest CVS. Remove the common attribute and only show
them in the beginning of the event list.

2004−06−02 John Finlay <finlay@moeraki.com>

* `pygtk−gtkpaned.xml` (*pack1*) (*pack2*) Change references to expand
param to resize. Thanks to Toon Verstraelen. Fixes #143589

2004−05−31 John Finlay <finlay@moeraki.com>

* `pygtk−gdkwindow.xml` (*add_filter*) Add description of this PyGTK 2.2
method.

2004−05−30 John Finlay <finlay@moeraki.com>

* `pygtk−gtktreemodel.xml` (*"rows−reordered"*) Note that new_order is
a gpointer value.

* `pygtk−gtktreeview.xml` (*"columns−changed"*) Fix confusing wording.
(*"test−collapse−row"*) (*"test−expand−row"*) Update return value
wording.

* `pygtk−gtktreedragdest.xml` (*row_drop_possible*) Add missing word.

2004−05−29 John Finlay <finlay@moeraki.com>

* `pygtk−gtktreeview.xml` (*set_column_drag_function*) Add description
of this PyGTK2.4 method.

* `pygtk−pygtkgenerictreemodel.xml` (*Description*) Add self as
a param to all the methods to be implemented and use rowref
instead of iter to avoid confusion. Correct method name from
on_get_iter_next() to on_iter_next()

* `pygtk-gtktreemodel.xml` (*iter_next*) Fix description.

2004−05−28 John Finlay <finlay@moeraki.com>

* `pygtk-gdkpixbuf.xml` (*save*)
(*gdk.pixbuf_new_from_file*)
(*gdk.pixbuf_new_from_file_at_size*)
(*gdk.pixbuf_new_from_inline*) Note that GError exception is
raised on error.

* `pygtk-gdkpixbufanimation.xml` (*gtk.gdk.PixbufAnimation*) Note that
GError exception is raised on error.

* `pygtk-gdkpixbufloader.xml` (*gtk.gdk.PixbufLoader*)
(*gtk.gdk.pixbuf_loader_new_with_mime_type*)
(*write*) (*close*) Note that GError exception is raised on error.

* `pygtk-gtkiconinfo.xml` (*load_icon*) Note that GError exception
is raised on error.

* `pygtk-gtkicontheme.xml` (*load_icon*) Note that GError exception
is raised on error.

* `pygtk-gtkuimanager.xml` (*add_ui_from_string*)
(*add_ui_from_file*) Note that GError exception is raised on error.

* `pygtk-gtkwindow.xml` (*set_icon_from_file*) Add.
(*gtk.window_set_default_icon_from_file*) Note that GError exception
is raised on error.

* `pygtk-gtkfilechooser.xml` (*add_shortcut_folder*)
(*remove_shortcut_folder*)
(*add_shortcut_folder_uri*)
(*remove_shortcut_folder_uri*) Note that GError exception is raised
on error.

* `pygtk-pangoattrlist.xml` (*pango.parse_markup*) Note that GError
exception is raised on error.

2004−05−27 John Finlay <finlay@moeraki.com>

* `pygtk-gtkcelllayout.xml` (*set_cell_data_func*) Add description
of this PyGTK 2.4 method.

* `pygtk-gtktreeviewcolumn.xml` Add note that GtkTreeViewColumn
implements the GtkCellLayout interface in PyGTK 2.4.

2004−05−25 John Finlay <finlay@moeraki.com>

* `pygtk-introduction.xml` Add reference link to www.pygtk.org
Thanks to Rafael Villar Burke.

* `pygtk-gtkentrycompletion.xml` (*Description*) Fix example code.

2004−05−24 John Finlay <finlay@moeraki.com>

* `pygtk-pygtktreemodelrowiter.xml` (*next*) Fix broken links and name.

2004−05−22 John Finlay <finlay@moeraki.com>

* `pygtk-gtktreemodelfilter.xml` (*set_modify_func*) Add description.
Add to Description re modify function.

* `pygtk2-ref.xml` Bump release number to 2.4.8.

2004−05−21 John Finlay <finlay@moeraki.com>

* `pygtk-gtktreemodel.xml` (*iter_n_children*) Change NULL to None.

2004−05−20 John Finlay <finlay@moeraki.com>

* `pygtk-gtktreeviewcolumn.xml` (*pack_start*)
(*pack_end*) expand param can default to TRUE.

=========== 2.4.7 ==============
2004−05−19 John Finlay <finlay@moeraki.com>

* `pygtk-pygtktreemodelrow.xml` Add.

* `pygtk-pygtktreemodelrowiter.xml` Add.

* `pygtk-gtk-classes.xml` Add pygtk−pygtktreemodelrow.xml and
pygtk−pygtktreemodelrowiter.xml

* `pygtk2-ref.xml` Bump release number to 2.4.7

* `pygtk-gtktreemodel.xml` Add description of mapping and iterator
protocol support.

=========== 2.4.6 ==============
2004−05−17 John Finlay <finlay@moeraki.com>

* `pygtk-gtk-constants.xml` (*gtk−selection−mode−constants*) Fix typo.

* `pygtk-gtktreemodel.xml` (*get*) Remove dangling tag.

* `pygtk2-ref.xml` Bump release number to 2.4.6

2004−05−16 John Finlay <finlay@moeraki.com>

* `pygtk-gtktreeselection.xml` (*set_selection_function*) Fix bogus
description of signature of func.

2004−05−15 John Finlay <finlay@moeraki.com>

* `pygtk-gtktreeselection.xml` Fix method links in Description.
(*set_mode*) Add detail on selection mode and reference to selection
constants.

(*selected_foreach*) Add note.

2004−05−13 John Finlay <finlay@moeraki.com>

* pygtk−gtktreeselection.xml
(*get_selected*) Add info that treeiter is None if no row selected.
(*get_selected_rows*) Correct return value is a tuple with
a tree modle and a list of selected paths.

2004−05−12 John Finlay <finlay@moeraki.com>

* `pygtk-gtktreeview.xml` (*get_dest_row_at_pos*) Change parameters
to x and y from drag_x and drag_y.

2004−05−10 John Finlay <finlay@moeraki.com>

* `pygtk-gtknotebook.xml` (*append_page*) (*append_page_menu*)
(*prepend_page*) (*prepend_page_menu*) (*insert_page*)
(*insert_page_menu*) Add description of return value for PyGTK 2.4
and above − these return a page index.

2004−05−07 John Finlay <finlay@moeraki.com>

* `pygtk-gtktreestore.xml` (*remove*) Fix return value explanation.

* `pygtk-gtkliststore.xml` (*iter_is_valid*) (*reorder*)
(*move_before*) (*move_after*) Change first release designation
to PyGTK 2.2.

* `pygtk-gtktreemodel.xml` (*get*) Add description of this PyGTK 2.4
method.
(*get*) Add column param to description.

2004−05−06 John Finlay <finlay@moeraki.com>

* `pygtk-gtktreestore` (*insert*) (*insert_after*) (*insert_before*)
(*append*) (*prepend*) Add initializers to the Synopsis.

2004−05−05 John Finlay <finlay@moeraki.com>

* `pygtk-gobject-constants` (*gobject−type−constants*) Fix link.

* `pygtk2-ref.xml` Bump release number to 2.4.5

============= 2.4.4 =============

2004−05−05 John Finlay <finlay@moeraki.com>

* `pygtk-gtkmenuitem.xml` (*activate*) Accidentally commented out
description.

* `pygtk-introduction.xml` Unscramble programlisting.

* `pygtk-gobject.xml` Fix Attribute formatting.

* `pygtk-gtkaction.xml` (*get_visible*) Fix link.

* `pygtk-gtkbox.xml` (*pack_start_defaults*) (*pack_end_defaults*) Add deprecation warnings. Remove references in Description.

* `pygtk-gtkcombo.xml` Fix Properties formatting.

* `pygtk-gtkcontainer.xml` Fix Attribute formatting.

* `pygtk-gtkdialog.xml` Fix Attribute formatting.

* `pygtk-gtkfileselection.xml` Fix Attribute formatting.

* `pygtk-gtkfontselectiondialog.xml` Fix Attribute formatting.

* `pygtk-gtkmessagedialog.xml` Fix Attribute formatting.

* `pygtk-gtknotebook.xml` Fix Attribute formatting.

* `pygtk-gtkrequisition.xml` Fix Attribute formatting.

* `pygtk-gtkselectiondata.xml` Fix Attribute formatting.

* `pygtk-gtkstyle.xml` Fix Attribute formatting.

* `pygtk-gtktextattributes.xml` Fix Attribute formatting.

* `pygtk-gtktextbuffer.xml` Fix Attribute formatting.

* `pygtk-gtktogglebutton.xml` Fix Attribute formatting.

* `pygtk-gtktooltips.xml` Fix Attribute formatting.

* `pygtk-gtkwidget.xml` Fix Attribute formatting.

* `pygtk-gtkwindow.xml` Fix Attribute formatting.

* `pygtk-gdkcolor.xml` Fix Attribute formatting.

* `pygtk-gdkdevice.xml` Fix Attribute formatting.

* `pygtk-gdkdragcontext.xml` Fix Attribute formatting.

* `pygtk-gdkdrawable.xml` Fix Attribute formatting.

* `pygtk-gdkevent.xml` Fix Attribute formatting.

* `pygtk-gdkgc.xml` Fix Attribute formatting.

* `pygtk-gdkpixbuf.xml` Fix Attribute formatting.

* `pygtk-gdkrectangle.xml` Fix Attribute formatting.

* `pygtk-gdkvisual.xml` Fix Attribute formatting.

* `pygtk-pangoattribute.xml` Fix Attribute formatting.

* `pygtk-pangocolor.xml` Fix Attribute formatting.

* `pygtk-pangoglyphstring.xml` Fix Attribute formatting.

* `pygtk2-ref.xml` Bump release number to 2.4.4

2004−05−04 John Finlay <finlay@moeraki.com>

* `pygtk-gtkmenuitem.xml` (*toggle_size_request*) Add description.

* `pygtk-gtktextiter.xml` (*backward_find_char*)
(*forward_find_char*) Add descriptions of these methods.
Fix spelling errors.

* `pygtk-gtktextview.xml` Spell checked.

* `pygtk-gtktextbuffer.xml` Spell checked.

* `pygtk-introduction.xml` Add section listing major changes since 1.9.

* `pygtk-gobject-maincontext.xml` Spell checked.

* `pygtk-gobject-functions.xml` Spell checked.

* `pygtk-gobject-constants.xml` Spell checked.

* `pygtk-gtkaccellabel.xml` Spell checked.

* `pygtk-gtkaction.xml` Spell checked.

2004−04−30 John Finlay <finlay@moeraki.com>

* `pygtk2-ref.xml` Bump release number to 2.4.3

* pygtk−gtkaction.xml
* pygtk−gtkactiongroup.xml
* pygtk−gtkborder.xml
* pygtk−gtkclipboard.xml
* pygtk−gtkcolorbutton.xml
* pygtk−gtkcombobox.xml
* pygtk−gtkcomboboxentry.xml
* pygtk−gtkentrycompletion.xml
* pygtk−gtkexpander.xml
* pygtk−gtkfilechooser.xml
* pygtk−gtkfilechooserdialog.xml
* pygtk−gtkfilechooserwidget.xml
* pygtk−gtkfilefilter.xml
* pygtk−gtkfontbutton.xml
* pygtk−gtkiconinfo.xml
* pygtk−gtkicontheme.xml

* pygtk−gtkradioaction.xml
* pygtk−gtktoolbutton.xml
* pygtk−gtkseparatortoolitem.xml
* pygtk−gtktoggleaction.xml
* pygtk−gtktoggletoolitem.xml
* pygtk−gtktoolbutton.xml
* pygtk−gtktoolitem.xml
* pygtk−gtktreemodelfilter.xml
* pygtk−gtktreerowreference.xml
* pygtk−gtkuimanager.xml
Add a note indicating which release of PyGTK the above were new in.

============ Release 2.4.2 ===============

2004−04−29 John Finlay <finlay@moeraki.com>

* `pygtk-gtkclipboard.xml` Fix notes to correct release availabilities for object, constructor and methods.

* `pygtk-introduction.xml` Update the introduction re: PyGTK 2.4 and Pango. Also add description of Child Properties.

* `pygtk-gtktreestore.xml` (*remove*) Add return value description.

* `pygtk-gtkliststore.xml` (*remove*) Add return value description.

* `pygtk2-ref.xml` Bump release level to 2.4.2

2004−04−27 John Finlay <finlay@moeraki.com>

* `pygtk-gobject-function.xml` (*io_add_watch*) Add description of callback signature.

* `pygtk2-ref.xml` Bump release level to 2.4.1

===== Release 2.4 =====

2004−04−27 John Finlay <finlay@moeraki.com>

* `pygtk-gtkclipboard.xml` Fix link to gtk.gdk.atom−intern().

* `pygtk-gtkexpander.xml` Fix link to pango.parse_markup().

* `pygtk-gtkicontheme.xml` Fix link to gtk.gdk.pixbuf_new_from_inline() function.

* `pygtk-gtkscale.xml` Fix link to pango.PIXELS().

* `pygtk-gtkwidget.xml` Fix link to gtk.gdk.Screen.

* `pygtk-gtkwindow.xml` Fix link to Gravity Constants.
Remove bogus link to gdk_notify_startup_complete() function.

* `pygtk-gtk-constants.xml` Fix links to gtk.gdk.DragConstext.finish() method.

* `pygtk-pangocolor.xml` Fix link to gobject.GBoxed.

* `pygtk-hierarchy.xml` Tweak formatting.

* `pygtk-hierarchy.xml` Use PyGTK 2.4 hierarchy.

* `pygtk-hierarchy.xml` Remove undocumented ATK entries, etc.

* `pygtk-pango-constants.xml` Remove reference to unimplemented functions.

* `pygtk-gdkwindow.xml` Remove reference to unimplmented function.

* `pygtk-gdkdrawable.xml` Remove reference to unimplmented function.

* `pygtk-gtkaccelgroup.xml` Remove reference to unimplemnted function.

* `pygtk-gobject-ginterface.xml` Add.

* `pygtk-gobject-classes.xml` Add pygtk–gobject–ginterface.xml

* `pygtk2-ref.xml` Update date and set version to 2.4

* `pygtk-gtkclipboard.xml` Remove reference to unimplemnted function. Add initlaizers for user_data in request_* methods.

* `pygtk-gtk-classes` Add pygtk–gtk–constants.xml

2004–04–26 John Finlay <finlay@moeraki.com>

* `pygtk-gdk-constants.xml` Add.

* `pygtk-gdk-classes.xml` Add pygtk–gdk–constants.xml to list.

* `pygtk-pango-constants.xml` Reorganize with a Synopsis and add a description for each set of constants.

* `pygtk-gobject-constants.xml` Add.

* `pygtk-gobject-gboxed.xml` Add.

* `pygtk-gobject-gboxed.xml` Remove gobject from Ancestry.

* `pygtk-gobject-gboxed.xml` Fix typo.

* `pygtk-gobject-maincontext.xml` Add.

* `pygtk-gobject-mainloop.xml` Add.

* `pygtk-gobject-classes.xml` Add pygtk–gobject–gboxed.xml, pygtk–gobject–maincontext.xml and pygtk–gobject–mainloop.xml to the list of files to be processed.

* `pygtk-gobject-maincontext.xml` Add description of constructor.

* `pygtk-gobject.xml` Move Ancestry section ahead of Description.

* `pygtk-gobject.xml` Add some attribute descriptions.

* `pygtk-gobject-gpointer.xml` Add.

* `pygtk-gtk-constants.xml` Add.

2004−04−25 John Finlay <finlay@moeraki.com>

* `pygtk-pango-functions.xml` Add.

* `pygtk-pangolanguage.xml` Add pango functions to cover the real
API pre−PyGTK 2.4.

* `pygtk-pango-functions.xml` Remove extraneous sections.

* `pygtk-gdkscreen.xml` (*"size−changed"*) Add description.

* `pygtk-gtk-stock-items.xml` (*gtk.stock_lookup*)
(*gtk.stock_list_ids*) Add reference links.

* `pygtk-pango-markup.xml` (*"fallback"*) Add description of this
attribute type.

* `pygtk-pangoattribute.xml` Reorder attributes.

* `pygtk-pangofontmap.xml` (*get_shape_engine_type*) Add description.

* `pygtk-gdk-classes.xml` Add new xml files for processing.

* `pygtk-gdk-functions.xml` Add links to all module functions.

* `pygtk-pango-classes.xml` Add class references for FontsetSimple
and pango functions for processing.

* `pygtk-pango-constants.xml` Add.

* `pygtk-pango-classes.xml` Add Pango Constants ref.

* `pygtk-gdkgc.xml` Fix typo.

* `pygtk-gdkwindow.xml` (*get_state*) Add to WINDOW_STATE flag list.

2004−04−24 John Finlay <finlay@moeraki.com>

* `pygtk-pangoattribute.xml` Fix typo. Reorder attributes.

* `pygtk-pangoattrlist.xml` (*filter*) Add description of this
PyGTK 2.4 method.

* `pygtk-pangocontext.xml` Fix link in Description.

* `pygtk-pangofontface.xml` (*list_sizes*) Add desciption of this PyGTK 2,4 method.

* `pygtk-pangofontfamily.xml` (*is_monospace*) Add description of this PyGTK 2.4 method.

* `pygtk-pangofontset.xml` Fix typo in Description. (*foreach*) Add description of this PyGTK 2.4 method.

* `pygtk-pangofontsetsimple.xml` Add.

2004−04−23 John Finlay <finlay@moeraki.com>

* `pygtk-gdkwindow.xml` (*set_accept_focus*) (*get_group*)
Add descriptions of these methods.
(*gtk.gdk.window_lookup_for_display*)
(*gtk.gdk.window_foreign_new_for_display*)
Add description for these functions.

* `pygtk-gdkatom.xml` Fix constructor link.

* `pygtk-gdkdisplaymanager.xml` Fix gdkdisplay link.

* `pygtk-gdkkeymap.xml` Fix signal links.
Fix keyval_get_keys link − change to lookup_key method.
(*"keys−changed"*) Add description of this GTK+ 2.2 signal.
(*gtk.gdk.keyval_to_lower*) Fix name.

* pygtk−gdk−functions.xml
(*gtk.gdk.screen_width*) (*gtk.gdk.screen_height*)
(*gtk.gdk.screen_width_mm*) (*gtk.gdk.screen_height_mm*)
Remove these descriptions − now with GdkWindow.
Add links to all functions.

* `pygtk-gdkwindow.xml` Fix links to gtk.Window.set_wmclass method.
fix reference to gtk.gdk.window_constrain_size function

* `pygtk-gdkvisual.xml` (*visual_get_best_with_type*)
Add missing parameter.

* `pygtk-gdkpixbuf.xml` (*gtk.gdk.pixbuf_new_from_file_at_size*)
Fix parameter name.

* `pygtk-gdk-functions.xml` Fix reference to gtk.gdk.atom_intern.
Update the description. Reorder function listing.

* `pygtk-gtkvseparator.xml` Fix link to constructor.

* `pygtk-gtkwindow.xml` (*gtk.window_set_auto_startup_notification*)
Add description for this PyGTK 2.2 function.

* `pygtk-gtkobject.xml` (*gtk.bindings_activate_event*) Add.
(*gtk.bindings_activate*) Fix name.

* `pygtk-gtk-stock-items.xml` (*gtk.stock_add*)
(*gtk.stock_lookup*) (*gtk.stock_list_ids*) Add description of
these functions.

* `pygtk-gtk-functions.xml` Add links to all functions and update
description.

* `pygtk-gtkwindow.xml` Fix typo.

* `pygtk-gtk-functions.xml` Add gtkWidget function links.

* `pygtk-pangoattribute.xml` (*AttrFallback*) Add description of this
PyGTK 2.4 function.

* `pygtk-pangoattriter.xml` (*get_attrs*) Add description of this
PyGTK 2.4 method.

2004−04−22 John Finlay <finlay@moeraki.com>

* pygtk−gdkpixbuf.xml
(*gtk.gdk.pixbuf_new_from_data*) (*gtk.gdk.pixbuf_new_from_array*)
Add descriptions of these PyGTK 2.2 functions.
(*gtk.gdk.pixbuf_new_from_file_at_size*)
(*gtk.gdk.pixbuf_get_formats*) (*gtk.gdk.pixbuf.get_file_info*)
Add descriptions of these PyGTK 2.4 functions.
(*"bits−per−sample"(("colorspace"*) (*"has−alpha"*) (*"height"*)
(*"n−channels"*) (*"pixels"*) (*"rowstride"*) (*"width"*)
Add descriptions fo these GTK+ 2.4 properties.

* `pygtk-gdkpixbuf.xml` (*gtk.gdk.pixbuf_get_formats*) Tweak description.

* `pygtk-gdkpixbufloader.xml` (*set_size*) (*get_format*)
Add descriptions for these PyGTK 2.4 methods.
(*gtk.gdk.pixbuf_loader_new_with_mime_type*)
Add description for this PyGTK 2.4 function
(*"size−prepared"*) Add desription of this GTK+ 2.2 signal.

* `pygtk-gdkpixmap.xml` (*gtk.gdk.pixmap_foreign_new_for_display*)
(*gtk.gdk.pixmap_lookup_for_display*) Add descriptions for these
PyGTK 2.2 functions.

* `pygtk-gdkrectangle.xml` (*union*) Fix typo.

* `pygtk-gdkscreen.xml` Add.
Add purpose description.

* `pygtk-gdkvisual.xml` Reorder attribute list.
(*get_screen*) Add desciption of this PyGTK 2.2 method.
(*query_depths*) (*query_visual_types*)
Add description of these PyGTK 2.4 functions.

* `pygtk-gdkwindow.xml` (*set_keep_above*) (*set_keep_below*)
(*set_skip_taskbar*) (*set_skip_pager*) (*set_geometry_hints*)

(*get_deskrelative_origin*) (*set_icon_list*)
(*fullscreen*) (*unfullscreen*)
Add descriptions of these methods.
(*gtk.gdk.window_at_pointer*)
Add description for this function.

2004−04−21 John Finlay <finlay@moeraki.com>

* `pygtk-gdkgc.xml` (*get_screen*) Add description of this PyGTK 2.2 method

* `pygtk-gdkkeymap.xml` (*keymap_get_for_display*) Add desciption of this PyGTK 2.2 function.
(*keyval_convert_case*) Add description of this PyGTK 2.4 function.
(*get_entries_for_keyval*)
(*get_entries_for_keycode*)
(*lookup_key*)
(*translate_keyboard_state*) Add descriptions of these PyGTK 2.4 methods.

* `pygtk-gdkevent.xml` (*gtk.gdk.KEY_PRESS*) Add "hardware_keycode" attribute description for PyGTK 2.2. Add "group" attribute description for PyGTK 2.4.

2004−04−20 John Finlay <finlay@moeraki.com>

* `pygtk-gtk-classes.xml` Add pygtk−gtk−stock−items.xml to list

* `pygtk-gtk-stock-items.xml` Fix reference to stock_network_24.png and stock_new_24.png

* `pygtk-gdkatom.xml` Tweak the description.

* `pygtk-gdkcolormap.xml` (*get_screen*) Add description of PyGTK 2.4 method.
(*gtk.gdk.colormap_get_system*) Add reference to gtk.gdk.Screen.get_system_colormap() method.

* `pygtk-gdkcursor.xml` (*get_display*) Add description of this PyGTK 2.2 method.
Add descriptions of 2 more constructor signatures for PyGTK 2.4.

* `pygtk-gdkdevice.xml` Reorder attribute list.

* `pygtk-gdkatom.xml` (*atom_intern*) Note that value of only_if_exists is ignored.

* `pygtk-gdkdisplay.xml` Add.

* `pygtk-gdkdisplaymanager.xml` Add.

* `pygtk-gdkdragcontext.xml` (*find_window_for_screen*) Add description for this PyGTK 2.2 method.

* `pygtk-gdkdrawable.xml` (*get_screen*) (*get_display*) (*draw_pixbuf*)

Add descriptions of these PyGTK 2.2 methods. Update the Description.

* `pygtk-gdkdrawable.xml` (*"xid"*) (*"handle"*)
Add attribute descriptions.

* `pygtk-gdkevent.xml` (*get_state*) Add note about availability.
Added description of event masks.
(*gtk.gdk.Event*) Added for PyGTK 2.2.

* `pygtk-gdkevent.xml` (*set_screen*) (*get_screen*) Add descriptions of these PyGTK 2.2 methods.

2004−04−19 John Finlay <finlay@moeraki.com>

* `pygtk-gtktreeview.xml` (*expand_to_path*) (*map_expanded_rows*) (*set_cursor_on_cell*)
Add descriptions of these methods added in PyGTK 2.2
(*"fixed−height−mode"*) (*"even−row−color"*) (*"odd−row−color"*)
Add descriptions of these properties.
Add signal return value descriptions as needed.

* `pygtk-gtktreeviewcolumn.xml` (*set_expand*) (*get_expand*) (*cell_get_position*)
Add description of these PyGTK 2.4 methods.
(*focus_cell*) Add description of this PyGTK 2.2 method.
(*"expand"*) Add description of this GTK+ 2.4 signal.

* `pygtk-gtkuimanager.xml` Add.

* `pygtk-gtkvbuttonbox.xml` Tweak description.

* `pygtk-gtkwidget.xml` (*set_no_show_all*) (*get_no_show_all*) (*queue_resize_no_redraw*) (*can_activate_accel*) (*get_clipboard*) (*get_screen*) (*has_screen*) (*get_display*) (*get_root_window*)
Add descriptions of these PyGTK 2.4 methods.
(*gtk.widget_list_style_properties*)
Add description of this PyGTK 2.4 function.
(*get_size_request*) Add description.
(*get_accessible*) Remove description.
(*"can−activate−accel"*) (*"screen−changed"*) (*"key−release−event"*)
Add descriptions for these signals.

* `pygtk-gtkwindow.xml` (*set_screen*) (*get_screen*) (*is_active*) (*has_toplevel_focus*) (*activate_key*) (*propagate_key_event*) (*fullscreen*) (*unfullscreen*) (*set−keep−above*) (*set−keep−below*)
Add description for these PyGTK 2.2 and 2.4 methods.
(*"accept−focus"*) (*"decorated"*) (*"gravity"*) (*"role"*) (*"has−toplevel−focus"*) (*"is−active"*) (*"screen"*) (*"skip−pager−hint"*) (*skip−taskbar−hint"*) (*"type−hint"*)
Add decriptions for these GTK+ 2.2 adn 2.4 properties.
(*gtk.set_default_icon*) (*gtk.set_default_icon_from_file*)
Add descriptions of these functions.

* `pygtk-gtktoolitem.xml` Fix reference to gtk.Button.set_relief() method.

* `pygtk-gtktoolbar.xml` Fix reference to gtk.SeparatorToolItem class.

* `pygtk-gtkseparatortoolitem.xml` Fix reference to gtk.SeparatorToolItem class.

* `pygtk-gtkscale.xml` Fix reference to get_layout_offsets() method.

* `pygtk-gtknotebook.xml` Fix formatting error.

* `pygtk-gtk-classes.xml` Add missing pygtk−gtkiconinfo.xml include.

* `pygtk-gtkwindow.xml` Fix references to gtk.Widget "window−state−event" signal.

* `pygtk-gdkatom.xml` Fix reference to gtk.atom_intern() constructor.

* `pygtk-gtkwidget.xml` Fix reference to gtk.widget_set_default_direction() function. Fix references and id for gtk.widget_list_style_properties function.

* `pygtk-gtkicontheme.xml` Fix reference to gtk.pixbuf_new_from_inline function.

* `pygtk-gtk-stock-items.xml` Add.

2004−04−18 John Finlay <finlay@moeraki.com>

* `pygtk-gtktreemodelfilter.xml` Add.

* `pygtk-gtktreemodelsort.xml` (*iter_is_valid*) Add description of PyGTK 2.2 method.

* `pygtk-gtktreerowreference.xml` Add.

* `pygtk-gtktreeselection.xml` (*get_selected_rows*) (*count_selected_rows*) (*unselect_range*) Add new methods for PyGTK 2.2

* `pygtk-gtktreestore.xml` (*iter_is_valid*) (*reorder*) (*swap*) (*move_after*) (*move_before*) Add descriptions for these PyGTK 2.2 methods.

2004−04−17 John Finlay <finlay@moeraki.com>

* `pygtk-gtktoolbutton.xml` Add.

* `pygtk-gtktoolitem.xml` Add.

* `pygtk-gtktoolitem.xml` (*set_tooltip*) tip_text and tip_private can default to None.

* `pygtk-gtkwindow.xml` (*tooltips_get_info_from_tip_window*) Add

description of PyGTK 2.4 method.

* `pygtk-gtktreeiter.xml` Add class info for TreeIter creation methods.

* `pygtk_gtktreemodel.xml` (*filter_new*) Add description of new PyGTK 2.4 method.

2004−04−16 John Finlay <finlay@moeraki.com>

* `pygtk-gtksocket.xml` (*"plug−removed"*) Add return value description for this signal handler.

* `pygtk-gtkspinbutton.xml` (*"input"*) (*"output"*) Add return value descriptions. Reorder property descriptions.

* `pygtk-gtkspinbutton.xml` (*"input"*) (*"output"*) Fix typos.

* `pygtk-gtkstatusbar.xml` (*"has−resize−grip"*) Add GTK+ 2.4 property description.

* `pygtk-gtk-functions.xml` (*gtk.draw_insertion_cursor*) Add.

* `pygtk-gtktable.xml` (*attach_defaults*) Remove − it's deprecated. Reorder property descriptions.

* `pygtk-gtktextbuffer.xml` (*create_tag*) Remove keyword role. (*select_range*) (*add_selection_clipboard*) (*remove_selection_clipboard*) (*cut_clipboard*) (*copy_clipboard*) (*paste_clipboard*) Add for PyGTK 2.4

* `pygtk-gtktextiter.xml` (*forward_visible_word_end*) (*backward_visible_word_start*) (*forward_visible_word_ends*) (*backward_visible_word_starts*) Add descriptions for PyGTK 2.4

* `pygtk-gtktexttag.xml` Reorder property descriptions. (*"event"*) Add return value description.

* `pygtk-gtktexttagtable.xml` (*foreach*) Add description for PyGTK 2.4

* pygtk−gtkaccelgroup.xml
pygtk−gtkpaned.xml
pygtk−gtkscrolledwindow.xml
pygtk−gtkspinbutton.xml
pygtk−gtktreeview.xml
pygtk−gtkwidget.xml
pygtk−gtkwindow.xml Make keycombos simultaneous.

* `pygtk-gtktextview.xml` (*"move−viewport"*) (*"select−all"*) Add descriptions of these signals for GTK+ 2.4 and 2.2.
(*"accepts−tab"*) (*"overwrite"*) Add descriptions of these GTK+ 2.4 properties.
(*set_accepts_tab*) (*get_accepts_tab*) (*set_overwrite*) (*get_overwrite*) Add descriptions of these PyGTK 2,4 methods.

* `pygtk-gtktoggleaction.xml` Add.

* `pygtk-gtktoggletoolbutton.xml` Add.

* `pygtk-gtktoolbar.xml` Add new GTK+ 2.4 method descriptions.
Add deprecation warnings for old toolbar methods.

2004−04−15 John Finlay <finlay@moeraki.com>

* `pygtk-gtkmessagedialog.xml` (*set_markup*) Add description of this
PyGTK 2.4 method.

* `pygtk-gtknotebook.xml` (*get_n_pages*) Add description.
Add GTK+2.4 style properties
Add return value description for "focus−tab" and "select−page"
signals.

* `pygtk-gtkoptionmenu.xml` Added deprecation warning for PyGTK 2.4

* `pygtk-gtkpaned.xml` (*get_child1*) (*get_child2*) Add descriptions of
these PyGTK 2.4 methods.
(*"max−position"*) (*"min−position"*) Add descriptions of these GTK+ 2.4
properties.
(*"resize"*) (*"shrink"*) Add descriptions of these GTK+ 2.4 child
properties.
Add return value description for signals.

* `pygtk-gtkplug.xml` (*gtk.plug_new_for_display*) Add description.
(*construct*) Add warning − method not available since PyGTK 2.2

* `pygtk-gtkradioaction.xml` Add.

* `pygtk-gtkradiobutton.xml` Add description of "group−changed" signal
for GTK+ 2.4

* `pygtk-gtkradiomenuitem.xml` Add description of "group−changed" signal
for GTK+ 2.4

* `pygtk-gtkradioaction.xml` Fix description of "group" property.

* `pygtk-gtkradiotoolbutton.xml` Add.

* `pygtk-gtkrange.xml` Reorder property descriptions.

* `pygtk-gtkrcstyle.xml` Add gtk.rc_reset_style() function for PyGTK 2.4

* `pygtk-gtkscale.xml` (*get_layout*) (*get_layout_offsets*) Add
descriptions for PyGTK 2.4

* `pygtk-gtkscrollbar.xml` Reorder property descriptions.

* `pygtk-gtkscrolledwindow.xml` Add "scrollbar−spacing" style property
available in GTK+ 2.2

* `pygtk_gtkselectiondata.xml` Add function description of

gtk.selection_owner_set_for_display().

* `pygtk_gtkseparatortoolitem.xml` Add. Add purpose description

* `pygtk-gtksettings.xml` (*gtk.settings_get_for_screen*) Add description. Add various properties for GTK+ 2.2 and 2.4

2004−04−14 John Finlay <finlay@moeraki.com>

* `pygtk-gtkiconsource.xml` Add gtk.icon_size_lookup_for_settings() function.

* `pygtk-gdkpixmap.xml` Fix description and parameter list of gtk.pixmap_create_from_data() function.

* `pygtk-gtkicontheme.xml` Add.

* `pygtk-gtkiconinfo.xml` Add.

* `pygtk-gtkinvisible.xml` Add the set_screen() and get_screen() methods available in PyGTK 2.2

* `pygtk-gtkitemfactory.xml` Add deprecation warning.

* `pygtk-gtklabel.xml` Fix typo.

* `pygtk-gtkliststore.xml` (*set_column_types*) (*iter_is_valid*) (*reorder*) (*swap*) (*move_after*) (*move_before*) Add descriptions for these methods.

* `pygtk-gtkmenu.xml` (*set_screen*) (*attach*) (*set_monitor*) Add descriptions for these methods.

* `pygtk-gtkmenu.xml` (*"move−scroll"*) Add description for this GTK+ 2.2 signal. Add descriptions for the style and child properties added in GTK+ 2.4

* `pygtk-gtkmenuitme.xml` (*"toggle−size−allocate"*) Fix typo.

* `pygtk-gtkmenushell.xml` (*select_first*) (*cancel*) Add descriptions for these methods in PyGTK 2.2 and 2.4 respectively.

2004−04−13 John Finlay <finlay@moeraki.com>

* `pygtk-gtkexpander.xml` Add.

* `pygtk-gtkimcontext.xml` Add return value description for "delete−surrounding" and "retrieve−surrounding" signals.

* `pygtk-gtkclipboard.xml` Add the request_* and wait_for_targets method descriptions.

2004−04−10 John Finlay <finlay@moeraki.com>

* pygtk-gtkactiongroup.xml (*add_toggle_actions*) Add description of is_active item in toggle action tuple.
(*add_radio_actions*) Add description of value and on_change parameters.
Remove callback description from radio action tuple.

* pygtk-gdkatom.xml Add.

* pygtk-gdk-functions.xml (*atom_intern*) Remove.
Fix dumb error.

* pygtk-gtkclipboard.xml Add.

* pygtk-gdk-classes.xml Add pygtk–gdkatom.xml.
Hide unimplemented new classes

* pygtk-gtk-classes Hide unimplemented new classes

* pygtk–gtkaccelgroup.xml
* pygtk–gtkbutton.xml
* pygtk–gtkentrycompletion.xml
* pygtk–gtkfilechooserdialog.xml
* pygtk–gtkfontselectiondialog.xml
* pygtk–gtkgenericcellrenderer.xml
* pygtk-gdkatom.xml Fix some link errors.

2004–04–09 John Finlay <finlay@moeraki.com>

* pygtk-gtkaction.xml Add.

* pygtk-gtkactiongroup.xml Add.

* pygtk-gtkborder.xml Add.

2004–04–08 John Finlay <finlay@moeraki.com>

* pygtk-gtkfilefilter.xml Fix links to get_needed() method. Fix custom function signature and description.

* pygtk-gtkfontbutton.xml Add.

* pygtk-gtkfontbutton.xml Add purpose description. Fix GtkFontSelectionDialog links.

* pygtk-gtkfontselectiondialog.xml Add note about lack of font filters, etc. in GTK+ 2.2 and above.

* pygtk-gtkhandlebox.xml Add "snap–edge–set" property description.
GTK+ 2.2 and above.

* pygtk-gtkhbuttonbox.xml Tweak the description.

2004–04–07 John Finlay <finlay@moeraki.com>

* pygtk-gtkfilechooser.xml Add.

* `pygtk-gtkfilechooserdialog.xml` Add.
* `pygtk-gtkfilechooserwidget.xml` Add.
* `pygtk-gtkfilefilter.xml` Add.

* `pygtk-gtkfilefilter.xml` (*add_custom*) Doesn't take keyword args.

2004−04−06 John Finlay <finlay@moeraki.com>

* `pygtk-gtkeventbox.xml` Add new methods: set_visible_window(), get_visible_window(), set_above_child() and get_above_child(); and properties: "above−child" and "visible−window" for PyGTK 2.4 and GTK+ 2.4.

2004−04−05 John Finlay <finlay@moeraki.com>

* `pygtk-gtkcomboboxentry.xml` Add.

* `pygtk-gtkcontainer.xml` Fix bug in synopsis of get_focus_chain().

* `pygtk-gtkcontainer.xml` Property "child" is read−only.

* `pygtk-gtkdrawingarea.xml` Tweak the description.

* `pygtk-gtkcombobox.xml`
* `pygtk-gtkcomboboxentry.xml` Add gtk.CellLayout to class synopsis.

* `pygtk-gtkeditable.xml` Tweak description to specify it's an interface.

* `pygtk-gtkentry.xml` Add PyGTK 2,4 methods set_alignment(), get_alignment(), set_completion(), get_completion(). Also add GTK+2.4 property "xalign"

* `pygtk-gtkentrycompletion.xml` Add.

2004−04−04 John Finlay <finlay@moeraki.com>

* `pygtk-gtkcombobox.xml` Add.

* `pygtk-gtkcelllayout.xml.` Add.

2004−04−03 John Finlay <finlay@moeraki.com>

* `pygtk-gtkcheckbutton.xml` Fix formatting of Ancestry description.

* `pygtk-gtkcheckmenuitem.xml` Minor reformatting.

* `pygtk-gtkcolorbutton.xml` Add description of GtkColorButton in GTK+ 2.4 and PyGTK 2.4.

* `pygtk-gtkcombo.xml` Added deprecation warning for PyGTK 2.4.

2004−04−02 John Finlay <finlay@moeraki.com>

* `pygtk-gtkbutton.xml` Added descriptions of "focus−on−click", "xalign"

and "yalign" propertiees for GTK+ 2.4. Added descriptions of the set_alignment(), get_alignment(), set_focus_on_click() and get_focus_on_click() methods available in PyGTK 2.4

* `pygtk-gtkbuttonbox.xml` Add description of get_child_secondary() method available in PyGTK 2.4.

* `pygtk-gtkcalendar.xml` Add descriptions of get_display_options() and set_display_options() methods available in PyGTK 2.4. Add deprecation warning for display_options() method.

* `pygtk-gtkcellrenderer.xml` Add description of editing_canceled() method available in PyGTK 2.4. Add description of "editing−canceled" signal available in GTK+ 2.4.

2004−04−01 John Finlay <finlay@moeraki.com>

* `pygtk-gtkaccelgroup.xml` Added descriptions of connect_by_path() method, and accel_map_add_entry(), accel_map_lock_path and accel_map_unlock_path() functions.

* `pygtk-gtkalignment.xml` Fix small bug in get_padding() description.

2004−03−31 John Finlay <finlay@moeraki.com>

* `pygtk-gtkalignment.xml` Add description of set_padding() and get_padding() methods available in PyGTK 2.4

2004−03−30 John Finlay <finlay@moeraki.com>

* `pygtk-gtkcellrenderertoggle.xml` Add description of "inconsistent" property added in GTK+ 2.2. Update other property descriptions.

* `pygtk-gtkcheckmenuitem.xml` Add description of "draw−as−radio" property fro GTK+ 2.4 and add PyGTK2.4 methods set_draw_as_radio() and get_draw_as_radio().

2004−03−29 John Finlay <finlay@moeraki.com>

* `pygtk-gtkaccelgroup.xml` Add documentation on the connect() method. Add a description of the return value for the "accel−activate" signal.

* `pygtk-gtkadjustment.xml` Add description of properties available in GTK+2.4. Reformat the Attributes description.

* `pygtk-gtkalignment.xml` Add description of padding properties available in GTK+ 2.4.

* `pygtk-gtkaccessible.xml` Added one line description.

* `pygtk-gtkcalendar.xml` Added description of properties available in GTK+2.4.

* `pygtk-gtkcellrenderertext.xml` Added description of properties

available in GTK+2.4.

2004−03−25 John Finlay <finlay@moeraki.com>

* `pygtk-gdkwindow.xml` Remove description of atom_intern function.

* `pygtk-gdk-function.xml` Add description of atom_intern function. Fixes #137935 (*thanks to Erik Grinaker*).

* `pygtk-gtkdrawingarea.xml` Add explanation that drawing is done on the contained gtk.gdk.Window using the gdk.Drawable methods. Fixes #136297. (*thanks to pachi@mmn−arquitectos.com*)

2004−03−22 John Finlay <finlay@moeraki.com>

* `pygtk-gtkmessagedialog.xml` (*MessageDialog*) Remove gtk.DIALOG_NO_SEPARATOR from the list of valid flags. Fixes #136984. (*thanks to j.jordens and Eric Grinaker*).

2004−04−13 Erik Grinaker <erikg@wired−networks.net>

* `pygtk-gdkwindow.xml` (*raise*): Changed name of raise() to
raise_(), and added a note to explain this is because
raise is a reserved Python keyword.

2004−01−28 John Finlay <finlay@moeraki.com>

* `pygtk-gtktogglebutton.xml` (*get_mode*)
(*set_mode*) Clarify the use of these methods as applied to checkbuttons
and radiobuttons and the results. (*thanks to Antonio A. A.*)

* `pygtk2-ref.xml` Change date. Version number to 1.9

2003−11−18 John Finlay <finlay@moeraki.com>

* `pygtk-gtktreemodel.xml` (*foreach*) Add description of function signature.

2003−10−07 John Finlay <finlay@moeraki.com>

* `pygtk2-ref.xml` Change date. Version number to 1.8

* `pygtk-gtkcellrenderertext.xml` Fix signature an ddescription of the "edited" signal to add path and new_text arguments. (thanks to Steve Chaplin)

* `pygtk-gtkcellrenderertoggle.xml` Fix signature and description of "toggled" signal to add path argument. (*thanks to Steve Chaplin*)

2003−08−25 John Finlay <finlay@moeraki.com>

* `pygtk-gtkadjustment.xml` (*set_all*) Remove deprecation note and add keyword designations.

* `pygtk-gdkpixbufloader.xml` (*write*) Set initializer on parameter count and change description to indicate how count is used.

* `pygtk-gtktextbuffer.xml` (*create_tag*) tag_name defaults to None.

2003−08−23 John Finlay <finlay@moeraki.com>

* `pygtk-gdkpixbuf.xml` Note that pixel_array attribute array contents can be changed.

* `pygtk-gdk-functions.xml` (*threads_init*) Remove comments about broken threads imiplementation which is now fixed in 1.99.17+.

* `pygtk-gtktreemodel.xml` (*rows_reordered*) Add description of this method.

* `pygtk-gtktreeview.xml` (*enable_model_drag_source*)
(*emable_model_drag_dest*)
(*get_drag_dest_row*)
(*get_dest_row_at_pos*)
(*enable_model_drag_source*)
(*enable_model_drag_dest*) Add descriptions of these methods.

* `pygtk-gtkwidget.xml` (*drag_source_set*) Fix typo.

* `pygtk-gtkdialog.xml` (*add_buttons*) Add description of this method.

* `pygtk-gobject.xml` (*handler_is_connected*) Add description of this method.

* `pygtk2-ref.xml` Change date. Version number to 1.7

* `pygtk-gtkdialog.xml` (*add_buttons*) Fix typo in synopsis.

* `pygtk-gobject.xml` (*handler_is_connected*) Fix link in synopsis.

* `pygtk-gdkwindow.xml` Fix link references to function−gdk−−window−foreign−new and function−gdk−−visual−get−system.

2003−07−20 John Finlay <finlay@moeraki.com>

* `pygtk2-ref.xml` Change date. Version number to 1.6

* `pygtk-gdkwindow.xml` (*gtk.gdk.Window*) Add documentation on the constructor.

2003−07−17 John Finlay <finlay@moeraki.com>

* `pygtk2-ref.xml` Change date and version number to 1.5.

* `pygtk-gtkeditable.xml` Remove extraneous "widget" parameter from "insert−text" signal description. Add description indicating position can't be retrieved in PyGTK.

2003−07−16 John Finlay <finlay@moeraki.com>

* `pygtk2-ref.xml` Change date and version number to 1.4.

* `pygtk-gtkitmefactory.xml` (*gtk.ItemFactory*)
(*construct*) Change container_type to refer to PyGTK types: gtk.Menu,
gtk.MenuBar and gtk.OptionMenu.

2003−07−15 John Finlay <finlay@moeraki.com>

* `pygtk2-ref.html` Change to version 1.3

* `pygtk-gtkliststore.xml` (*insert*)
(*insert_after*)
(*insert_before*)
(*prepend*)
(*append*) Add an entry for the return value for these methods.

* `pygtk-gtktoolbar.xml` (*insert_stock*) Remove extraneous stock_id entry.

2003−07−12 John Finlay <finlay@moeraki.com>

* `pygtk2-ref.html` Change version number.

* `pygtk-pangoattriter.xml` Change to pango.AttrIterator to match
GTK+and PyGTK naming.

* `pygtk-pangoattrlist.xml` (*get_iterator*) Change to reference
pango.AttrIterator.

2003−07−11 John Finlay <finlay@moeraki.com>

* `pygtk2-ref.xml` Change date tag to pubdate tag so it will be
displayed.

* `pygtk-pangolayout.xml` Correct the references to the Pango Markup
Language reference page.

* `pygtk-gtk-functions.xml` Fix incorrect reference to gtk.Object class.

* `pygtk-gtktextview.xml` Remove reference to pango.TabArray.free

* `Pango` Add content to pango class files (*they were empty*):
pygtk−pangoattribute.xml
pygtk−pangoattriter.xml
pygtk−pangoattrlist.xml
pygtk−pangocolor.xml
pygtk−pangocontext.xml
pygtk−pangofontdescription.xml
pygtk−pangofontface.xml
pygtk−pangofontfamily.xml
pygtk−pangofontmap.xml
pygtk−pangofontmetrics.xml

pygtk−pangofontset.xml
pygtk−pangofont.xml
pygtk−pangoglyphstring.xml
pygtk−pangolanguage.xml
pygtk−pangolayout.xml
pygtk−pango−markup.xml
pygtk−pangotabarray.xml

* `pygtk-pango-classes.xml` Add include for:
pygtk−pangoattribute.xml
pygtk−pangoattriter.xml
pygtk−pango−markup.xml

* `pygtk2-ref.html` Update date and version number. Add pango class reference chapter.

2003−07−10 John Finlay <finlay@moeraki.com>

* `Pango` Add pango class files:
pygtk−pangoattribute.xml
pygtk−pangoattriter.xml
pygtk−pangoattrlist.xml
pygtk−pangocolor.xml
pygtk−pangocontext.xml
pygtk−pangofontdescription.xml
pygtk−pangofontface.xml
pygtk−pangofontfamily.xml
pygtk−pangofontmap.xml
pygtk−pangofontmetrics.xml
pygtk−pangofontset.xml
pygtk−pangofont.xml
pygtk−pangoglyphstring.xml
pygtk−pangolanguage.xml
pygtk−pangolayout.xml
pygtk−pango−markup.xml
pygtk−pangotabarray.xml

* `ChangeLog` Add this change log

* `pygtk2-ref.xml` Include ChangeLog info in reference.

* `pygtk-gtkliststore.xml` (*insert*)
(*insert_before*)
(*insert_after*)
(*prepend*)
(*append*) Fix description of row arg to be a sequence of values.

* `pygtk-gtktreestore.xml` (*insert*)
(*insert_before*)
(*insert_after*)
(*prepend*)
(*append*) Fix description of row arg to be a sequence of values.

2003−07−02 John Finlay <finlay@moeraki.com>

* `pygtk2-ref.xml` Update date.

* `pygtk-gdk-functions.xml` (*gtk.gdk.threads_init*) Update description of thread support.

* `ChangeLog` Add change log.

Copyright and License Notice

Reference Page Format

Each `PyGTK` class is described in a reference page that has a number of sections in a fixed format. Each reference page will have a subset of the following sections:

Name	The name and a one–line description of the class.
Synopsis	A synopsis of the class and its methods and optionally a list of associated functions.
Ancestry	The list of the parent classes of the class. This section may not be present in all class descriptions.
Properties	A list of the properties (internal state) supported by the class. This section may not be present in all classes. The property descriptions include the name, the access operations (e.g. Read, Write), and a brief description. Properties are accessed using the `gobject.set_property()` and `gobject.get_property()` methods that are available to every `PyGTK` object. This section may not be present in all class descriptions.
Style Properties	A list of style properties supported by the class. Similar to the properties (described above) the style properties hold information about the style of a widgets e.g. border style, shadow type, etc. Most widgets do not support style properties so this section is not present in most class descriptions. Only PyGTK 2.4 has the ability to access style properties.
Child Properties	A list of child properties supported by the class. Similar to the properties (described above) the child properties hold information about the properties of a widget's child widget. Only container widgets support child properties so this section is not present in most class descriptions.
Attributes	A set of internal object state data accessible as Python attributes (e.g. object.attr). The attribute descriptions include a name by which the attribute data is accessed, the access

	mode (e.g. Read, Write), and a brief description of the attribute. Most `PyGTK` classes do not support attributes so this section is not present in most class descriptions.
Signal Prototypes	A list of the signals supported by the class including the signal name and a synopsis of the signal handler function prototype. This section may not be present in all class descriptions; most `gtk.gdk` classes do not support signals.
Description	A description of the class and possibly some of the methods supported by the class.
Constructor	The description of the class object constructor including the synopsis with brief parameter descriptions and a description of th use of the constructor. There may be more than one constructor description if the constructor supports different parameter lists. This section may not be present in all class descriptions.
Methods	A list of methods supported by the class. Each method description includes: a synopsis of the method and its parameters as well as a brief description of each parameter and return value (if any); and, a description of the use of the method.
Functions	A list of related functions. Each function description includes a synopsis of the function and its parameters and return value (if any), and a description of the use of the function.
Signals	A list of signals including a synopsis of the signal handler prototype function with its parameters and return value (if any). The signal emission conditions are briefly described. This section is not present in all class descriptions; specifically, the `gtk.gdk` classes do not usually support signals.

The function and method synopsis parameters are displayed in **bold** to denote Python keyword parameters. Also if the parameter is optional its default value will be displayed. For example the gtk.Button() constructor synopsis is:

```
    gtk.Button(label=None, stock=None)
```

The parameters *`label`* and *`stock`* are keyword parameters that can be specified in a call either by position or keyword (in which case position is not important). The following calls have the same result:

```
  b = gtk.Button("Cancel")
  b = gtk.Button(label="Cancel")
  b = gtk.Button("Cancel", None)
  b = gtk.Button("Cancel", stock=None)
  b = gtk.Button(stock=None, label="Cancel")
```

Parameters that are not keyword parameters are displayed in *italic* and must be specified positionally but may also be optional.

Prev | Up | Next
Introduction | Home | Copyright and License Notice