Guile-GNOME: Gdk

version 2.16.2, updated 9 December 2011

Damon Chaplin
many others

This manual is for `(gnome gdk)` (version 2.16.2, updated 9 December 2011)

GNOME Documentation Project, the names have been printed in caps or initial caps.

Short Contents

1 Overview 1
2 General 2
3 GdkDisplay 7
4 GdkDisplayManager 14
5 GdkScreen 15
6 Points, Rectangles and Regions 24
7 Graphics Contexts 27
8 Drawing Primitives 32
9 Bitmaps and Pixmaps 42
10 GdkRGB 44
11 Images 48
12 Pixbufs 50
13 Colormaps and Colors 54
14 Visuals 56
15 Fonts 59
16 Cursors 65
17 Windows 69
18 Events 93
19 Event Structures 96
20 Key Values 98
21 Selections 102
22 Drag and Drop 105
23 Properties and Atoms 108
24 Threads 110
25 Input Devices 115
26 Pango Interaction 118
27 Cairo Interaction 123
28 X Window System Interaction 125
29 The GdkPixbuf Structure 127
30 File Loading 130
31 Image Data in Memory 132
32 Scaling 134
33 Utilities 139
34 Animations 141
35 GdkPixbufLoader 143

36 Module Interface 147
37 Undocumented 148
Type Index 149
Function Index 150

1 Overview

`(gnome gdk)` wraps the platform-specific layer of GTK+ for Guile. It is a part of Guile-GNOME.

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

2 General

Library initialization and miscellaneous functions

2.1 Overview

This section describes the GDK initialization functions and miscellaneous utility functions.

2.2 Usage

`gdk-get-display-arg-name` ⇒ (*ret* `mchars`) [Function]

Gets the display name specified in the command line arguments passed to `gdk-init` or `gdk-parse-args`, if any.

ret the display name, if specified explicitely, otherwise '`#f`' this string is owned by GTK+ and must not be modified or freed.

Since 2.2

`gdk-set-locale` ⇒ (*ret* `mchars`) [Function]

Initializes the support for internationalization by calling the `setlocale` system call. This function is called by `gtk-set-locale` and so GTK+ applications should use that instead.

The locale to use is determined by the `LANG` environment variable, so to run an application in a certain locale you can do something like this:

```
export LANG="fr"
... run application ...
```

If the locale is not supported by X then it is reset to the standard `"C"` locale.

ret the resulting locale.

`gdk-set-sm-client-id` (*sm_client_id* `mchars`) [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.)

sm-client-id

the client id assigned by the session manager when the connection was opened, or '`#f`' to remove the property.

`gdk-notify-startup-complete` [Function]

Indicates to the GUI environment that the application has finished loading. If the applications opens windows, this function is normally called after opening the application's initial set of windows.

GTK+ will call this function automatically after opening the first `<gtk-window>` unless `gtk-window-set-auto-startup-notification` is called to disable that feature.

Since 2.2

`gdk-get-program-class` ⇒ (*ret* `mchars`) [Function]

Gets the program class. Unless the program class has explicitly been set with `gdk-set-program-class` or with the option, the default value is the program name (determined with `g-get-prgname`) with the first character converted to uppercase.

ret the program class.

`gdk-set-program-class` (*program_class* `mchars`) [Function]

Sets the program class. The X11 backend uses the program class to set the class name part of the '`WM_CLASS`' property on toplevel windows; see the ICCCM.

program-class
a string.

`gdk-get-display` ⇒ (*ret* `mchars`) [Function]

Gets the name of the display, which usually comes from the `DISPLAY` environment variable or the

ret the name of the display.

`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. It's main use is for trapping X errors with `gdk-error-trap-push` and `gdk-error-trap-pop`.

`gdk-screen-width` ⇒ (*ret* `int`) [Function]

Returns the width of the default screen in pixels.

ret the width of the default screen in pixels.

`gdk-screen-height` ⇒ (*ret* `int`) [Function]

Returns the height of the default screen in pixels.

ret the height of the default screen in pixels.

`gdk-screen-width-mm` ⇒ (*ret* `int`) [Function]

Returns the width of the default screen in millimeters. Note that on many X servers this value will not be correct.

ret the width of the default screen in millimeters, though it is not always correct.

`gdk-screen-height-mm` ⇒ (*ret* `int`) [Function]

Returns the height of the default screen in millimeters. Note that on many X servers this value will not be correct.

ret the height of the default screen in millimeters, though it is not always correct.

gdk-pointer-grab (*window* <gdk-window>) (*owner_events* bool) (*event_mask* <gdk-event-mask>) (*confine_to* <gdk-window>) (*cursor* <gdk-cursor>) (*time_* unsigned-int32) ⇒ (*ret* <gdk-grab-status>) [Function]

Grabs the pointer (usually a mouse) so that all events are passed to this application until the pointer is ungrabbed with `gdk-pointer-ungrab`, or the grab window becomes unviewable. This overrides any previous pointer grab by this client.

Pointer grabs are used for operations which need complete control over mouse events, even if the mouse leaves the application. For example in GTK+ it is used for Drag and Drop, for dragging the handle in the `<gtk-hpaned>` and `<gtk-vpaned>` widgets, and for resizing columns in `<gtk-clist>` 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 '#t'.

If you set up anything at the time you take the grab that needs to be cleaned up when the grab ends, you should handle the `<gdk-event-grab-broken>` events that are emitted when the grab ends unvoluntarily.

window the `<gdk-window>` which will own the grab (the grab window).

owner-events
: if '#f' then all pointer events are reported with respect to *window* and are only reported if selected by *event-mask*. If '#t' 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, which is used in accordance with *owner-events*. Note that only pointer events (i.e. button and motion events) may be selected.

confine-to If non-'#f', the pointer will be confined to this 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 cursor to display while the grab is active. If this is '#f' 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 which led to this pointer grab. This usually comes from a `<gdk-event-button>` struct, though 'GDK_CURRENT_TIME' can be used if the time isn't known.

ret 'GDK_GRAB_SUCCESS' if the grab was successful.

gdk-pointer-ungrab (*time_* unsigned-int32) [Function]

Ungrabs the pointer, if it is grabbed by this application.

time a timestamp from a `<gdk-event>`, or '`GDK_CURRENT_TIME`' if no timestamp is available.

gdk-pointer-is-grabbed ⇒ (*ret* bool) [Function]

Returns '`#t`' if the pointer is currently grabbed by this application.

Note that this does not take the inmplicit pointer grab on button presses into account.

ret '`#t`' if the pointer is currently grabbed by this application.*

gdk-set-double-click-time (*msec* unsigned-int) [Function]

Set the double click time for the default display. See `gdk-display-set-double-click-time`. See also `gdk-display-set-double-click-distance`. Applications should *not* set this, it is a global user-configured setting.

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

gdk-keyboard-grab (*window* <gdk-window>) (*owner_events* bool) (*time_* unsigned-int32) ⇒ (*ret* <gdk-grab-status>) [Function]

Grabs the keyboard so that all events are passed to this application until the keyboard is ungrabbed with `gdk-keyboard-ungrab`. This overrides any previous keyboard grab by this client.

If you set up anything at the time you take the grab that needs to be cleaned up when the grab ends, you should handle the `<gdk-event-grab-broken>` events that are emitted when the grab ends unvoluntarily.

window the `<gdk-window>` which will own the grab (the grab window).

owner-events
if '`#f`' then all keyboard events are reported with respect to *window*. If '`#t`' 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, independant of the event mask set by the application.

time a timestamp from a `<gdk-event>`, or '`GDK_CURRENT_TIME`' if no timestamp is available.

ret '`GDK_GRAB_SUCCESS`' if the grab was successful.

gdk-keyboard-ungrab (*time_* unsigned-int32) [Function]

Ungrabs the keyboard, if it is grabbed by this application.

time a timestamp from a `<gdk-event>`, or '`GDK_CURRENT_TIME`' if no timestamp is available.

gdk-beep [Function]

Emits a short beep on the default display.

gdk-get-use-xshm ⇒ (*ret* bool) [Function]

'`gdk_get_use_xshm`' is deprecated and should not be used in newly-written code.

Returns '`#t`' if GDK will attempt to use the MIT-SHM shared memory extension.

The shared memory extension is used for `<gdk-image>`, and consequently for GdkRGB. It enables much faster drawing by communicating with the X server through SYSV shared memory calls. However, it can only be used if the X client and server are on the same machine and the server supports it.

ret '`#t`' if use of the MIT shared memory extension will be attempted.

gdk-set-use-xshm (*use_xshm* `bool`) [Function]

'`gdk_set_use_xshm`' is deprecated and should not be used in newly-written code.

Sets whether the use of the MIT shared memory extension should be attempted. This function is mainly for internal use. It is only safe for an application to set this to '`#f`', since if it is set to '`#t`' and the server does not support the extension it may cause warning messages to be output.

use-xshm '`#t`' if use of the MIT shared memory extension should be attempted.

gdk-error-trap-push [Function]

This function allows X errors to be trapped instead of the normal behavior of exiting the application. It should only be used if it is not possible to avoid the X error in any other way.

```
gdk_error_trap_push ();

/* ... Call the X function which may cause an error here ... */

/* Flush the X queue to catch errors now. */
gdk_flush ();

if (gdk_error_trap_pop ())
  {
    /* ... Handle the error here ... */
  }
```

gdk-error-trap-pop ⇒ (*ret* `int`) [Function]

Removes the X error trap installed with `gdk-error-trap-push`.

ret the X error code, or 0 if no error occurred.

3 GdkDisplay

Controls the keyboard/mouse pointer grabs and a set of s

3.1 Overview

`<gdk-display>` objects purpose are two fold:

To grab/ungrab keyboard focus and mouse pointer

To manage and provide information about the `<gdk-screen>`(s) available for this `<gdk-display>`

`<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 `<gdk-screen>` objects currently instanciated by the application. It is also used to grab and release the keyboard and the mouse pointer.

3.2 Usage

`<gdk-display>` [Class]

Derives from `<gobject>`.

This class defines no direct slots.

`closed` (*arg0* `<gboolean>`) [Signal on `<gdk-display>`]

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

Since 2.2

`gdk-display-open` (*display_name* `mchars`) ⇒ (*ret* `<gdk-display>`) [Function]

Opens a display.

display-name
: the name of the display to open

ret a `<gdk-display>`, or '`#f`' if the display could not be opened.

Since 2.2

`gdk-display-get-default` ⇒ (*ret* `<gdk-display>`) [Function]

Gets the default `<gdk-display>`. This is a convenience function for:

```
gdk_display_manager_get_default_display (gdk_display_manager_get ())
```

ret a `<gdk-display>`, or '`#f`' if there is no default display.

Since 2.2

`gdk-display-get-name` (*self* `<gdk-display>`) ⇒ (*ret* `mchars`) [Function]
`get-name` [Method]

Gets the name of the display.

display a `<gdk-display>`

ret a string representing the display name. This string is owned by GDK and should not be modified or freed.

Since 2.2

`gdk-display-get-n-screens` (*self* `<gdk-display>`) ⇒ (*ret* `int`) [Function]
`get-n-screens` [Method]

Gets the number of screen managed by the *display*.

display a `<gdk-display>`

ret number of screens.

Since 2.2

`gdk-display-get-screen` (*self* `<gdk-display>`) (*screen_num* `int`) ⇒ (*ret* `<gdk-screen>`) [Function]
`get-screen` [Method]

Returns a screen object for one of the screens of the display.

display a `<gdk-display>`

screen-num the screen number

ret the `<gdk-screen>` object

Since 2.2

`gdk-display-get-default-screen` (*self* `<gdk-display>`) ⇒ (*ret* `<gdk-screen>`) [Function]
`get-default-screen` [Method]

Get the default `<gdk-screen>` for *display*.

display a `<gdk-display>`

ret the default `<gdk-screen>` object for *display*

Since 2.2

`gdk-display-pointer-ungrab` (*self* `<gdk-display>`) (*time_* `unsigned-int32`) [Function]
`pointer-ungrab` [Method]

Release any pointer grab.

display a `<gdk-display>`.

time a timestap (e.g. '`GDK_CURRENT_TIME`').

Since 2.2

`gdk-display-keyboard-ungrab` (*self* `<gdk-display>`) (*time_* `unsigned-int32`) [Function]
`keyboard-ungrab` [Method]

Release any keyboard grab

display a `<gdk-display>`.

time a timestap (e.g `<gdk-current-time>`).

Since 2.2

`gdk-display-pointer-is-grabbed` (*self* `<gdk-display>`) [Function]
⇒ (*ret* `bool`)
`pointer-is-grabbed` [Method]

Test if the pointer is grabbed.

display a `<gdk-display>`

ret '`#t`' if an active X pointer grab is in effect

Since 2.2

`gdk-display-beep` (*self* `<gdk-display>`) [Function]
`beep` [Method]

Emits a short beep on *display*

display a `<gdk-display>`

Since 2.2

`gdk-display-sync` (*self* `<gdk-display>`) [Function]
`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. Calling `gdk-display-sync` before `gdk-error-trap-pop` makes sure that any errors generated from earlier requests are handled before the error trap is removed.

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

display a `<gdk-display>`

Since 2.2

`gdk-display-flush` (*self* `<gdk-display>`) [Function]
`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 function explicitely. A common case where this function 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 function will do nothing.

display a `<gdk-display>`

Since 2.4

`gdk-display-close` (*self* `<gdk-display>`) [Function]
`close` [Method]

Closes the connection to the windowing system for the given display, and cleans up associated resources.

display a `<gdk-display>`

Since 2.2

`gdk-display-list-devices` (*self* `<gdk-display>`) ⇒ (*ret* `glist-of`) [Function]
`list-devices` [Method]

Returns the list of available input devices attached to *display*. The list is statically allocated and should not be freed.

display a `<gdk-display>`

ret a list of `<gdk-device>`

Since 2.2

`gdk-display-get-event` (*self* `<gdk-display>`) ⇒ (*ret* `<gdk-event>`) [Function]
`get-event` [Method]

Gets the next `<gdk-event>` to be processed for *display*, fetching events from the windowing system if necessary.

display a `<gdk-display>`

ret the next `<gdk-event>` to be processed, or '`#f`' if no events are pending. The returned `<gdk-event>` should be freed with `gdk-event-free`.

Since 2.2

`gdk-display-peek-event` (*self* `<gdk-display>`) [Function]
⇒ (*ret* `<gdk-event>`)
`peek-event` [Method]

Gets a copy of the first `<gdk-event>` in the *display*'s event queue, without removing the event from the queue. (Note that this function will not get more events from the windowing system. It only checks the events that have already been moved to the GDK event queue.)

display a `<gdk-display>`

ret a copy of the first `<gdk-event>` on the event queue, or '`#f`' if no events are in the queue. The returned `<gdk-event>` should be freed with `gdk-event-free`.

Since 2.2

`gdk-display-put-event` (*self* `<gdk-display>`) (*event* `<gdk-event>`) [Function]
`put-event` [Method]

Appends a copy of the given event onto the front of the event queue for *display*.

display a `<gdk-display>`

event a `<gdk-event>`.

Since 2.2

gdk-display-set-double-click-time (*self* <gdk-display>) (*msec* unsigned-int) [Function]
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 <gdk-2button-press> event). Applications should *not* set this, it is a global user-configured setting.

display a <gdk-display>

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

Since 2.2

gdk-display-get-window-at-pointer (*self* <gdk-display>) ⇒ (*ret* <gdk-window>) (*win_x* int) (*win_y* int) [Function]
get-window-at-pointer [Method]

Obtains the window underneath the mouse pointer, returning the location of that window in *win-x*, *win-y* for *screen*. Returns '#f' if the window under the mouse pointer is not known to GDK (for example, belongs to another application).

display a <gdk-display>

win-x return location for origin of the window under the pointer

win-y return location for origin of the window under the pointer

ret the window under the mouse pointer, or '#f'

Since 2.2

gdk-display-warp-pointer (*self* <gdk-display>) (*screen* <gdk-screen>) (*x* int) (*y* int) [Function]
warp-pointer [Method]

Warps the pointer of *display* to the point *x*,*y* on the screen *screen*, unless the pointer is confined to a window by a grab, in which case it will be moved as far as allowed by the grab. Warping the pointer creates events as if the user had moved the mouse instantaneously to the destination.

Note that the pointer should normally be under the control of the user. This function was added to cover some rare use cases like keyboard navigation support for the color picker in the <gtk-color-selection-dialog>.

display a <gdk-display>

screen the screen of *display* to warp the pointer to

x the x coordinate of the destination

y the y coordinate of the destination

Since 2.8

gdk-display-supports-cursor-color (*self* <gdk-display>) ⇒ (*ret* bool) [Function]
supports-cursor-color [Method]

Returns '#t' if multicolored cursors are supported on *display*. Otherwise, cursors have only a forground and a background color.

display a <gdk-display>

ret whether cursors can have multiple colors.

Since 2.4

`gdk-display-supports-cursor-alpha` (*self* `<gdk-display>`) [Function]
⇒ (*ret* `bool`)

`supports-cursor-alpha` [Method]

Returns '`#t`' if cursors can use an 8bit alpha channel on *display*. Otherwise, cursors are restricted to bilevel alpha (i.e. a mask).

display a <gdk-display>

ret whether cursors can have alpha channels.

Since 2.4

`gdk-display-get-default-cursor-size` (*self* `<gdk-display>`) [Function]
⇒ (*ret* `unsigned-int`)

`get-default-cursor-size` [Method]

Returns the default size to use for cursors on *display*.

display a <gdk-display>

ret the default cursor size.

Since 2.4

`gdk-display-get-maximal-cursor-size` (*self* `<gdk-display>`) [Function]
⇒ (*width* `unsigned-int`) (*height* `unsigned-int`)

`get-maximal-cursor-size` [Method]

Gets the maximal size to use for cursors on *display*.

display a <gdk-display>

width the return location for the maximal cursor width

height the return location for the maximal cursor height

Since 2.4

`gdk-display-get-default-group` (*self* `<gdk-display>`) [Function]
⇒ (*ret* `<gdk-window>`)

`get-default-group` [Method]

Returns the default group leader window for all toplevel windows on *display*. This window is implicitly created by GDK. See `gdk-window-set-group`.

display a <gdk-display>

ret The default group leader window for *display*

Since 2.4

`gdk-display-supports-shapes` (*self* `<gdk-display>`) ⇒ (*ret* `bool`) [Function]

`supports-shapes` [Method]

Returns '`#t`' if `gdk-window-shape-combine-mask` can be used to create shaped windows on *display*.

display a `<gdk-display>`

ret '`#t`' if shaped windows are supported

Since 2.10

`gdk-display-supports-input-shapes` (*self* `<gdk-display>`) [Function]
⇒ (*ret* `bool`)

`supports-input-shapes` [Method]

Returns '`#t`' if `gdk-window-input-shape-combine-mask` can be used to modify the input shape of windows on *display*.

display a `<gdk-display>`

ret '`#t`' if windows with modified input shape are supported

Since 2.10

4 GdkDisplayManager

Maintains a list of all open s

4.1 Overview

The purpose of the `<gdk-display-manager>` singleton object is to offer notification when displays appear or disappear or the default display changes.

4.2 Usage

`<gdk-display-manager>` [Class]

Derives from `<gobject>`.

This class defines the following slots:

`default-display`
: The default display for GDK

`display-opened` (*arg0* `<gdk-display>`) [Signal on `<gdk-display-manager>`]

The ::display_opened signal is emitted when a display is opened.

Since 2.2

`gdk-display-manager-get` ⇒ (*ret* `<gdk-display-manager>`) [Function]

Returns the global `<gdk-display-manager>` singleton; `gdk-parse-pargs`, `gdk-init`, or `gdk-init-check` must have been called first.

ret the singleton `<gdk-display-manager>` object.

Since 2.2

`gdk-display-manager-list-displays` [Function]
(*self* `<gdk-display-manager>`) ⇒ (*ret* `gslist-of`)

`list-displays` [Method]

List all currently open displays.

display-manager
: a `<gdk-display-manager>`

ret a newly allocated `<gs-list>` of `<gdk-display>` objects. Free this list with `g-slist-free` when you are done with it.

Since 2.2

`gdk-display-get-core-pointer` (*self* `<gdk-display>`) [Function]
⇒ (*ret* `<gdk-device>`)

`get-core-pointer` [Method]

Returns the core pointer device for the given display

display a `<gdk-display>`

ret the core pointer device; this is owned by the display and should not be freed.

Since 2.2

5 GdkScreen

Object representing a physical screen

5.1 Overview

`<gdk-screen>` objects are the GDK representation of a physical screen. It is used throughout GDK and GTK+ 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 (`gdk-screen-get-default-colormap`), the screen width (`gdk-screen-get-width`), etc.

Note that a screen may consist of multiple monitors which are merged to form a large screen area.

5.2 Usage

`<gdk-screen>` [Class]

Derives from `<gobject>`.

This class defines the following slots:

`font-options`
: The default font options for the screen

`resolution`
: The resolution for fonts on the screen

`size-changed` [Signal on `<gdk-screen>`]

The ::size_changed signal is emitted when the pixel width or height of a screen changes.

Since 2.2

`composited-changed` [Signal on `<gdk-screen>`]

The ::composited_changed signal is emitted when the composited status of the screen changes

Since 2.10

`gdk-screen-get-default` ⇒ (*ret* `<gdk-screen>`) [Function]

Gets the default screen for the default display. (See `gdk-display-get-default`).

ret a `<gdk-screen>`, or '`#f`' if there is no default display.

Since 2.2

`gdk-screen-get-default-colormap` (*self* `<gdk-screen>`) [Function]
⇒ (*ret* `<gdk-colormap>`)

`get-default-colormap` [Method]

Gets the default colormap for *screen*.

screen a `<gdk-screen>`

ret the default `<gdk-colormap>`.

Since 2.2

`gdk-screen-set-default-colormap` (*self* `<gdk-screen>`) [Function]
(*colormap* `<gdk-colormap>`)

`set-default-colormap` [Method]

Sets the default *colormap* for *screen*.

screen a `<gdk-screen>`

colormap a `<gdk-colormap>`

Since 2.2

`gdk-screen-get-system-colormap` (*self* `<gdk-screen>`) [Function]
⇒ (*ret* `<gdk-colormap>`)

`get-system-colormap` [Method]

Gets the system's default colormap for *screen*

screen a `<gdk-screen>`

ret the default colormap for *screen*.

Since 2.2

`gdk-screen-get-system-visual` (*self* `<gdk-screen>`) [Function]
⇒ (*ret* `<gdk-visual>`)

`get-system-visual` [Method]

Get the system's default visual for *screen*. This is the visual for the root window of the display. The return value should not be freed.

screen a `<gdk-screen>`.

ret the system visual

Since 2.2

`gdk-screen-get-rgb-colormap` (*self* `<gdk-screen>`) [Function]
⇒ (*ret* `<gdk-colormap>`)

`get-rgb-colormap` [Method]

Gets the preferred colormap for rendering image data on *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.

screen a `<gdk-screen>`.

ret the preferred colormap

Since 2.2

`gdk-screen-get-rgb-visual` (*self* `<gdk-screen>`) [Function]
⇒ (*ret* `<gdk-visual>`)

`get-rgb-visual` [Method]

Gets a "preferred visual" chosen by GdkRGB for rendering image data on *screen*. In previous versions of GDK, this was the only visual GdkRGB could use for rendering. In current versions, it's simply the visual GdkRGB would have chosen as the optimal one in those previous versions. GdkRGB can now render to drawables with any visual.

screen a `<gdk-screen>`

ret The `<gdk-visual>` chosen by GdkRGB.

Since 2.2

`gdk-screen-get-rgba-colormap` (*self* `<gdk-screen>`) [Function]
⇒ (*ret* `<gdk-colormap>`)

`get-rgba-colormap` [Method]

Gets a colormap to use for creating windows or pixmaps with an alpha channel. The windowing system on which GTK+ is running may not support this capability, in which case '`#f`' will be returned. Even if a non-'`#f`' value is returned, its possible that the window's alpha channel won't be honored when displaying the window on the screen: in particular, for X an appropriate windowing manager and compositing manager must be running to provide appropriate display.

screen a `<gdk-screen>`.

ret a colormap to use for windows with an alpha channel or '`#f`' if the capability is not available.

Since 2.8

`gdk-screen-get-rgba-visual` (*self* `<gdk-screen>`) [Function]
⇒ (*ret* `<gdk-visual>`)

`get-rgba-visual` [Method]

Gets a visual to use for creating windows or pixmaps with an alpha channel. See the docs for `gdk-screen-get-rgba-colormap` for caveats.

screen a `<gdk-screen>`

ret a visual to use for windows with an alpha channel or '`#f`' if the capability is not available.

Since 2.8

`gdk-screen-is-composited` (*self* `<gdk-screen>`) ⇒ (*ret* `bool`) [Function]

`is-composited` [Method]

Returns whether windows with an RGBA visual can reasonably be expected to have their alpha channel drawn correctly on the screen.

On X11 this function returns whether a compositing manager is compositing *screen*.

screen a `<gdk-screen>`

ret Whether windows with RGBA visuals can reasonably be expected to have their alpha channels drawn correctly on the screen.

Since 2.10

`gdk-screen-get-root-window` (*self* `<gdk-screen>`) [Function]
⇒ (*ret* `<gdk-window>`)

`get-root-window` [Method]

Gets the root window of *screen*.

screen a `<gdk-screen>`

ret the root window

Since 2.2

`gdk-screen-get-display` (*self* `<gdk-screen>`) [Function]
⇒ (*ret* `<gdk-display>`)

`get-display` [Method]

Gets the display to which the *screen* belongs.

screen a `<gdk-screen>`

ret the display to which *screen* belongs

Since 2.2

`gdk-screen-get-number` (*self* `<gdk-screen>`) ⇒ (*ret* `int`) [Function]

`get-number` [Method]

Gets the index of *screen* among the screens in the display to which it belongs. (See `gdk-screen-get-display`)

screen a `<gdk-screen>`

ret the index

Since 2.2

`gdk-screen-get-width` (*self* `<gdk-screen>`) ⇒ (*ret* `int`) [Function]

`get-width` [Method]

Gets the width of *screen* in pixels

screen a `<gdk-screen>`

ret the width of *screen* in pixels.

Since 2.2

`gdk-screen-get-height` (*self* `<gdk-screen>`) ⇒ (*ret* `int`) [Function]

`get-height` [Method]

Gets the height of *screen* in pixels

screen a `<gdk-screen>`

ret the height of *screen* in pixels.

Since 2.2

`gdk-screen-get-width-mm` (*self* `<gdk-screen>`) ⇒ (*ret* `int`) [Function]

`get-width-mm` [Method]

Gets the width of *screen* in millimeters. Note that on some X servers this value will not be correct.

screen a `<gdk-screen>`

ret the width of *screen* in millimeters.

Since 2.2

`gdk-screen-get-height-mm` (*self* `<gdk-screen>`) ⇒ (*ret* `int`) [Function]
`get-height-mm` [Method]

Returns the height of *screen* in millimeters. Note that on some X servers this value will not be correct.

screen a `<gdk-screen>`

ret the heigth of *screen* in millimeters.

Since 2.2

`gdk-screen-list-visuals` (*self* `<gdk-screen>`) ⇒ (*ret* `glist-of`) [Function]
`list-visuals` [Method]

Lists the available visuals for the specified *screen*. A 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.

Call `g-list-free` on the return value when you're finished with it.

screen the relevant `<gdk-screen>`.

ret a list of visuals; the list must be freed, but not its contents

Since 2.2

`gdk-screen-get-toplevel-windows` (*self* `<gdk-screen>`) [Function]
⇒ (*ret* `glist-of`)
`get-toplevel-windows` [Method]

Obtains a list of all toplevel windows known to GDK on the screen *screen*. A toplevel window is a child of the root window (see `gdk-get-default-root-window`).

The returned list should be freed with `g-list-free`, but its elements need not be freed.

screen The `<gdk-screen>` where the toplevels are located.

ret list of toplevel windows, free with `g-list-free`

Since 2.2

`gdk-screen-make-display-name` (*self* `<gdk-screen>`) [Function]
⇒ (*ret* `mchars`)
`make-display-name` [Method]

Determines the name to pass to `gdk-display-open` to get a `<gdk-display>` with this screen as the default screen.

screen a `<gdk-screen>`

ret a newly allocated string, free with `g-free`

Since 2.2

`gdk-screen-get-n-monitors` (*self* `<gdk-screen>`) ⇒ (*ret* `int`) [Function]
`get-n-monitors` [Method]

Returns the number of monitors which *screen* consists of.

screen a `<gdk-screen>`.

ret number of monitors which *screen* consists of.

Since 2.2

`gdk-screen-get-monitor-geometry` (*self* `<gdk-screen>`) (*monitor_num* `int`) (*dest* `<gdk-rectangle>`) [Function]
`get-monitor-geometry` [Method]

Retrieves the `<gdk-rectangle>` representing the size and position of the individual monitor within the entire screen area.

Note that the size of the entire screen area can be retrieved via `gdk-screen-get-width` and `gdk-screen-get-height`.

screen a `<gdk-screen>`.

monitor-num
the monitor number.

dest a `<gdk-rectangle>` to be filled with the monitor geometry

Since 2.2

`gdk-screen-get-monitor-at-point` (*self* `<gdk-screen>`) (*x* `int`) (*y* `int`) ⇒ (*ret* `int`) [Function]
`get-monitor-at-point` [Method]

Returns the monitor number in which the point (*x*,*y*) is located.

screen a `<gdk-screen>`.

x the x coordinate in the virtual screen.

y the y coordinate in the virtual screen.

ret the monitor number in which the point (*x*,*y*) lies, or a monitor close to (*x*,*y*) if the point is not in any monitor.

Since 2.2

`gdk-screen-get-monitor-at-window` (*self* `<gdk-screen>`) (*window* `<gdk-window>`) ⇒ (*ret* `int`) [Function]
`get-monitor-at-window` [Method]

Returns the number of the monitor in which the largest area of the bounding rectangle of *window* resides.

screen a `<gdk-screen>`.

window a `<gdk-window>`

ret the monitor number in which most of *window* is located, or if *window* does not intersect any monitors, a monitor, close to *window*.

Since 2.2

`gdk-screen-broadcast-client-message` (*self* `<gdk-screen>`) (*event* `<gdk-event>`) [Function]
`broadcast-client-message` [Method]

On X11, sends an X ClientMessage event to all toplevel windows on *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.

screen the `<gdk-screen>` where the event will be broadcasted.

event the `<gdk-event>`.

Since 2.2

gdk-screen-get-setting (*self* `<gdk-screen>`) (*name* `mchars`) (*value* `<gvalue>`) ⇒ (*ret* `bool`) [Function]

get-setting [Method]

Retrieves a desktop-wide setting such as double-click time for the `<gdk-screen>`*screen*.

FIXME needs a list of valid settings here, or a link to more information.

screen the `<gdk-screen>` where the setting is located

name the name of the setting

value location to store the value of the setting

ret '`#t`' if the setting existed and a value was stored in *value*, '`#f`' otherwise.

Since 2.2

gdk-screen-get-font-options (*self* `<gdk-screen>`) ⇒ (*ret* `cairo-font-options-t`) [Function]

get-font-options [Method]

Gets any options previously set with `gdk-screen-set-font-options`.

screen a `<gdk-screen>`

ret the current font options, or '`#f`' if no default font options have been set.

Since 2.10

gdk-screen-set-font-options (*self* `<gdk-screen>`) (*options* `cairo-font-options-t`) [Function]

set-font-options [Method]

Sets the default font options for the screen. These options will be set on any `<pango-context>`'s newly created with `gdk-pango-context-get-for-screen`. Changing the default set of font options does not affect contexts that have already been created.

screen a `<gdk-screen>`

options a `<cairo-font-options-t>`, or '`#f`' to unset any previously set default font options.

Since 2.10

`gdk-screen-get-resolution` (*self* `<gdk-screen>`) ⇒ (*ret* `double`) [Function]
`get-resolution` [Method]

Gets the resolution for font handling on the screen; see `gdk-screen-set-resolution` for full details.

screen a `<gdk-screen>`

ret the current resolution, or -1 if no resolution has been set.

Since 2.10

`gdk-screen-set-resolution` (*self* `<gdk-screen>`) (*dpi* `double`) [Function]
`set-resolution` [Method]

Sets the resolution for font handling on the screen. This is a scale factor between points specified in a `<pango-font-description>` and cairo units. The default value is 96, meaning that a 10 point font will be 13 units high. (10 * 96. / 72. = 13.3).

screen a `<gdk-screen>`

dpi the resolution in "dots per inch". (Physical inches aren't actually involved; the terminology is conventional.)

Since 2.10

`gdk-screen-get-active-window` (*self* `<gdk-screen>`) [Function]
⇒ (*ret* `<gdk-window>`)
`get-active-window` [Method]

Returns the screen's currently active window.

On X11, this is done by inspecting the _NET_ACTIVE_WINDOW property on the root window, as described in the Extended Window Manager Hints (`http://www.freedesktop.org/Standards/wm-spec`). If there is no currently currently active window, or the window manager does not support the _NET_ACTIVE_WINDOW hint, this function returns '`#f`'.

On other platforms, this function may return '`#f`', depending on whether it is implementable on that platform.

The returned window should be unrefed using `g-object-unref` when no longer needed.

screen a `<gdk-screen>`

ret the currently active window, or '`#f`'.

Since 2.10

`gdk-screen-get-window-stack` (*self* `<gdk-screen>`) [Function]
⇒ (*ret* `glist-of`)
`get-window-stack` [Method]

Returns a `<g-list>` of `<gdk-window>`s representing the current window stack.

On X11, this is done by inspecting the _NET_CLIENT_LIST_STACKING property on the root window, as described in the Extended Window Manager Hints (`http://www.freedesktop.org/Standards/wm-spec`). If the window manager does not support the _NET_CLIENT_LIST_STACKING hint, this function returns '`#f`'.

On other platforms, this function may return '#f', depending on whether it is implementable on that platform.

The returned list is newly allocated and owns references to the windows it contains, so it should be freed using `g-list-free` and its windows unrefed using `g-object-unref` when no longer needed.

screen a `<gdk-screen>`

ret a list of `<gdk-window>`s for the current window stack, or '#f'.

Since 2.10

`gdk-spawn-command-line-on-screen` (*screen* `<gdk-screen>`) (*command_line* `mchars`) ⇒ (*ret* `bool`) [Function]

Like `g-spawn-command-line-async`, except the child process is spawned in such an environment that on calling `gdk-display-open` it would be returned a `<gdk-display>` with *screen* as the default screen.

This is useful for applications which wish to launch an application on a specific screen.

screen a `<gdk-screen>`

command-line
a command line

error return location for errors

ret '#t' on success, '#f' if error is set.

Since 2.4

6 Points, Rectangles and Regions

Simple graphical data types

6.1 Overview

GDK provides the `<gdk-point>`, `<gdk-rectangle>`, `<gdk-region>` and `<gdk-span>` data types for representing pixels and sets of pixels on the screen.

`<gdk-point>` is a simple structure containing an x and y coordinate of a point.

`<gdk-rectangle>` is a structure holding the position and size of a rectangle. The intersection of two rectangles can be computed with `gdk-rectangle-intersect`. To find the union of two rectangles use `gdk-rectangle-union`.

`<gdk-region>` is an opaque data type holding a set of arbitrary pixels, and is usually used for clipping graphical operations (see `gdk-gc-set-clip-region`).

`<gdk-span>` is a structure holding a spanline. A spanline is a horizontal line that is one pixel wide. It is mainly used when rasterizing other graphics primitives. It can be intersected to regions by using `gdk-region-spans-intersect-foreach`.

6.2 Usage

`<gdk-rectangle>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`<gdk-region>` [Class]

Opaque pointer.

This class defines no direct slots.

`gdk-region-new` ⇒ (*ret* `<gdk-region>`) [Function]

Creates a new empty `<gdk-region>`.

ret a new empty `<gdk-region>`

`gdk-region-copy` (*self* `<gdk-region>`) ⇒ (*ret* `<gdk-region>`) [Function]

Copies *region*, creating an identical new region.

region a `<gdk-region>`

ret a new region identical to *region*

`gdk-region-rectangle` (*rectangle* `<gdk-rectangle>`) ⇒ (*ret* `<gdk-region>`) [Function]

Creates a new region containing the area *rectangle*.

rectangle a `<gdk-rectangle>`

ret a new region

`gdk-region-destroy` (*self* `<gdk-region>`) [Function]

Destroys a `<gdk-region>`.

region a `<gdk-region>`

`gdk-region-empty` (*self* `<gdk-region>`) ⇒ (*ret* `bool`) [Function]

Finds out if the `<gdk-region>` is empty.

region a `<gdk-region>`

ret '`#t`' if *region* is empty.

`gdk-region-equal` (*self* `<gdk-region>`) (*region2* `<gdk-region>`) [Function]
⇒ (*ret* `bool`)

Finds out if the two regions are the same.

region1 a `<gdk-region>`

region2 a `<gdk-region>`

ret '`#t`' if *region1* and *region2* are equal.

`gdk-region-point-in` (*self* `<gdk-region>`) (*x* `int`) (*y* `int`) [Function]
⇒ (*ret* `bool`)

Finds out if a point is in a region.

region a `<gdk-region>`

x the x coordinate of a point

y the y coordinate of a point

ret '`#t`' if the point is in *region*.

`gdk-region-rect-in` (*self* `<gdk-region>`) [Function]
(*rectangle* `<gdk-rectangle>`) ⇒ (*ret* `<gdk-overlap-type>`)

Tests whether a rectangle is within a region.

region a `<gdk-region>`.

rectangle a `<gdk-rectangle>`.

ret '`GDK_OVERLAP_RECTANGLE_IN`', '`GDK_OVERLAP_RECTANGLE_OUT`', or '`GDK_OVERLAP_RECTANGLE_PART`', depending on whether the rectangle is inside, outside, or partly inside the `<gdk-region>`, respectively.

`gdk-region-offset` (*self* `<gdk-region>`) (*dx* `int`) (*dy* `int`) [Function]

Moves a region the specified distance.

region a `<gdk-region>`

dx the distance to move the region horizontally

dy the distance to move the region vertically

`gdk-region-shrink` (*self* `<gdk-region>`) (*dx* `int`) (*dy* `int`) [Function]

Resizes a region by the specified amount. Positive values shrink the region. Negative values expand it.

region a `<gdk-region>`

dx the number of pixels to shrink the region horizontally

dy the number of pixels to shrink the region vertically `gdk-region-union-with-rect` (*self* `<gdk-region>`) (*rect* `<gdk-rectangle>`) [Function]

Sets the area of *region* to the union of the areas of *region* and *rect*. The resulting area is the set of pixels contained in either *region* or *rect*.

region a `<gdk-region>`.

rect a `<gdk-rectangle>`.

`gdk-region-intersect` (*self* `<gdk-region>`) (*source2* `<gdk-region>`) [Function]

Sets the area of *source1* to the intersection of the areas of *source1* and *source2*. The resulting area is the set of pixels contained in both *source1* and *source2*.

source1 a `<gdk-region>`

source2 another `<gdk-region>`

`gdk-region-union` (*self* `<gdk-region>`) (*source2* `<gdk-region>`) [Function]

Sets the area of *source1* to the union of the areas of *source1* and *source2*. The resulting area is the set of pixels contained in either *source1* or *source2*.

source1 a `<gdk-region>`

source2 a `<gdk-region>`

`gdk-region-subtract` (*self* `<gdk-region>`) (*source2* `<gdk-region>`) [Function]

Subtracts the area of *source2* from the area *source1*. The resulting area is the set of pixels contained in *source1* but not in *source2*.

source1 a `<gdk-region>`

source2 another `<gdk-region>`

`gdk-region-xor` (*self* `<gdk-region>`) (*source2* `<gdk-region>`) [Function]

Sets the area of *source1* to the exclusive-OR of the areas of *source1* and *source2*. The resulting area is the set of pixels contained in one or the other of the two sources but not in both.

source1 a `<gdk-region>`

source2 another `<gdk-region>`

7 Graphics Contexts

Objects to encapsulate drawing properties

7.1 Overview

All drawing operations in GDK take a *graphics context* (GC) argument. 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 `gdk-gc-new-with-values`, or can be set one-by-one using functions such as `gdk-gc-set-foreground`. A few of the values in the GC, such as the dash pattern, can only be set by the latter method.

7.2 Usage

`<gdk-gc>` [Class]

Derives from `<gobject>`.

This class defines no direct slots.

`gdk-gc-new` (*drawable* `<gdk-drawable>`) ⇒ (*ret* `<gdk-gc>`) [Function]

Create a new graphics context with default values.

drawable a `<gdk-drawable>`. The created GC must always be used with drawables of the same depth as this one.

ret the new graphics context.

`gdk-gc-get-screen` (*self* `<gdk-gc>`) ⇒ (*ret* `<gdk-screen>`) [Function]
`get-screen` [Method]

Gets the `<gdk-screen>` for which *gc* was created

gc a `<gdk-gc>`.

ret the `<gdk-screen>` for *gc*.

Since 2.2

`gdk-gc-set-foreground` (*self* `<gdk-gc>`) (*color* `<gdk-color>`) [Function]
`set-foreground` [Method]

Sets the foreground color for a graphics context. Note that this function uses *color->pixel*, use `gdk-gc-set-rgb-fg-color` to specify the foreground color as red, green, blue components.

gc a `<gdk-gc>`.

color the new foreground color.

gdk-gc-set-background (*self* <gdk-gc>) (*color* <gdk-color>) [Function]
set-background [Method]

Sets the background color for a graphics context. Note that this function uses *color->pixel*, use `gdk-gc-set-rgb-bg-color` to specify the background color as red, green, blue components.

gc a `<gdk-gc>`.

color the new background color.

gdk-gc-set-rgb-fg-color (*self* <gdk-gc>) (*color* <gdk-color>) [Function]
set-rgb-fg-color [Method]

Set the foreground color of a GC using an unallocated color. The pixel value for the color will be determined using GdkRGB. If the colormap for the GC 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 function for a GC without a colormap is an error.

gc a `<gdk-gc>`

color an unallocated `<gdk-color>`.

gdk-gc-set-rgb-bg-color (*self* <gdk-gc>) (*color* <gdk-color>) [Function]
set-rgb-bg-color [Method]

Set the background color of a GC using an unallocated color. The pixel value for the color will be determined using GdkRGB. If the colormap for the GC 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 function for a GC without a colormap is an error.

gc a `<gdk-gc>`

color an unallocated `<gdk-color>`.

gdk-gc-set-font (*self* <gdk-gc>) (*font* <gdk-font>) [Function]
set-font [Method]

'`gdk_gc_set_font`' is deprecated and should not be used in newly-written code.

Sets the font for a graphics context. (Note that all text-drawing functions in GDK take a *font* argument; the value set here is used when that argument is '`#f`'.)

gc a `<gdk-gc>`.

font the new font.

gdk-gc-set-function (*self* <gdk-gc>) (*function* <gdk-function>) [Function]
set-function [Method]

Determines how the current pixel values and the pixel values being drawn are combined to produce the final pixel values.

gc a `<gdk-gc>`.

function the `<gdk-function>` to use

gdk-gc-set-fill (*self* `<gdk-gc>`) (*fill* `<gdk-fill>`) [Function]
set-fill [Method]

Set the fill mode for a graphics context.

gc a `<gdk-gc>`.

fill the new fill mode.

gdk-gc-set-tile (*self* `<gdk-gc>`) (*tile* `<gdk-pixmap>`) [Function]
set-tile [Method]

Set a tile pixmap for a graphics context. This will only be used if the fill mode is '`GDK_TILED`'.

gc a `<gdk-gc>`.

tile the new tile pixmap.

gdk-gc-set-stipple (*self* `<gdk-gc>`) (*stipple* `<gdk-pixmap>`) [Function]
set-stipple [Method]

Set the stipple bitmap for a graphics context. The stipple will only be used if the fill mode is '`GDK_STIPPLED`' or '`GDK_OPAQUE_STIPPLED`'.

gc a `<gdk-gc>`.

stipple the new stipple bitmap.

gdk-gc-set-ts-origin (*self* `<gdk-gc>`) (*x* `int`) (*y* `int`) [Function]
set-ts-origin [Method]

Set the origin when using tiles or stipples with the GC. The tile or stipple will be aligned such that the upper left corner of the tile or stipple will coincide with this point.

gc a `<gdk-gc>`.

x the x-coordinate of the origin.

y the y-coordinate of the origin.

gdk-gc-set-clip-origin (*self* `<gdk-gc>`) (*x* `int`) (*y* `int`) [Function]
set-clip-origin [Method]

Sets the origin of the clip mask. The coordinates are interpreted relative to the upper-left corner of the destination drawable of the current operation.

gc a `<gdk-gc>`.

x the x-coordinate of the origin.

y the y-coordinate of the origin.

gdk-gc-set-clip-mask (*self* `<gdk-gc>`) (*mask* `<gdk-drawable>`) [Function]
set-clip-mask [Method]

Sets the clip mask for a graphics context from a bitmap. The clip mask is interpreted relative to the clip origin. (See `gdk-gc-set-clip-origin`).

gc the `<gdk-gc>`.

mask a bitmap.

gdk-gc-set-clip-rectangle (*self* `<gdk-gc>`) (*rectangle* `<gdk-rectangle>`) [Function]
set-clip-rectangle [Method]

Sets the clip mask for a graphics context from a rectangle. The clip mask is interpreted relative to the clip origin. (See `gdk-gc-set-clip-origin`).

gc a `<gdk-gc>`.

rectangle the rectangle to clip to.

gdk-gc-set-clip-region (*self* `<gdk-gc>`) (*region* `<gdk-region>`) [Function]
set-clip-region [Method]

Sets the clip mask for a graphics context from a region structure. The clip mask is interpreted relative to the clip origin. (See `gdk-gc-set-clip-origin`).

gc a `<gdk-gc>`.

region the `<gdk-region>`.

gdk-gc-set-subwindow (*self* `<gdk-gc>`) (*mode* `<gdk-subwindow-mode>`) [Function]
set-subwindow [Method]

Sets how drawing with this GC on a window will affect child windows of that window.

gc a `<gdk-gc>`.

mode the subwindow mode.

gdk-gc-set-exposures (*self* `<gdk-gc>`) (*exposures* `bool`) [Function]
set-exposures [Method]

Sets whether copying non-visible portions of a drawable using this graphics context generate exposure events for the corresponding regions of the destination drawable. (See `gdk-draw-drawable`).

gc a `<gdk-gc>`.

exposures if '`#t`', exposure events will be generated.

gdk-gc-set-line-attributes (*self* `<gdk-gc>`) (*line_width* `int`) (*line_style* `<gdk-line-style>`) (*cap_style* `<gdk-cap-style>`) (*join_style* `<gdk-join-style>`) [Function]
set-line-attributes [Method]

Sets various attributes of how lines are drawn. See the corresponding members of `<gdk-gc-values>` for full explanations of the arguments.

gc a `<gdk-gc>`.

line-width the width of lines.

line-style the dash-style for lines.

cap-style the manner in which the ends of lines are drawn.

join-style the in which lines are joined together.

`gdk-gc-copy` (*self* `<gdk-gc>`) (*src_gc* `<gdk-gc>`) [Function]
`copy` [Method]

Copy the set of values from one graphics context onto another graphics context.

dst-gc the destination graphics context.

src-gc the source graphics context.

`gdk-gc-set-colormap` (*self* `<gdk-gc>`) (*colormap* `<gdk-colormap>`) [Function]
`set-colormap` [Method]

Sets the colormap for the GC to the given colormap. The depth of the colormap's visual must match the depth of the drawable for which the GC was created.

gc a `<gdk-gc>`

colormap a `<gdk-colormap>`

`gdk-gc-get-colormap` (*self* `<gdk-gc>`) ⇒ (*ret* `<gdk-colormap>`) [Function]
`get-colormap` [Method]

Retrieves the colormap for a given GC, if it exists. A GC will have a colormap if the drawable for which it was created has a colormap, or if a colormap was set explicitely with gdk_gc_set_colormap.

gc a `<gdk-gc>`

ret the colormap of *gc*, or '`#f`' if *gc* doesn't have one.

`gdk-gc-offset` (*self* `<gdk-gc>`) (*x_offset* `int`) (*y_offset* `int`) [Function]
`offset` [Method]

Offset attributes such as the clip and tile-stipple origins of the GC so that drawing at x - x_offset, y - y_offset with the offset GC has the same effect as drawing at x, y with the original GC.

gc a `<gdk-gc>`

x-offset amount by which to offset the GC in the X direction

y-offset amount by which to offset the GC in the Y direction

8 Drawing Primitives

Functions for drawing points, lines, arcs, and text

8.1 Overview

These functions 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 `<gdk-window>` or `<gdk-pixmap>` objects.

Many of the drawing operations take a `<gdk-gc>` argument, which represents a graphics context. This `<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 data structures like `<pango-context>`, `<pango-layout>` or `<pango-layout-line>` as arguments. If you're using GTK+, the ususal way to obtain these structures is via `gtk-widget-create-pango-context` or `gtk-widget-create-pango-layout`.

8.2 Usage

`<gdk-drawable>` [Class]

Derives from `<gobject>`.

This class defines no direct slots.

`gdk-drawable-get-display` (*self* `<gdk-drawable>`) [Function]
⇒ (*ret* `<gdk-display>`)

`get-display` [Method]

Gets the `<gdk-display>` associated with a `<gdk-drawable>`.

drawable a `<gdk-drawable>`

ret the `<gdk-display>` associated with *drawable*

Since 2.2

`gdk-drawable-get-screen` (*self* `<gdk-drawable>`) [Function]
⇒ (*ret* `<gdk-screen>`)

`get-screen` [Method]

Gets the `<gdk-screen>` associated with a `<gdk-drawable>`.

drawable a `<gdk-drawable>`

ret the `<gdk-screen>` associated with *drawable*

Since 2.2

`gdk-drawable-get-visual` (*self* `<gdk-drawable>`) [Function]
⇒ (*ret* `<gdk-visual>`)

`get-visual` [Method]

Gets the `<gdk-visual>` describing the pixel format of *drawable*.

drawable a `<gdk-drawable>`

ret a `<gdk-visual>`

`gdk-drawable-set-colormap` (*self* `<gdk-drawable>`) (*colormap* `<gdk-colormap>`) [Function]

`set-colormap` [Method]

Sets the colormap associated with *drawable*. 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 `<gdk-gc>` to draw to a drawable, or copying one drawable to another, the colormaps should match.

drawable a `<gdk-drawable>`

colormap a `<gdk-colormap>`

`gdk-drawable-get-colormap` (*self* `<gdk-drawable>`) ⇒ (*ret* `<gdk-colormap>`) [Function]

`get-colormap` [Method]

Gets the colormap for *drawable*, if one is set; returns '`#f`' otherwise.

drawable a `<gdk-drawable>`

ret the colormap, or '`#f`'

`gdk-drawable-get-depth` (*self* `<gdk-drawable>`) ⇒ (*ret* `int`) [Function]

`get-depth` [Method]

Obtains 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.

drawable a `<gdk-drawable>`

ret number of bits per pixel

`gdk-drawable-get-size` (*self* `<gdk-drawable>`) ⇒ (*width* `int`) (*height* `int`) [Function]

`get-size` [Method]

Fills **width* and **height* with the size of *drawable*. *width* or *height* can be '`#f`' if you only want the other one.

On the X11 platform, if *drawable* is a `<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.

drawable a `<gdk-drawable>`

width location to store drawable's width, or '`#f`'

height location to store drawable's height, or '`#f`'

`gdk-drawable-get-clip-region` (*self* `<gdk-drawable>`) ⇒ (*ret* `<gdk-region>`) [Function]

`get-clip-region` [Method]

Computes the region of a drawable that potentially can be written to by drawing primitives. This region will not take into account the clip region for the GC, and may also not take into account other factors such as if the window is obscured by other windows, but no area outside of this region will be affected by drawing primitives.

drawable a `<gdk-drawable>`

ret a `<gdk-region>`. This must be freed with `gdk-region-destroy` when you are done.

`gdk-drawable-get-visible-region` (*self* `<gdk-drawable>`) [Function]
⇒ (*ret* `<gdk-region>`)
`get-visible-region` [Method]

Computes the region of a drawable that is potentially visible. This does not necessarily take into account if the window is obscured by other windows, but no area outside of this region is visible.

drawable a `<gdk-drawable>`

ret a `<gdk-region>`. This must be freed with `gdk-region-destroy` when you are done.

`gdk-draw-point` (*drawable* `<gdk-drawable>`) (*gc* `<gdk-gc>`) (*x* `int`) (*y* `int`) [Function]

Draws a point, using the foreground color and other attributes of the `<gdk-gc>`.

drawable a `<gdk-drawable>` (a `<gdk-window>` or a `<gdk-pixmap>`).

gc a `<gdk-gc>`.

x the x coordinate of the point.

y the y coordinate of the point.

`gdk-draw-line` (*drawable* `<gdk-drawable>`) (*gc* `<gdk-gc>`) (*x1_* `int`) (*y1_* `int`) (*x2_* `int`) (*y2_* `int`) [Function]

Draws a line, using the foreground color and other attributes of the `<gdk-gc>`.

drawable a `<gdk-drawable>` (a `<gdk-window>` or a `<gdk-pixmap>`).

gc a `<gdk-gc>`.

x1 the x coordinate of the start point.

y1 the y coordinate of the start point.

x2 the x coordinate of the end point.

y2 the y coordinate of the end point.

`gdk-draw-pixbuf` (*drawable* `<gdk-drawable>`) (*gc* `<gdk-gc>`) (*pixbuf* `<gdk-pixbuf>`) (*src_x* `int`) (*src_y* `int`) (*dest_x* `int`) (*dest_y* `int`) (*width* `int`) (*height* `int`) (*dither* `<gdk-rgb-dither>`) (*x_dither* `int`) (*y_dither* `int`) [Function]

Renders a rectangular portion of a pixbuf to a drawable. 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-'`#f`' window argument. Otherwise a colormap must be set on them with `gdk-drawable-set-colormap`.

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.

drawable Destination drawable.

gc a `<gdk-gc>`, used for clipping, or '`#f`'

pixbuf a `<gdk-pixbuf>`

src-x Source X coordinate within pixbuf.

src-y Source Y coordinates 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.

height Height of region to render, in pixels, or -1 to use pixbuf height.

dither Dithering mode for `<gdk-rgb>`.

x-dither X offset for dither.

y-dither Y offset for dither.

Since 2.2

`gdk-draw-rectangle` (*drawable* `<gdk-drawable>`) (*gc* `<gdk-gc>`) (*filled* `bool`) (*x* `int`) (*y* `int`) (*width* `int`) (*height* `int`) [Function]

Draws a rectangular outline or filled rectangle, using the foreground color and other attributes of the `<gdk-gc>`.

A rectangle drawn filled is 1 pixel smaller in both dimensions than a rectangle outlined. Calling '`gdk_draw_rectangle (window, gc, TRUE, 0, 0, 20, 20)`' results in a filled rectangle 20 pixels wide and 20 pixels high. Calling '`gdk_draw_rectangle (window, 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.

drawable a `<gdk-drawable>` (a `<gdk-window>` or a `<gdk-pixmap>`).

gc a `<gdk-gc>`.

filled '`#t`' if the rectangle should be filled.

x the x coordinate of the left edge of the rectangle.

y the y coordinate of the top edge of the rectangle.

width the width of the rectangle.

height the height of the rectangle.

`gdk-draw-arc` (*drawable* `<gdk-drawable>`) (*gc* `<gdk-gc>`) (*filled* `bool`) (*x* `int`) (*y* `int`) (*width* `int`) (*height* `int`) (*angle1* `int`) (*angle2* `int`) [Function]

Draws an arc or a filled 'pie slice'. The arc is defined by the bounding rectangle of the entire ellipse, and the start and end angles of the part of the ellipse to be drawn.

drawable a <gdk-drawable> (a <gdk-window> or a <gdk-pixmap>).

gc a <gdk-gc>.

filled '#t' if the arc should be filled, producing 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*, in 1/64ths of a degree.

gdk-draw-glyphs (*drawable* <gdk-drawable>) (*gc* <gdk-gc>) (*font* <pango-font>) (*x* int) (*y* int) (*glyphs* <pango-glyph-string>) [Function]

This is a low-level function; 99% of text rendering should be done using gdk-draw-layout instead.

A glyph is a single image in a font. This function draws a sequence of glyphs. To obtain a sequence of glyphs you have to understand a lot about internationalized text handling, which you don't want to understand; thus, use gdk-draw-layout instead of this function, gdk-draw-layout handles the details.

drawable a <gdk-drawable>

gc a <gdk-gc>

font font to be used

x X coordinate of baseline origin

y Y coordinate of baseline origin

glyphs the glyph string to draw

gdk-draw-glyphs-transformed (*drawable* <gdk-drawable>) (*gc* <gdk-gc>) (*matrix* <pango-matrix>) (*font* <pango-font>) (*x* int) (*y* int) (*glyphs* <pango-glyph-string>) [Function]

Renders a <pango-glyph-string> onto a drawable, possibly transforming the layed-out coordinates through a transformation matrix. Note that the transformation matrix for *font* is not changed, so to produce correct rendering results, the *font* must have been loaded using a <pango-context> with an identical transformation matrix to that passed in to this function.

See also gdk-draw-glyphs, gdk-draw-layout.

drawable a <gdk-drawable>

gc a <gdk-gc>

matrix a <pango-matrix>, or '#f' to use an identity transformation

font the font in which to draw the string *x* the x position of the start of the string (in Pango units in user space coordinates)

y the y position of the baseline (in Pango units in user space coordinates)

glyphs the glyph string to draw

Since 2.6

gdk-draw-layout-line (*drawable* `<gdk-drawable>`) (*gc* `<gdk-gc>`) (*x* `int`) (*y* `int`) (*line* `<pango-layout-line>`) [Function]

Render a `<pango-layout-line>` onto an GDK drawable

If the layout's `<pango-context>` has a transformation matrix set, then *x* and *y* specify the position of the left edge of the baseline (left is in before-tranform user coordinates) in after-transform device coordinates.

drawable the drawable on which to draw the line

gc base graphics to use

x the x position of start of string (in pixels)

y the y position of baseline (in pixels)

line a `<pango-layout-line>`

gdk-draw-layout-line-with-colors (*drawable* `<gdk-drawable>`) (*gc* `<gdk-gc>`) (*x* `int`) (*y* `int`) (*line* `<pango-layout-line>`) (*foreground* `<gdk-color>`) (*background* `<gdk-color>`) [Function]

Render a `<pango-layout-line>` onto a `<gdk-drawable>`, overriding the layout's normal colors with *foreground* and/or *background*. *foreground* and *background* need not be allocated.

If the layout's `<pango-context>` has a transformation matrix set, then *x* and *y* specify the position of the left edge of the baseline (left is in before-tranform user coordinates) in after-transform device coordinates.

drawable the drawable on which to draw the line

gc base graphics to use

x the x position of start of string (in pixels)

y the y position of baseline (in pixels)

line a `<pango-layout-line>`

foreground
foreground override color, or '`#f`' for none

background
background override color, or '`#f`' for none

gdk-draw-layout (*drawable* `<gdk-drawable>`) (*gc* `<gdk-gc>`) (*x* `int`) (*y* `int`) (*layout* `<pango-layout>`) [Function]

Render a `<pango-layout>` onto a GDK drawable

If the layout's `<pango-context>` has a transformation matrix set, then *x* and *y* specify the position of the top left corner of the bounding box (in device space) of the transformed layout.

If you're using GTK+, the usual way to obtain a `<pango-layout>` is `gtk-widget-create-pango-layout`.

drawable the drawable on which to draw string

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>`

gdk-draw-layout-with-colors (*drawable* `<gdk-drawable>`) (*gc* `<gdk-gc>`) (*x* `int`) (*y* `int`) (*layout* `<pango-layout>`) (*foreground* `<gdk-color>`) (*background* `<gdk-color>`) [Function]

Render a `<pango-layout>` onto a `<gdk-drawable>`, overriding the layout's normal colors with *foreground* and/or *background*. *foreground* and *background* need not be allocated.

If the layout's `<pango-context>` has a transformation matrix set, then *x* and *y* specify the position of the top left corner of the bounding box (in device space) of the transformed layout.

If you're using GTK+, the ususal way to obtain a `<pango-layout>` is `gtk-widget-create-pango-layout`.

drawable the drawable on which to draw string

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
foreground override color, or '`#f`' for none

background
background override color, or '`#f`' for none

gdk-draw-string (*drawable* `<gdk-drawable>`) (*font* `<gdk-font>`) (*gc* `<gdk-gc>`) (*x* `int`) (*y* `int`) (*string* `mchars`) [Function]

'`gdk_draw_string`' is deprecated and should not be used in newly-written code. Use `gdk-draw-layout` instead.

Draws a string of characters in the given font or fontset.

drawable a `<gdk-drawable>` (a `<gdk-window>` or a `<gdk-pixmap>`).

font a `<gdk-font>`.

gc a `<gdk-gc>`.

x the x coordinate of the left edge of the text.

y the y coordinate of the baseline of the text.

string the string of characters to draw.

`gdk-draw-drawable` (*drawable* `<gdk-drawable>`) (*gc* `<gdk-gc>`) (*src* `<gdk-drawable>`) (*xsrc* `int`) (*ysrc* `int`) (*xdest* `int`) (*ydest* `int`) (*width* `int`) (*height* `int`) [Function]

Copies the *width* x *height* region of *src* at coordinates (*xsrc*, *ysrc*) to coordinates (*xdest*, *ydest*) in *drawable*. *width* and/or *height* may be given as -1, in which case the entire *src* drawable will be copied.

Most fields in *gc* are not used for this operation, but notably 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/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 the stipple on the `<gdk-gc>`, set the fill mode to '`GDK_STIPPLED`', and then draw the rectangle.

drawable a `<gdk-drawable>`

gc a `<gdk-gc>` sharing the drawable's visual and colormap

src the source `<gdk-drawable>`, which may be the same as *drawable*

xsrc X position in *src* of rectangle to draw

ysrc Y position in *src* of rectangle to draw

xdest X position in *drawable* where the rectangle should be drawn

ydest Y position in *drawable* where the rectangle should be drawn

width width of rectangle to draw, or -1 for entire *src* width

height height of rectangle to draw, or -1 for entire *src* height

`gdk-draw-image` (*drawable* `<gdk-drawable>`) (*gc* `<gdk-gc>`) (*image* `<gdk-image>`) (*xsrc* `int`) (*ysrc* `int`) (*xdest* `int`) (*ydest* `int`) (*width* `int`) (*height* `int`) [Function]

Draws a `<gdk-image>` onto a drawable. The depth of the `<gdk-image>` must match the depth of the `<gdk-drawable>`.

drawable a `<gdk-drawable>` (a `<gdk-window>` or a `<gdk-pixmap>`).

gc a `<gdk-gc>`.

image the `<gdk-image>` to draw.

xsrc the left edge of the source rectangle within *image*.

ysrc the top of the source rectangle within *image*.

xdest the x coordinate of the destination within *drawable*.

ydest the y coordinate 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*.

`gdk-drawable-get-image` (*self* `<gdk-drawable>`) (*x* `int`) (*y* `int`) (*width* `int`) (*height* `int`) ⇒ (*ret* `<gdk-image>`) [Function]

`get-image` [Method]

A `<gdk-image>` stores client-side image data (pixels). In contrast, `<gdk-pixmap>` and `<gdk-window>` are server-side objects. `gdk-drawable-get-image` obtains the pixels from a server-side drawable as a client-side `<gdk-image>`. The format of a `<gdk-image>` depends on the `<gdk-visual>` of the current display, which makes manipulating `<gdk-image>` extremely difficult; therefore, in most cases you should use `gdk-pixbuf-get-from-drawable` instead of this lower-level function. A `<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.

x, *y*, *width*, and *height* define the region of *drawable* to obtain as an image.

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 `<gdk-window>` and partially offscreen or obscured, then the obscured portions of the returned image will contain undefined data.

drawable a `<gdk-drawable>`

x x coordinate on *drawable*

y y coordinate on *drawable*

width width of region to get

height height or region to get

ret a `<gdk-image>` containing the contents of *drawable*

`gdk-drawable-copy-to-image` (*self* `<gdk-drawable>`) (*image* `<gdk-image>`) (*src_x* `int`) (*src_y* `int`) (*dest_x* `int`) (*dest_y* `int`) (*width* `int`) (*height* `int`) ⇒ (*ret* `<gdk-image>`) [Function]

`copy-to-image` [Method]

Copies a portion of *drawable* into the client side image structure *image*. If *image* is '`#f`', creates a new image of size *width* x *height* and copies into that. See `gdk-drawable-get-image` for further details.

drawable a `<gdk-drawable>`

image a `<gdk-drawable>`, or '`#f`' if a new *image* should be created.

src-x x coordinate on *drawable*

src-y y coordinate on *drawable*

dest-x x coordinate within *image*. Must be 0 if *image* is '`#f`'

dest-y y coordinate within *image*. Must be 0 if *image* is '`#f`'

width width of region to get

height height or region to get

ret *image*, or a new a `<gdk-image>` containing the contents of *drawable*

Since 2.4

9 Bitmaps and Pixmaps

Offscreen drawables

9.1 Overview

Pixmaps are offscreen drawables. They can be drawn upon with the standard drawing primitives, then copied to another drawable (such as a `<gdk-window>`) with `gdk-pixmap-draw`. The depth of a pixmap is the number of bits per pixels. Bitmaps are simply pixmaps with a depth of 1. (That is, they are monochrome bitmaps - each pixel can be either on or off).

9.2 Usage

`gdk-pixmap-new` (*drawable* `<gdk-drawable>`) (*width* `int`) (*height* `int`) (*depth* `int`) ⇒ (*ret* `<gdk-pixmap>`) [Function]

Create a new pixmap with a given size and depth.

drawable A `<gdk-drawable>`, used to determine default values for the new pixmap. Can be '`#f`' 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. If -1, and *drawable* is not '`#f`', the depth of the new pixmap will be equal to that of *drawable*.

ret the `<gdk-pixmap>`

`gdk-bitmap-create-from-data` (*drawable* `<gdk-drawable>`) (*data* `mchars`) (*width* `int`) (*height* `int`) ⇒ (*ret* `<gdk-drawable>`) [Function]

Creates a new bitmap from data in XBM format.

drawable a `<gdk-drawable>`, used to determine default values for the new pixmap. Can be '`#f`', in which case the root window is used.

data a pointer to the XBM data.

width the width of the new pixmap in pixels.

height the height of the new pixmap in pixels.

ret the `<gdk-bitmap>`

`gdk-pixmap-create-from-data` (*drawable* `<gdk-drawable>`) (*data* `mchars`) (*width* `int`) (*height* `int`) (*depth* `int`) (*fg* `<gdk-color>`) (*bg* `<gdk-color>`) ⇒ (*ret* `<gdk-pixmap>`) [Function]

Create a two-color pixmap from data in XBM data.

drawable a `<gdk-drawable>`, used to determine default values for the new pixmap. Can be '`#f`', if the depth is given.

data a pointer to the 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.

fg the foreground color.

bg the background color.

ret the `<gdk-pixmap>`

10 GdkRGB

Renders RGB, grayscale, or indexed image data to a GdkDrawable

10.1 Overview

GdkRGB is a low-level module which renders RGB, grayscale, and indexed colormap images to a `<gdk-drawable>`. It does this as efficiently as possible, handling issues such as colormaps, visuals, dithering, temporary buffers, and so on. Most code should use the higher-level `<gdk-pixbuf>` features in place of this module; for example, `gdk-pixbuf-render-to-drawable` uses GdkRGB in its implementation.

GdkRGB allocates a color cube to use when rendering images. You can set the threshold for installing colormaps with `gdk-rgb-set-min-colors`. The default is 5x5x5 (125). If a colorcube of this size or larger can be allocated in the default colormap, then that's done. Otherwise, GdkRGB creates its own private colormap. Setting it to 0 means that it always tries to use the default colormap, and setting it to 216 means that it always creates a private one if it cannot allocate the 6x6x6 colormap in the default. If you always want a private colormap (to avoid consuming too many colormap entries for other apps, say), you can use '`gdk_rgb_set_install(TRUE)`'. Setting the value greater than 216 exercises a bug in older versions of GdkRGB. Note, however, that setting it to 0 doesn't let you get away with ignoring the colormap and visual - a colormap is always created in grayscale and direct color modes, and the visual is changed in cases where a "better" visual than the default is available.

```
&#x0023;include <gtk/gtk.h>

&#x0023;define IMAGE_WIDTH 256
&#x0023;define IMAGE_HEIGHT 256

guchar rgbbuf[IMAGE_WIDTH * IMAGE_HEIGHT * 3];

gboolean on_darea_expose (GtkWidget *widget,
  GdkEventExpose *event,
  gpointer user_data);

int
main (int argc, char *argv[])
{
  GtkWidget *window, *darea;
  gint x, y;
  guchar *pos;

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  darea = gtk_drawing_area_new ();
  gtk_widget_set_size_request (darea, IMAGE_WIDTH, IMAGE_HEIGHT);
```

```
  gtk_container_add (GTK_CONTAINER (window), darea);
  gtk_signal_connect (GTK_OBJECT (darea), "expose-event",
                      GTK_SIGNAL_FUNC (on_darea_expose), NULL);
  gtk_widget_show_all (window);

  /* Set up the RGB buffer. */
  pos = rgbbuf;
  for (y = 0; y < IMAGE_HEIGHT; y++)
    {
      for (x = 0; x < IMAGE_WIDTH; x++)
{
  *pos++ = x - x % 32; /* Red. */
  *pos++ = (x / 32) * 4 + y - y % 32; /* Green. */
  *pos++ = y - y % 32; /* Blue. */
}
    }

  gtk_main ();
  return 0;
}


gboolean
on_darea_expose (GtkWidget *widget,
 GdkEventExpose *event,
 gpointer user_data)
{
  gdk_draw_rgb_image (widget->window, widget->style->fg_gc[GTK_STATE_NORMAL],
      0, 0, IMAGE_WIDTH, IMAGE_HEIGHT,
      GDK_RGB_DITHER_MAX, rgbbuf, IMAGE_WIDTH * 3);

  return TRUE;
}
```

10.2 Usage

`gdk-rgb-init` [Function]

: ‘`gdk_rgb_init`’ is deprecated and should not be used in newly-written code.

  This function no longer does anything at all. It’s completely useless (and harmless).

`gdk-rgb-gc-set-foreground` (*gc* `<gdk-gc>`) (*rgb* `unsigned-int32`) [Function]

: ‘`gdk_rgb_gc_set_foreground`’ is deprecated and should not be used in newly-written code.

  Sets the foreground color in *gc* to the specified color (or the closest approximation, in the case of limited visuals).

  gc The `<gdk-gc>` to modify.

rgb The color, represented as a 0xRRGGBB integer value.

`gdk-rgb-gc-set-background` (*gc* `<gdk-gc>`) (*rgb* `unsigned-int32`) [Function]

'`gdk_rgb_gc_set_background`' is deprecated and should not be used in newly-written code.

Sets the background color in *gc* to the specified color (or the closest approximation, in the case of limited visuals).

gc The `<gdk-gc>` to modify.

rgb The color, represented as a 0xRRGGBB integer value.

`gdk-rgb-xpixel-from-rgb` (*rgb* `unsigned-int32`) [Function]
⇒ (*ret* `unsigned-long`)

'`gdk_rgb_xpixel_from_rgb`' is deprecated and should not be used in newly-written code.

Finds the X pixel closest in color to the *rgb* color specified. This value may be used to set the struct.

rgb The color, represented as a 0xRRGGBB integer value.

ret The X pixel value.

`gdk-rgb-set-install` (*install* `bool`) [Function]

If *install* is '`#t`', directs GdkRGB to always install a new "private" colormap rather than trying to find a best fit with the colors already allocated. Ordinarily, GdkRGB will install a colormap only 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.

install '`#t`' to set install mode.

`gdk-rgb-set-min-colors` (*min_colors* `int`) [Function]

Sets the minimum number of colors for the color cube. Generally, GdkRGB tries to allocate the largest color cube it can. If it can't allocate a color cube at least as large as *min-colors*, it installs a private colormap.

min-colors The minimum number of colors accepted.

`gdk-rgb-get-visual` ⇒ (*ret* `<gdk-visual>`) [Function]

Gets a "preferred visual" chosen by GdkRGB for rendering image data on the default screen. In previous versions of GDK, this was the only visual GdkRGB could use for rendering. In current versions, it's simply the visual GdkRGB would have chosen as the optimal one in those previous versions. GdkRGB can now render to drawables with any visual.

ret The `<gdk-visual>` chosen by GdkRGB.

`gdk-rgb-get-colormap` ⇒ (*ret* `<gdk-colormap>`) [Function]

Get the preferred colormap for rendering image data. 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.

ret the preferred colormap

`gdk-rgb-ditherable` ⇒ (*ret* `bool`) [Function]

Determines whether the preferred 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.

ret '`#t`' if the preferred visual is ditherable.

`gdk-rgb-colormap-ditherable` (*cmap* `<gdk-colormap>`) [Function]
⇒ (*ret* `bool`)

Determines whether the visual associated with *cmap* 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.

cmap a `<gdk-colormap>`

ret '`#t`' if the visual associated with *cmap* is ditherable.

`gdk-rgb-set-verbose` (*verbose* `bool`) [Function]

Sets the "verbose" flag. This is generally only useful for debugging.

verbose '`#t`' if verbose messages are desired.

11 Images

A client-side area for bit-mapped graphics

11.1 Overview

The `<gdk-image>` type represents an area for drawing graphics. It has now been superceded to a large extent by the much more flexible GdkRGB functions.

To create an empty `<gdk-image>` use `gdk-image-new`. To create a `<gdk-image>` from bitmap data use `gdk-image-new-bitmap`. To create an image from part of a `<gdk-window>` use `gdk-drawable-get-image`.

The image can be manipulated with `gdk-image-get-pixel` and `gdk-image-put-pixel`, or alternatively by changing the actual pixel data. Though manipulating the pixel data requires complicated code to cope with the different formats that may be used.

To draw a `<gdk-image>` in a `<gdk-window>` or `<gdk-pixmap>` use `gdk-draw-image`.

To destroy a `<gdk-image>` use `gdk-image-destroy`.

11.2 Usage

`<gdk-image>` [Class]

Derives from `<gobject>`.

This class defines no direct slots.

`gdk-image-new` (*type* `<gdk-image-type>`) (*visual* `<gdk-visual>`) (*width* `int`) (*height* `int`) ⇒ (*ret* `<gdk-image>`) [Function]

Creates a new `<gdk-image>`.

type the type of the `<gdk-image>`, one of '`GDK_IMAGE_NORMAL`', '`GDK_IMAGE_SHARED`' and '`GDK_IMAGE_FASTEST`'. '`GDK_IMAGE_FASTEST`' is probably the best choice, since it will try creating a '`GDK_IMAGE_SHARED`' image first and if that fails it will then use '`GDK_IMAGE_NORMAL`'.

visual the `<gdk-visual>` to use for the image.

width the width of the image in pixels.

height the height of the image in pixels.

ret a new `<gdk-image>`, or '`#f`' if the image could not be created.

`gdk-image-get` (*drawable* `<gdk-drawable>`) (*x* `int`) (*y* `int`) (*width* `int`) (*height* `int`) ⇒ (*ret* `<gdk-image>`) [Function]

'`gdk_image_get`' is deprecated and should not be used in newly-written code.

This is a deprecated wrapper for `gdk-drawable-get-image`; `gdk-drawable-get-image` should be used instead. Or even better: in most cases `gdk-pixbuf-get-from-drawable` is the most convenient choice.

drawable a `<gdk-drawable>`

x x coordinate in *window*

y y coordinate in *window*

width width of area in *window*

height height of area in *window*

ret a new `<gdk-image>` or '`#f`'

`gdk-image-get-colormap` (*self* `<gdk-image>`) [Function]
⇒ (*ret* `<gdk-colormap>`)

`get-colormap` [Method]

Retrieves 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 explicitely with `gdk-image-set-colormap`.

image a `<gdk-image>`

ret colormap for the image

`gdk-image-set-colormap` (*self* `<gdk-image>`) [Function]
(*colormap* `<gdk-colormap>`)

`set-colormap` [Method]

Sets the colormap for the image to the given colormap. Normally there's no need to use this function, 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.

image a `<gdk-image>`

colormap a `<gdk-colormap>`

`gdk-image-put-pixel` (*self* `<gdk-image>`) (*x* `int`) (*y* `int`) [Function]
(*pixel* `unsigned-int32`)

`put-pixel` [Method]

Sets a pixel in a `<gdk-image>` to a given pixel value.

image a `<gdk-image>`.

x the x coordinate of the pixel to set.

y the y coordinate of the pixel to set.

pixel the pixel value to set.

`gdk-image-get-pixel` (*self* `<gdk-image>`) (*x* `int`) (*y* `int`) [Function]
⇒ (*ret* `unsigned-int32`)

`get-pixel` [Method]

Gets a pixel value at a specified position in a `<gdk-image>`.

image a `<gdk-image>`.

x the x coordinate of the pixel to get.

y the y coordinate of the pixel to get.

ret the pixel value at the given position.

12 Pixbufs

Functions for rendering pixbufs on drawables

12.1 Overview

These functions allow to render pixbufs on drawables. Pixbufs are client-side images. For details on how to create and manipulate pixbufs, see the `<gdk-pixbuf>` API documentation.

12.2 Usage

`gdk-pixbuf-render-threshold-alpha` (*self* `<gdk-pixbuf>`) (*bitmap* `<gdk-drawable>`) (*src_x* `int`) (*src_y* `int`) (*dest_x* `int`) (*dest_y* `int`) (*width* `int`) (*height* `int`) (*alpha_threshold* `int`) [Function]

`render-threshold-alpha` [Method]

Takes the opacity values in a rectangular portion of a pixbuf and thresholds them to produce a bi-level alpha mask that can be used as a clipping mask for a drawable.

pixbuf A pixbuf.

bitmap Bitmap where the bilevel mask will be painted to.

src-x Source X coordinate.

src-y source Y coordinate.

dest-x Destination X coordinate.

dest-y Destination Y coordinate.

width Width of region to threshold, or -1 to use pixbuf width

height Height of region to threshold, or -1 to use pixbuf height

alpha-threshold
Opacity values below this will be painted as zero; all other values will be painted as one.

`gdk-pixbuf-render-to-drawable` (*self* `<gdk-pixbuf>`) (*drawable* `<gdk-drawable>`) (*gc* `<gdk-gc>`) (*src_x* `int`) (*src_y* `int`) (*dest_x* `int`) (*dest_y* `int`) (*width* `int`) (*height* `int`) (*dither* `<gdk-rgb-dither>`) (*x_dither* `int`) (*y_dither* `int`) [Function]

`render-to-drawable` [Method]

'`gdk_pixbuf_render_to_drawable`' has been deprecated since version 2.4 and should not be used in newly-written code. This function is obsolete. Use `gdk-draw-pixbuf` instead.

Renders a rectangular portion of a pixbuf to a drawable while using the specified GC. This is done using GdkRGB, so the specified drawable must have the GdkRGB visual and colormap. Note that this function will ignore the opacity information for images with an alpha channel; the GC must already have the clipping mask set if you want transparent regions to show through.

For an explanation of dither offsets, see the GdkRGB documentation. In brief, the dither offset is important when re-rendering partial regions of an image to a rendered version of the full image, or for 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.

pixbuf A pixbuf.

drawable Destination drawable.

gc GC used for rendering.

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

height Height of region to render, in pixels, or -1 to use pixbuf height

dither Dithering mode for GdkRGB.

x-dither X offset for dither.

y-dither Y offset for dither.

`gdk-pixbuf-render-to-drawable-alpha` (*self* `<gdk-pixbuf>`) [Function]
(*drawable* `<gdk-drawable>`) (*src_x* `int`) (*src_y* `int`) (*dest_x* `int`)
(*dest_y* `int`) (*width* `int`) (*height* `int`)
(*alpha_mode* `<gdk-pixbuf-alpha-mode>`) (*alpha_threshold* `int`)
(*dither* `<gdk-rgb-dither>`) (*x_dither* `int`) (*y_dither* `int`)

`render-to-drawable-alpha` [Method]

'`gdk_pixbuf_render_to_drawable_alpha`' has been deprecated since version 2.4 and should not be used in newly-written code. This function is obsolete. Use `gdk-draw-pixbuf` instead.

Renders a rectangular portion of a pixbuf to a drawable. 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-'`#f`' window argument. Otherwise a colormap must be set on them with gdk_drawable_set_colormap.

On older X servers, rendering pixbufs with an alpha channel involves round trips to the X server, and may be somewhat slow.

pixbuf A pixbuf.

drawable Destination drawable.

src-x Source X coordinate within pixbuf.

src-y Source Y coordinates 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.

height Height of region to render, in pixels, or -1 to use pixbuf height.

alpha-mode
Ignored. Present for backwards compatibility.

alpha-threshold
Ignored. Present for backwards compatibility

dither Dithering mode for GdkRGB.

x-dither X offset for dither.

y-dither Y offset for dither.

`gdk-pixbuf-get-from-drawable` (*self* `<gdk-pixbuf>`) (*src* `<gdk-drawable>`) (*cmap* `<gdk-colormap>`) (*src_x* `int`) (*src_y* `int`) (*dest_x* `int`) (*dest_y* `int`) (*width* `int`) (*height* `int`) ⇒ (*ret* `<gdk-pixbuf>`) [Function]

`get-from-drawable` [Method]

Transfers image data from a `<gdk-drawable>` and converts it to an RGB(A) representation inside a `<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 the drawable *src* has no colormap (`gdk-drawable-get-colormap` returns '`#f`'), then a suitable colormap must be specified. Typically a `<gdk-window>` or a pixmap created by passing a `<gdk-window>` to `gdk-pixmap-new` will already have a colormap associated with it. If the drawable has a colormap, the *cmap* argument will be ignored. If the drawable 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, `gdk-colormap-get-system` returns the correct colormap to use.

If the specified destination pixbuf *dest* is '`#f`', then this function will create an RGB pixbuf with 8 bits per channel and no alpha, with the same size specified by the *width* and *height* arguments. In this case, the *dest-x* and *dest-y* arguments must be specified as 0. If the specified destination pixbuf is not '`#f`' and it contains alpha information, then the filled pixels will be set to full opacity (alpha = 255).

If the specified drawable is a pixmap, then the requested source rectangle must be completely contained within the pixmap, otherwise the function will return '`#f`'. 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 the specified drawable 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), then '`#f`' will be returned.

If memory can't be allocated for the return value, '`#f`' will be returned instead.

(In short, there are several ways this function can fail, and if it fails it returns '`#f`'; so check the return value.)

This function calls `gdk-drawable-get-image` internally and converts the resulting image to a `<gdk-pixbuf>`, so the documentation for `gdk-drawable-get-image` may also be relevant.

dest Destination pixbuf, or '`#f`' if a new pixbuf should be created.

src Source drawable.

cmap A colormap if *src* doesn't have one set.

src-x Source X coordinate within drawable.

src-y Source Y coordinate within drawable.

dest-x Destination X coordinate in pixbuf, or 0 if *dest* is NULL.

dest-y Destination Y coordinate in pixbuf, or 0 if *dest* is NULL.

width Width in pixels of region to get.

height Height in pixels of region to get.

ret The same pixbuf as *dest* if it was non-'`#f`', or a newly-created pixbuf with a reference count of 1 if no destination pixbuf was specified, or '`#f`' on error

`gdk-pixbuf-get-from-image` (*self* `<gdk-pixbuf>`) (*src* `<gdk-image>`) [Function]
(*cmap* `<gdk-colormap>`) (*src_x* `int`) (*src_y* `int`) (*dest_x* `int`) (*dest_y* `int`)
(*width* `int`) (*height* `int`) ⇒ (*ret* `<gdk-pixbuf>`)

`get-from-image` [Method]

Same as `gdk-pixbuf-get-from-drawable` but gets the pixbuf from an image.

dest Destination pixbuf, or '`#f`' if a new pixbuf should be created.

src Source `<gdk-image>`.

cmap A colormap, or '`#f`' to use the one for *src*

src-x Source X coordinate within drawable.

src-y Source Y coordinate within drawable.

dest-x Destination X coordinate in pixbuf, or 0 if *dest* is NULL.

dest-y Destination Y coordinate in pixbuf, or 0 if *dest* is NULL.

width Width in pixels of region to get.

height Height in pixels of region to get.

ret *dest*, newly-created pixbuf if *dest* was '`#f`', '`#f`' on error

13 Colormaps and Colors

Manipulation of colors and colormaps

13.1 Overview

These functions are used to modify colormaps. A colormap is an object that contains the mapping between the color values stored in memory and the RGB values that are used to display color values. In general, colormaps only contain significant information for pseudo-color visuals, but even for other visual types, a colormap object is required in some circumstances.

There are a couple of special colormaps that can be retrieved. The system colormap (retrieved with `gdk-colormap-get-system`) is the default colormap of the system. If you are using GdkRGB, there is another colormap that is important - the colormap in which GdkRGB works, retrieved with `gdk-rgb-get-cmap`. However, when using GdkRGB, it is not generally necessary to allocate colors directly.

In previous revisions of this interface, a number of functions that take a `<gdk-colormap>` parameter were replaced with functions whose names began with "gdk_colormap_". This process will probably be extended somewhat in the future - `gdk-color-white`, `gdk-color-black`, and `gdk-color-change` will probably become aliases.

13.2 Usage

`<gdk-color>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`<gdk-colormap>` [Class]

Derives from `<gobject>`.

This class defines no direct slots.

`gdk-color-copy` (*self* `<gdk-color>`) ⇒ (*ret* `<gdk-color>`) [Function]

Makes a copy of a color structure. The result must be freed using `gdk-color-free`.

color a `<gdk-color>`.

ret a copy of *color*.

`gdk-color-white` (*colormap* `<gdk-colormap>`) (*color* `<gdk-color>`) ⇒ (*ret* `int`) [Function]

'`gdk_color_white`' is deprecated and should not be used in newly-written code.

Returns the white color for a given colormap. The resulting value has already allocated been allocated.

colormap a `<gdk-colormap>`.

color the location to store the color.

ret '`#t`' if the allocation succeeded.

gdk-color-black (*colormap* <gdk-colormap>) (*color* <gdk-color>) ⇒ (*ret* int) [Function]

'gdk_color_black' is deprecated and should not be used in newly-written code.

Returns the black color for a given colormap. The resulting value has already benn allocated.

colormap a <gdk-colormap>.

color the location to store the color.

ret '#t' if the allocation succeeded.

gdk-color-parse (*spec* mchars) (*color* <gdk-color>) ⇒ (*ret* bool) [Function]

Parses a textual specification of a color and fill in the , <gdk-color> structure. The color is *not* allocated, you must call gdk-colormap-alloc-color yourself. The text string can be in any of the forms accepted by x-parse-color; these include name for a color from 'rgb.txt', such as 'DarkSlateGray', or a hex specification such as '#3050b2' or '#35b'.

spec the string specifying the color.

color the <gdk-color> to fill in

ret '#t' if the parsing succeeded.

gdk-color-alloc (*colormap* <gdk-colormap>) (*color* <gdk-color>) ⇒ (*ret* int) [Function]

'gdk_color_alloc' is deprecated and should not be used in newly-written code. Use gdk-colormap-alloc-color instead.

Allocates a single color from a colormap.

colormap a <gdk-colormap>.

color The color to allocate. On return, the filled in.

ret '#t' if the allocation succeeded.

gdk-color-change (*colormap* <gdk-colormap>) (*color* <gdk-color>) ⇒ (*ret* int) [Function]

'gdk_color_change' is deprecated and should not be used in newly-written code.

Changes the value of a color that has already been allocated. If *colormap* is not a private colormap, then the color must have been allocated using gdk-colormap-alloc-colors with the *writeable* set to '#t'.

colormap a <gdk-colormap>.

color a <gdk-color>, with the color to change in the field, and the new value in the remaining fields.

ret '#t' if the color was successfully changed.

14 Visuals

Low-level display hardware information

14.1 Overview

A `<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 GTK+. Visuals are useful to interpret the contents of a `<gdk-image>`, but you should avoid `<gdk-image>` precisely because its contents depend on the display hardware; use `<gdk-pixbuf>` instead, for all but the most low-level purposes. Also, anytime you provide a `<gdk-colormap>`, the visual is implied as part of the colormap (`gdk-colormap-get-visual`), so you won't have to provide a visual in addition.

There are several standard visuals. The visual returned by `gdk-visual-get-system` is the system's default visual. `gdk-rgb-get-visual` return the visual most suited to displaying full-color image data. If you use the calls in `<gdk-rgb>`, you should create your windows using this visual (and the colormap returned by `gdk-rgb-get-colormap`).

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, '`GDK_VISUAL_PSEUDO_COLOR`' is preferred at 8bpp, otherwise, the visual types are ranked in the order of (highest to lowest) '`GDK_VISUAL_DIRECT_COLOR`', '`GDK_VISUAL_TRUE_COLOR`', '`GDK_VISUAL_PSEUDO_COLOR`', '`GDK_VISUAL_STATIC_COLOR`', '`GDK_VISUAL_GRAYSCALE`', then '`GDK_VISUAL_STATIC_GRAY`'.

14.2 Usage

`<gdk-visual>` [Class]

Derives from `<gobject>`.

This class defines no direct slots.

`gdk-list-visuals` ⇒ (*ret* `glist-of`) [Function]

Lists the available visuals for the default screen. (See `gdk-screen-list-visuals`) A 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.

Call `g-list-free` on the return value when you're finished with it.

ret a list of visuals; the list must be freed, but not its contents

`gdk-visual-get-best-depth` ⇒ (*ret* `int`) [Function]

Get the best available depth for the default GDK screen. "Best" means "largest," i.e. 32 preferred over 24 preferred over 8 bits per pixel.

ret best available depth

`gdk-visual-get-best-type` ⇒ (*ret* `<gdk-visual-type>`) [Function]

Return the best available visual type for the default GDK screen.

ret best visual type

`gdk-visual-get-system` ⇒ (*ret* `<gdk-visual>`) [Function]

Get the system'sdefault visual for the default GDK screen. This is the visual for the root window of the display. The return value should not be freed.

ret system visual

`gdk-visual-get-best` ⇒ (*ret* `<gdk-visual>`) [Function]

Get the visual with the most available colors for the default GDK screen. The return value should not be freed.

ret best visual

`gdk-visual-get-best-with-depth` (*depth* `int`) [Function]
⇒ (*ret* `<gdk-visual>`)

Get the best visual with depth *depth* for the default GDK screen. Color visuals and visuals with mutable colormaps are preferred over grayscale or fixed-colormap visuals. The return value should not be freed. '`#f`' may be returned if no visual supports *depth*.

depth a bit depth

ret best visual for the given depth

`gdk-visual-get-best-with-type` (*visual_type* `<gdk-visual-type>`) [Function]
⇒ (*ret* `<gdk-visual>`)

Get the best visual of the given *visual-type* for the default GDK screen. Visuals with higher color depths are considered better. The return value should not be freed. '`#f`' may be returned if no visual has type *visual-type*.

visual-type
a visual type

ret best visual of the given type

`gdk-visual-get-best-with-both` (*depth* `int`) [Function]
(*visual_type* `<gdk-visual-type>`) ⇒ (*ret* `<gdk-visual>`)

Combines `gdk-visual-get-best-with-depth` and `gdk-visual-get-best-with-type`.

depth a bit depth

visual-type
a visual type

ret best visual with both *depth* and *visual-type*, or '`#f`' if none

`gdk-visual-get-screen` (*self* `<gdk-visual>`) ⇒ (*ret* `<gdk-screen>`) [Function]
`get-screen` [Method]

Gets the screen to which this visual belongs *visual* a `<gdk-visual>`

ret the screen to which this visual belongs.

Since 2.2

15 Fonts

Loading and manipulating fonts

15.1 Overview

The `<gdk-font>` data type represents a font for drawing on the screen. These functions provide support for loading fonts, and also for determining the dimensions of characters and strings when drawn with a particular font.

Fonts in X are specified by a *X Logical Font Description*. The following description is considerably simplified. For definitive information about XLFD's see the X reference documentation. A X Logical Font Description (XLFD) consists of a sequence of fields separated (and surrounded by) '-' characters. For example, Adobe Helvetica Bold 12 pt, has the full description:

```
"-adobe-helvetica-bold-r-normal--12-120-75-75-p-70-iso8859-1"
```

The fields in the XLFD are:

When specifying a font via a X logical Font Description, '*' can be used as a wildcard to match any portion of the XLFD. For instance, the above example could also be specified as It is generally a good idea to use wildcards for any portion of the XLFD that your program does not care about specifically, since that will improve the chances of finding a matching font.

```
"-*-helvetica-bold-r-normal--*-120-*-*-*-*-iso8859-1"
```

A *fontset* is a list of fonts that is used for drawing international text that may contain characters from a number of different character sets. It is represented by a list of XLFD's.

The font for a given character set is determined by going through the list of XLFD's in order. For each one, if the registry and and encoding fields match the desired character set, then that font is used, otherwise if the XLFD contains wild-cards for the registry and encoding fields, the registry and encoding for the desired character set are substituted in and a lookup is done. If a match is found that font is used. Otherwise, processing continues on to the next font in the list.

The functions for determining the metrics of a string come in several varieties that can take a number of forms of string input:

16-bit string
Multibyte string
Wide character string

When using functions like `gdk-string-width` that take a `<gchar *>`, if the font is of type 'GDK_FONT_FONT' and is an 8-bit font, then each `<gchar>` indexes the glyphs in the font directly.

For functions taking a `<gchar *>`, if the font is of type 'GDK_FONT_FONT', and is a 16-bit font, then the `<gchar *>` argument is interpreted as a `<guint16 *>` cast to a `<gchar *>` and each `<guint16>` indexes the glyphs in the font directly.

For functions taking a `<gchar *>`, if the font is of type 'GDK_FONT_FONTSET', then the input string is interpreted as a *multibyte* encoded according to the current locale. (A multibyte string is one in which each character may consist of one or more bytes, with different lengths for different characters in the string). They can be converted to and from wide character strings (see below) using `gdk-wcstombs` and `gdk-mbstowcs`.) The string will be rendered using one or more different fonts from the fontset.

For a number of the text-measuring functions, GDK provides a variant (such as `gdk-text-width-wc`) which takes a `<gdk-wchar *>` instead of a `<gchar *>`. The input is then taken to be a wide character string in the encoding of the current locale. (A wide character string is a string in which each character consists of several bytes, and the width of each character in the string is constant.)

GDK provides functions to determine a number of different measurements (metrics) for a given string. (Need diagram here).

descent

left bearing
right bearing
width bearing

The vertical distance from the origin of the drawing opereration to the top of the drawn character.

The vertical distance from the origin of the drawing opereration to the bottom of the drawn character.

The horizontal distance from the origin of the drawing operation to the left-most part of the drawn character.

The horizontal distance from the origin of the drawing operation to the right-most part of the drawn character.

The horizontal distance from the origin of the drawing operation to the correct origin for drawing another string to follow the current one. Depending on the font, this could be greater than or less than the right bearing.

15.2 Usage

`<gdk-font>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`gdk-font-load` (*font_name* `mchars`) ⇒ (*ret* `<gdk-font>`) [Function]

‘`gdk_font_load`’ is deprecated and should not be used in newly-written code.

Loads a font.

The font may be newly loaded or looked up the font in a cache. You should make no assumptions about the initial reference count.

font-name a XLFD describing the font to load.

ret a `<gdk-font>`, or ‘`#f`’ if the font could not be loaded.

`gdk-fontset-load` (*fontset_name* `mchars`) ⇒ (*ret* `<gdk-font>`) [Function]

‘`gdk_fontset_load`’ is deprecated and should not be used in newly-written code.

Loads a fontset.

The fontset may be newly loaded or looked up in a cache. You should make no assumptions about the initial reference count.

fontset-name
: a comma-separated list of XLFDs describing the component fonts of the fontset to load.

ret a `<gdk-font>`, or '`#f`' if the fontset could not be loaded.

`gdk-font-from-description` [Function]
(*font_desc* `<pango-font-description>`) ⇒ (*ret* `<gdk-font>`)

'`gdk_font_from_description`' is deprecated and should not be used in newly-written code.

Load a `<gdk-font>` based on a Pango font description. This font will only be an approximation of the Pango font, and internationalization will not be handled correctly. This function should only be used for legacy code that cannot be easily converted to use Pango. Using Pango directly will produce better results.

font-desc a `<pango-font-description>`.

ret the newly loaded font, or '`#f`' if the font cannot be loaded.

`gdk-font-id` (*self* `<gdk-font>`) ⇒ (*ret* `int`) [Function]

'`gdk_font_id`' is deprecated and should not be used in newly-written code.

Returns the X Font ID for the given font.

font a `<gdk-font>`.

ret the numeric X Font ID

`gdk-string-extents` (*font* `<gdk-font>`) (*string* `mchars`) [Function]
⇒ (*lbearing* `int`) (*rbearing* `int`) (*width* `int`) (*ascent* `int`) (*descent* `int`)

'`gdk_string_extents`' is deprecated and should not be used in newly-written code.

Gets the metrics of a nul-terminated string.

font a `<gdk-font>`.

string the nul-terminated string to measure.

lbearing the left bearing of the string.

rbearing the right bearing of the string.

width the width of the string.

ascent the ascent of the string.

descent the descent of the string.

`gdk-string-width` (*font* `<gdk-font>`) (*string* `mchars`) ⇒ (*ret* `int`) [Function]

'`gdk_string_width`' is deprecated and should not be used in newly-written code.

Determines the width of a nul-terminated string. (The distance from the origin of the string to the point where the next string in a sequence of strings should be drawn) *font* a <gdk-font>

string the nul-terminated string to measure

ret the width of the string in pixels.

gdk-text-width (*font* <gdk-font>) (*text* mchars) (*text_length* int) ⇒ (*ret* int) [Function]

'gdk_text_width' is deprecated and should not be used in newly-written code.

Determines the width of a given string.

font a <gdk-font>

text the text to measure.

text-length the length of the text in bytes.

ret the width of the string in pixels.

gdk-char-width (*font* <gdk-font>) (*character* char) ⇒ (*ret* int) [Function]

'gdk_char_width' is deprecated and should not be used in newly-written code. Use gdk-text-extents instead.

Determines the width of a given character.

font a <gdk-font>

character the character to measure.

ret the width of the character in pixels.

gdk-char-width-wc (*font* <gdk-font>) (*character* unsigned-int32) ⇒ (*ret* int) [Function]

'gdk_char_width_wc' is deprecated and should not be used in newly-written code.

Determines the width of a given wide character. (Encoded in the wide-character encoding of the current locale).

font a <gdk-font>

character the character to measure.

ret the width of the character in pixels.

gdk-string-measure (*font* <gdk-font>) (*string* mchars) ⇒ (*ret* int) [Function]

'gdk_string_measure' is deprecated and should not be used in newly-written code.

Determines the distance from the origin to the rightmost portion of a nul-terminated string when drawn. This is not the correct value for determining the origin of the next portion when drawing text in multiple pieces. See gdk-string-width.

font a <gdk-font>

string the nul-terminated string to measure.

ret the right bearing of the string in pixels.

`gdk-text-measure` (*font* `<gdk-font>`) (*text* `mchars`) (*text_length* `int`) [Function]
⇒ (*ret* `int`)

'`gdk_text_measure`' is deprecated and should not be used in newly-written code.

Determines the distance from the origin to the rightmost portion of a string when drawn. This is not the correct value for determining the origin of the next portion when drawing text in multiple pieces. See `gdk-text-width`.

font a `<gdk-font>`

text the text to measure.

text-length
the length of the text in bytes.

ret the right bearing of the string in pixels.

`gdk-char-measure` (*font* `<gdk-font>`) (*character* `char`) ⇒ (*ret* `int`) [Function]

'`gdk_char_measure`' is deprecated and should not be used in newly-written code.

Determines the distance from the origin to the rightmost portion of a character when drawn. This is not the correct value for determining the origin of the next portion when drawing text in multiple pieces.

font a `<gdk-font>`

character the character to measure.

ret the right bearing of the character in pixels.

`gdk-string-height` (*font* `<gdk-font>`) (*string* `mchars`) ⇒ (*ret* `int`) [Function]

'`gdk_string_height`' is deprecated and should not be used in newly-written code.

Determines the total height of a given nul-terminated string. This value is not generally useful, because you cannot determine how this total height will be drawn in relation to the baseline. See `gdk-string-extents`.

font a `<gdk-font>`

string the nul-terminated string to measure.

ret the height of the string in pixels.

`gdk-text-height` (*font* `<gdk-font>`) (*text* `mchars`) (*text_length* `int`) [Function]
⇒ (*ret* `int`)

'`gdk_text_height`' is deprecated and should not be used in newly-written code.

Determines the total height of a given string. This value is not generally useful, because you cannot determine how this total height will be drawn in relation to the baseline. See `gdk-text-extents`.

font a `<gdk-font>`

text the text to measure.

text-length
the length of the text in bytes.

ret the height of the string in pixels.

`gdk-char-height` (*font* `<gdk-font>`) (*character* `char`) ⇒ (*ret* `int`) [Function]

'`gdk_char_height`' is deprecated and should not be used in newly-written code. Use `gdk-text-extents` instead.

Determines the total height of a given character. This value is not generally useful, because you cannot determine how this total height will be drawn in relation to the baseline. See `gdk-text-extents`.

font a `<gdk-font>`

character the character to measure.

ret the height of the character in pixels.

16 Cursors

Standard and pixmap cursors

16.1 Overview

These functions are used to create and destroy cursors. There is a number of standard cursors, but it is also possible to construct new cursors from pixmaps and pixbufs. There may be limitations as to what kinds of cursors can be constructed on a given display, see `gdk-display-supports-cursor-alpha`, `gdk-display-supports-cursor-color`, `gdk-display-get-default-cursor-size` and `gdk-display-get-maximal-cursor-size`.

Cursors by themselves are not very interesting, they must be be bound to a window for users to see them. This is done with `gdk-window-set-cursor` or by setting the cursor member of the `<gdk-window-attr>` struct passed to `gdk-window-new`.

16.2 Usage

`<gdk-cursor>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`gdk-cursor-new` (*cursor_type* `<gdk-cursor-type>`) [Function]
⇒ (*ret* `<gdk-cursor>`)

Creates a new cursor from the set of builtin cursors for the default display. See `gdk-cursor-new-for-display`.

To make the cursor invisible, use `gdk-cursor-new-from-pixmap` to create a cursor with no pixels in it.

cursor-type
cursor to create

ret a new `<gdk-cursor>`

`gdk-cursor-new-from-pixmap` (*source* `<gdk-pixmap>`) [Function]
(*mask* `<gdk-pixmap>`) (*fg* `<gdk-color>`) (*bg* `<gdk-color>`) (*x* `int`) (*y* `int`)
⇒ (*ret* `<gdk-cursor>`)

Creates a new cursor from a given pixmap and mask. Both the pixmap and mask must have a depth of 1 (i.e. each pixel has only 2 values - on or off). The standard cursor size is 16 by 16 pixels. You can create a bitmap from inline data as in the below example.

```
/* This data is in X bitmap format, and can be created with the 'bitmap'
   utility. */
&#x0023;define cursor1_width 16
&#x0023;define cursor1_height 16
static unsigned char cursor1_bits[] = {
  0x80, 0x01, 0x40, 0x02, 0x20, 0x04, 0x10, 0x08, 0x08, 0x10, 0x04, 0x20,
  0x82, 0x41, 0x41, 0x82, 0x41, 0x82, 0x82, 0x41, 0x04, 0x20, 0x08, 0x10,
```

```
     0x10, 0x08, 0x20, 0x04, 0x40, 0x02, 0x80, 0x01};

   static unsigned char cursor1mask_bits[] = {
     0x80, 0x01, 0xc0, 0x03, 0x60, 0x06, 0x30, 0x0c, 0x18, 0x18, 0x8c, 0x31,
     0xc6, 0x63, 0x63, 0xc6, 0x63, 0xc6, 0xc6, 0x63, 0x8c, 0x31, 0x18, 0x18,
     0x30, 0x0c, 0x60, 0x06, 0xc0, 0x03, 0x80, 0x01};


    GdkCursor *cursor;
    GdkPixmap *source, *mask;
    GdkColor fg = { 0, 65535, 0, 0 }; /* Red. */
    GdkColor bg = { 0, 0, 0, 65535 }; /* Blue. */


    source = gdk_bitmap_create_from_data (NULL, cursor1_bits,
                                          cursor1_width, cursor1_height);
    mask = gdk_bitmap_create_from_data (NULL, cursor1mask_bits,
                                        cursor1_width, cursor1_height);
    cursor = gdk_cursor_new_from_pixmap (source, mask, &fg, &bg, 8, 8);
    gdk_pixmap_unref (source);
    gdk_pixmap_unref (mask);


    gdk_window_set_cursor (widget->window, cursor);
```

source the pixmap specifying the cursor.

mask the pixmap specifying the mask, which must be the same size as *source*.

fg the foreground color, used for the bits in the source which are 1. The color does not have to be allocated first.

bg the background color, used for the bits in the source which are 0. The color does not have to be allocated first.

x the horizontal offset of the 'hotspot' of the cursor.

y the vertical offset of the 'hotspot' of the cursor.

ret a new `<gdk-cursor>`.

`gdk-cursor-new-from-pixbuf` (*display* `<gdk-display>`) (*pixbuf* `<gdk-pixbuf>`) (*x* `int`) (*y* `int`) ⇒ (*ret* `<gdk-cursor>`) [Function]

Creates a new cursor from a pixbuf.

Not all GDK backends support RGBA cursors. If they are not supported, a monochrome approximation will be displayed. The functions `gdk-display-supports-cursor-alpha` and `gdk-display-supports-cursor-color` can be used to determine whether RGBA cursors are supported; `gdk-display-get-default-cursor-size` and `gdk-display-get-maximal-cursor-size` give information about cursor sizes.

On the X backend, support for RGBA cursors requires a sufficently new version of the X Render extension.

display the `<gdk-display>` for which the cursor will be created

pixbuf the `<gdk-pixbuf>` containing the cursor image

x the horizontal offset of the 'hotspot' of the cursor.

y the vertical offset of the 'hotspot' of the cursor.

ret a new `<gdk-cursor>`.

Since 2.4

`gdk-cursor-new-from-name` (*display* `<gdk-display>`) (*name* `mchars`) ⇒ (*ret* `<gdk-cursor>`) [Function]

Creates a new cursor by looking up *name* in the current cursor theme.

display the `<gdk-display>` for which the cursor will be created

name the name of the cursor

ret a new `<gdk-cursor>`, or '`#f`' if there is no cursor with the given name

Since 2.8

`gdk-cursor-new-for-display` (*display* `<gdk-display>`) (*cursor_type* `<gdk-cursor-type>`) ⇒ (*ret* `<gdk-cursor>`) [Function]

Creates a new cursor from the set of builtin cursors. Some useful ones are:

-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-

<gdk-right-ptr> (right-facing arrow)

<gdk-crosshair> (crosshair)

(I-beam)

(busy)

(for moving objects)

(a right-pointing hand)

(a left-pointing hand)

<gdk-left-side> (resize left side)

<gdk-right-side> (resize right side)

<gdk-top-left-corner> (resize northwest corner)

<gdk-top-right-corner> (resize northeast corner)

<gdk-bottom-left-corner> (resize southwest corner)

<gdk-bottom-right-corner> (resize southeast corner)

<gdk-top-side> (resize top side)

<gdk-bottom-side> (resize bottom side)

<gdk-sb-h-double-arrow> (move vertical splitter)

<gdk-sb-v-double-arrow> (move horizontal splitter)

To make the cursor invisible, use `gdk-cursor-new-from-pixmap` to create a cursor with no pixels in it.

display the <gdk-display> for which the cursor will be created

cursor-type
cursor to create

ret a new <gdk-cursor>

Since 2.2

gdk-cursor-get-display (*self* <gdk-cursor>) ⇒ (*ret* <gdk-display>) [Function]

Returns the display on which the <gdk-cursor> is defined.

cursor a <gdk-cursor>.

ret the <gdk-display> associated to *cursor*

Since 2.2

gdk-cursor-get-image (*self* <gdk-cursor>) ⇒ (*ret* <gdk-pixbuf>) [Function]

Returns a <gdk-pixbuf> with the image used to display the cursor.

Note that depending on the capabilities of the windowing system and on the cursor, GDK may not be able to obtain the image data. In this case, '#f' is returned.

cursor a <gdk-cursor>

ret a <gdk-pixbuf> representing *cursor*, or '#f'

Since 2.8

17 Windows

Onscreen display areas in the target window system

17.1 Overview

A `<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>` on the GTK+ level. A `<gtk-window>` is a toplevel window, the thing a user might think of as a "window" with a titlebar and so on; a `<gtk-window>` may contain many `<gdk-window>`. For example, each `<gtk-button>` has a `<gdk-window>` associated with it.

17.2 Usage

`gdk-window-destroy` (*self* `<gdk-window>`) [Function]
`destroy` [Method]

Destroys the window system resources associated with *window* and decrements *window*'s reference count. The window system resources for all children of *window* are also destroyed, but the children's reference counts are not decremented.

Note that a window will not be destroyed automatically when its reference count reaches zero. You must call this function yourself before that happens.

window a `<gdk-window>`

`gdk-window-get-window-type` (*self* `<gdk-window>`) [Function]
⇒ (*ret* `<gdk-window-type>`)
`get-window-type` [Method]

Gets the type of the window. See `<gdk-window-type>`.

window a `<gdk-window>`

ret type of window

`gdk-window-at-pointer` ⇒ (*ret* `<gdk-window>`) (*win_x* `int`) [Function]
(*win_y* `int`)

Obtains the window underneath the mouse pointer, returning the location of that window in *win-x*, *win-y*. Returns '`#f`' if the window under the mouse pointer is not known to GDK (if the window belongs to another application and a `<gdk-window>` hasn't been created for it with `gdk-window-foreign-new`)

NOTE: For multihead-aware widgets or applications use `gdk-display-get-window-at-pointer` instead.

win-x return location for origin of the window under the pointer

win-y return location for origin of the window under the pointer

ret window under the mouse pointer

`gdk-window-show` (*self* `<gdk-window>`) [Function]
`show` [Method]

Like `gdk-window-show-unraised`, but also raises the window to the top of the window stack (moves the window to the front of the Z-order).

This function maps a window so it's visible onscreen. Its opposite is `gdk-window-hide`.

When implementing a `<gtk-widget>`, you should call this function on the widget's `<gdk-window>` as part of the "map" method.

window a `<gdk-window>`

`gdk-window-show-unraised` (*self* `<gdk-window>`) [Function]
`show-unraised` [Method]

Shows a `<gdk-window>` onscreen, but does not modify its stacking order. In contrast, `gdk-window-show` will raise the window to the top of the window stack.

On the X11 platform, in Xlib terms, this function calls `x-map-window` (it also updates some internal GDK state, which means that you can't really use `x-map-window` directly on a GDK window).

window a `<gdk-window>`

`gdk-window-hide` (*self* `<gdk-window>`) [Function]
`hide` [Method]

For toplevel windows, withdraws them, so they will no longer be known to the window manager; for all windows, unmaps them, so they won't be displayed. Normally done automatically as part of `gtk-widget-hide`.

window a `<gdk-window>`

`gdk-window-is-visible` (*self* `<gdk-window>`) ⇒ (*ret* `bool`) [Function]
`is-visible` [Method]

Checks whether the window has been mapped (with `gdk-window-show` or `gdk-window-show-unraised`).

window a `<gdk-window>`

ret '`#t`' if the window is mapped

`gdk-window-is-viewable` (*self* `<gdk-window>`) ⇒ (*ret* `bool`) [Function]
`is-viewable` [Method]

Check if the window and all ancestors of the window are mapped. (This is not necessarily "viewable" in the X sense, since we only check as far as we have GDK window parents, not to the root window.)

window a `<gdk-window>`

ret '`#t`' if the window is viewable

`gdk-window-get-state` (*self* `<gdk-window>`) [Function]
⇒ (*ret* `<gdk-window-state>`)
`get-state` [Method]

Gets the bitwise OR of the currently active window state flags, from the `<gdk-window-state>` enumeration.

window a `<gdk-window>`

ret window state bitfield

`gdk-window-withdraw` (*self* `<gdk-window>`) [Function]
`withdraw` [Method]

Withdraws a window (unmaps it and asks the window manager to forget about it). This function is not really useful as `gdk-window-hide` automatically withdraws toplevel windows before hiding them.

window a toplevel `<gdk-window>`

`gdk-window-iconify` (*self* `<gdk-window>`) [Function]
`iconify` [Method]

Asks to iconify (minimize) *window*. The window manager may choose to ignore the request, but normally will honor it. Using `gtk-window-iconify` is preferred, if you have a `<gtk-window>` widget.

This function only makes sense when *window* is a toplevel window.

window a toplevel `<gdk-window>`

`gdk-window-deiconify` (*self* `<gdk-window>`) [Function]
`deiconify` [Method]

Attempt to deiconify (unminimize) *window*. On X11 the window manager may choose to ignore the request to deiconify. When using GTK+, use `gtk-window-deiconify` instead of the `<gdk-window>` variant. Or better yet, you probably want to use `gtk-window-present`, which raises the window, focuses it, unminimizes it, and puts it on the current desktop.

window a toplevel `<gdk-window>`

`gdk-window-stick` (*self* `<gdk-window>`) [Function]
`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 `<gtk-window>`, `gtk-window-stick` may be more useful.)

On the X11 platform, this function depends on window manager support, so may have no effect with many window managers. However, GDK 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.

window a toplevel `<gdk-window>`

`gdk-window-unstick` (*self* `<gdk-window>`) [Function]
`unstick` [Method]

Reverse operation for `gdk-window-stick`; see `gdk-window-stick`, and `gtk-window-unstick`.

window a toplevel `<gdk-window>`

`gdk-window-maximize` (*self* `<gdk-window>`) [Function]
`maximize` [Method]

Maximizes the window. If the window was already maximized, then this function does nothing.

On X11, asks the window manager to maximize *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, and GDK makes a best effort to get it to happen.

On Windows, reliably maximizes the window.

window a toplevel `<gdk-window>`

gdk-window-unmaximize (*self* `<gdk-window>`) [Function]
unmaximize [Method]

Unmaximizes the window. If the window wasn't maximized, then this function does nothing.

On X11, asks the window manager to unmaximize *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, and GDK makes a best effort to get it to happen.

On Windows, reliably unmaximizes the window.

window a toplevel `<gdk-window>`

gdk-window-fullscreen (*self* `<gdk-window>`) [Function]
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 function does nothing.

On X11, asks the window manager to put *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"; so you can't rely on the fullscreenification actually happening. But it will happen with most standard window managers, and GDK makes a best effort to get it to happen.

window a toplevel `<gdk-window>`

Since 2.2

gdk-window-unfullscreen (*self* `<gdk-window>`) [Function]
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 *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"; so you can't rely on the unfullscreenification actually happening. But it will happen with most standard window managers, and GDK makes a best effort to get it to happen.

window a toplevel `<gdk-window>`

Since 2.2

gdk-window-set-keep-above (*self* `<gdk-window>`) (*setting* `bool`) [Function]
set-keep-above [Method]

Set if *window* must be kept above other windows. If the window was already above, then this function does nothing.

On X11, asks the window manager to keep *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"; so you can't rely on the window being kept above. But it will happen with most standard window managers, and GDK makes a best effort to get it to happen.

window a toplevel `<gdk-window>`

setting whether to keep *window* above other windows

Since 2.4

gdk-window-set-keep-below (*self* `<gdk-window>`) (*setting* `bool`) [Function]
set-keep-below [Method]

Set if *window* must be kept below other windows. If the window was already below, then this function does nothing.

On X11, asks the window manager to keep *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"; so you can't rely on the window being kept below. But it will happen with most standard window managers, and GDK makes a best effort to get it to happen.

window a toplevel `<gdk-window>`

setting whether to keep *window* below other windows

Since 2.4

gdk-window-move (*self* `<gdk-window>`) (*x* `int`) (*y* `int`) [Function]
move [Method]

Repositions a window relative to its parent window. For toplevel windows, window managers may ignore or modify the move; you should probably use `gtk-window-move` on a `<gtk-window>` widget anyway, instead of using GDK functions. For child windows, the move will reliably succeed.

If you're also planning to resize the window, use `gdk-window-move-resize` to both move and resize simultaneously, for a nicer visual effect.

window a `<gdk-window>`

x X coordinate relative to window's parent

y Y coordinate relative to window's parent

gdk-window-resize (*self* `<gdk-window>`) (*width* `int`) (*height* `int`) [Function]
resize [Method]

Resizes *window*; for toplevel windows, asks the window manager to resize the window. The window manager may not allow the resize. When using GTK+, use `gtk-window-resize` instead of this low-level GDK function.

Windows may not be resized below 1x1.

If you're also planning to move the window, use `gdk-window-move-resize` to both move and resize simultaneously, for a nicer visual effect.

window a `<gdk-window>`

width new width of the window

height new height of the window

`gdk-window-move-resize` (*self* `<gdk-window>`) (*x* `int`) (*y* `int`) (*width* `int`) (*height* `int`) [Function]

`move-resize` [Method]

Equivalent to calling `gdk-window-move` and `gdk-window-resize`, 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 `gdk-window-move-resize`.)

window a `<gdk-window>`

x new X position relative to window's parent

y new Y position relative to window's parent

width new width

height new height

`gdk-window-scroll` (*self* `<gdk-window>`) (*dx* `int`) (*dy* `int`) [Function]

`scroll` [Method]

Scroll the contents of *window*, both pixels and children, by the given amount. *window* itself does not move. Portions of the window that the scroll operation brings in from offscreen 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.)

window a `<gdk-window>`

dx Amount to scroll in the X direction

dy Amount to scroll in the Y direction

`gdk-window-move-region` (*self* `<gdk-window>`) (*region* `<gdk-region>`) (*dx* `int`) (*dy* `int`) [Function]

`move-region` [Method]

Move the part of *window* indicated by *region* by *dy* pixels in the Y direction and *dx* pixels in the X direction. The portions of *region* that not covered by the new position of *region* are invalidated.

Child windows are not moved.

window a `<gdk-window>`

region The `<gdk-region>` to move

dx Amount to move in the X direction

dy Amount to move in the Y direction

Since 2.8

`gdk-window-reparent` (*self* `<gdk-window>`) (*new_parent* `<gdk-window>`) (*x* `int`) (*y* `int`) [Function]
`reparent` [Method]

Reparents *window* into the given *new-parent*. The window being reparented will be unmapped as a side effect.

window a `<gdk-window>`

new-parent new parent to move *window* into

x X location inside the new parent

y Y location inside the new parent

`gdk-window-clear` (*self* `<gdk-window>`) [Function]
`clear` [Method]

Clears an entire *window* to the background color or background pixmap.

window a `<gdk-window>`

`gdk-window-clear-area` (*self* `<gdk-window>`) (*x* `int`) (*y* `int`) (*width* `int`) (*height* `int`) [Function]
`clear-area` [Method]

Clears an area of *window* to the background color or background pixmap.

window a `<gdk-window>`

x x coordinate of rectangle to clear

y y coordinate of rectangle to clear

width width of rectangle to clear

height height of rectangle to clear

`gdk-window-clear-area-e` (*self* `<gdk-window>`) (*x* `int`) (*y* `int`) (*width* `int`) (*height* `int`) [Function]
`clear-area-e` [Method]

Like `gdk-window-clear-area`, but also generates an expose event for the cleared area.

This function has a stupid name because it dates back to the mists time, pre-GDK-1.0.

window a `<gdk-window>`

x x coordinate of rectangle to clear

y y coordinate of rectangle to clear

width width of rectangle to clear

height height of rectangle to clear

gdk-window-raise (*self* `<gdk-window>`) [Function]
raise [Method]

Raises *window* to the top of the Z-order (stacking order), so that other windows with the same parent window appear below *window*. This is true whether or not the windows are visible.

If *window* is a toplevel, the window manager may choose to deny the request to move the window in the Z-order, `gdk-window-raise` only requests the restack, does not guarantee it.

window a `<gdk-window>`

gdk-window-lower (*self* `<gdk-window>`) [Function]
lower [Method]

Lowers *window* to the bottom of the Z-order (stacking order), so that other windows with the same parent window appear above *window*. This is true whether or not the other windows are visible.

If *window* is a toplevel, the window manager may choose to deny the request to move the window in the Z-order, `gdk-window-lower` only requests the restack, does not guarantee it.

Note that `gdk-window-show` raises the window again, so don't call this function before `gdk-window-show`. (Try `gdk-window-show-unraised`.)

window a `<gdk-window>`

gdk-window-focus (*self* `<gdk-window>`) (*timestamp* `unsigned-int32`) [Function]
focus [Method]

Sets keyboard focus to *window*. In most cases, `gtk-window-present` should be used on a `<gtk-window>`, rather than calling this function.

window a `<gdk-window>`

timestamp
timestamp of the event triggering the window focus

gdk-window-register-dnd (*self* `<gdk-window>`) [Function]
register-dnd [Method]

Registers a window as a potential drop destination.

window a `<gdk-window>`.

gdk-window-begin-resize-drag (*self* `<gdk-window>`) [Function]
(*edge* `<gdk-window-edge>`) (*button* `int`) (*root_x* `int`) (*root_y* `int`)
(*timestamp* `unsigned-int32`)
begin-resize-drag [Method]

Begins a window resize operation (for a toplevel window). You might use this function to implement a "window resize grip," for example; in fact `<gtk-statusbar>` uses it. The function works best with window managers that support the Extended Window Manager Hints (`http://www.freedesktop.org/Standards/wm-spec`), but has a fallback implementation for other window managers.

window a toplevel `<gdk-window>`

edge the edge or corner from which the drag is started

button the button being used to drag

root-x root window X coordinate of mouse click that began the drag

root-y root window Y coordinate of mouse click that began the drag

timestamp
timestamp of mouse click that began the drag (use `gdk-event-get-time`)

`gdk-window-begin-move-drag` (*self* `<gdk-window>`) (*button* `int`) (*root_x* `int`) (*root_y* `int`) (*timestamp* `unsigned-int32`) [Function]

`begin-move-drag` [Method]

Begins a window move operation (for a toplevel window). You might use this function to implement a "window move grip," for example. The function works best with window managers that support the Extended Window Manager Hints (`http://www.freedesktop.org/Standards/wm-spec`), but has a fallback implementation for other window managers.

window a toplevel `<gdk-window>`

button the button being used to drag

root-x root window X coordinate of mouse click that began the drag

root-y root window Y coordinate of mouse click that began the drag

timestamp
timestamp of mouse click that began the drag

`gdk-window-begin-paint-rect` (*self* `<gdk-window>`) (*rectangle* `<gdk-rectangle>`) [Function]

`begin-paint-rect` [Method]

A convenience wrapper around `gdk-window-begin-paint-region` which creates a rectangular region for you. See `gdk-window-begin-paint-region` for details.

window a `<gdk-window>`

rectangle rectangle you intend to draw to

`gdk-window-begin-paint-region` (*self* `<gdk-window>`) (*region* `<gdk-region>`) [Function]

`begin-paint-region` [Method]

Indicates that you are beginning the process of redrawing *region*. A backing store (offscreen buffer) large enough to contain *region* will be created. The backing store will be initialized with the background color or background pixmap for *window*. Then, all drawing operations performed on *window* will be diverted to the backing store. When you call `gdk-window-end-paint`, the backing store will be copied to *window*, making it visible onscreen. Only the part of *window* contained in *region* will be modified; that is, drawing operations are clipped to *region*.

The net result of all this is to remove flicker, because the user sees the finished product appear all at once when you call `gdk-window-end-paint`. If you draw to *window* directly without calling `gdk-window-begin-paint-region`, the user may see flicker as individual drawing operations are performed in sequence. The clipping and background-initializing features of `gdk-window-begin-paint-region` are conveniences for the programmer, so you can avoid doing that work yourself.

When using GTK+, the widget system automatically places calls to `gdk-window-begin-paint-region` and `gdk-window-end-paint` around emissions of the expose_event signal. That is, if you're writing an expose event handler, you can assume that the exposed area in `<gdk-event-expose>` has already been cleared to the window background, is already set as the clip region, and already has a backing store. Therefore in most cases, application code need not call `gdk-window-begin-paint-region`. (You can disable the automatic calls around expose events on a widget-by-widget basis by calling `gtk-widget-set-double-buffered`.)

If you call this function multiple times before calling the matching `gdk-window-end-paint`, the backing stores are pushed onto a stack. `gdk-window-end-paint` copies the topmost backing store onscreen, subtracts the topmost region from all other regions in the stack, and pops the stack. All drawing operations affect only the topmost backing store in the stack. One matching call to `gdk-window-end-paint` is required for each call to `gdk-window-begin-paint-region`.

window a `<gdk-window>`

region region you intend to draw to

`gdk-window-end-paint` (*self* `<gdk-window>`) [Function]
`end-paint` [Method]

Indicates that the backing store created by the most recent call to `gdk-window-begin-paint-region` should be copied onscreen and deleted, leaving the next-most-recent backing store or no backing store at all as the active paint region. See `gdk-window-begin-paint-region` for full details. It is an error to call this function without a matching `gdk-window-begin-paint-region` first.

window a `<gdk-window>`

`gdk-window-invalidate-rect` (*self* `<gdk-window>`) [Function]
(*rect* `<gdk-rectangle>`) (*invalidate_children* `bool`)
`invalidate-rect` [Method]

A convenience wrapper around `gdk-window-invalidate-region` which invalidates a rectangular region. See `gdk-window-invalidate-region` for details.

window a `<gdk-window>`

rect rectangle to invalidate

invalidate-children
whether to also invalidate child windows

`gdk-window-invalidate-region` (*self* `<gdk-window>`) [Function]
(*region* `<gdk-region>`) (*invalidate_children* `bool`)
`invalidate-region` [Method]

Adds *region* to the update area for *window*. The update area is the region that needs to be redrawn, or "dirty region." The call `gdk-window-process-updates` sends one or more expose events to the window, which together cover the entire update area.

An application would normally redraw the contents of *window* in response to those expose events.

GDK will call `gdk-window-process-all-updates` on your behalf whenever your program returns to the main loop and becomes idle, so normally there's no need to do that manually, you just need to invalidate regions that you know should be redrawn.

The *invalidate-children* parameter controls whether the region of each child window that intersects *region* will also be invalidated. If '`#f`', then the update area for child windows will remain unaffected. See gdk_window_invalidate_maybe_recurse if you need fine grained control over which children are invalidated.

window a `<gdk-window>`

region a `<gdk-region>`

invalidate-children
'`#t`' to also invalidate child windows

`gdk-window-get-update-area` (*self* `<gdk-window>`) [Function]
⇒ (*ret* `<gdk-region>`)

`get-update-area` [Method]

Transfers ownership of the update area from *window* to the caller of the function. That is, after calling this function, *window* will no longer have an invalid/dirty region; the update area is removed from *window* and handed to you. If a window has no update area, `gdk-window-get-update-area` returns '`#f`'. You are responsible for calling `gdk-region-destroy` on the returned region if it's non-'`#f`'.

window a `<gdk-window>`

ret the update area for *window*

`gdk-window-freeze-updates` (*self* `<gdk-window>`) [Function]

`freeze-updates` [Method]

Temporarily freezes a window such that it won't receive expose events. The window will begin receiving expose events again when `gdk-window-thaw-updates` is called. If `gdk-window-freeze-updates` has been called more than once, `gdk-window-thaw-updates` must be called an equal number of times to begin processing exposes.

window a `<gdk-window>`

`gdk-window-thaw-updates` (*self* `<gdk-window>`) [Function]

`thaw-updates` [Method]

Thaws a window frozen with `gdk-window-freeze-updates`.

window a `<gdk-window>`

`gdk-window-process-all-updates` [Function]

Calls `gdk-window-process-updates` for all windows (see `<gdk-window>`) in the application.

`gdk-window-process-updates` (*self* `<gdk-window>`) [Function]
(*update_children* `bool`)

`process-updates` [Method]

Sends one or more expose events to *window*. The areas in each expose event will cover the entire update area for the window (see `gdk-window-invalidate-region` for details). Normally GDK calls `gdk-window-process-all-updates` on your behalf, so there's no need to call this function unless you want to force expose events to be delivered immediately and synchronously (vs. the usual case, where GDK delivers them in an idle handler). Occasionally this is useful to produce nicer scrolling behavior, for example.

window a `<gdk-window>`

update-children
whether to also process updates for child windows

gdk-window-set-debug-updates (*setting* `bool`) [Function]

With update debugging enabled, calls to `gdk-window-invalidate-region` clear the invalidated region of the screen to a noticeable color, and GDK pauses for a short time before sending exposes to windows during `gdk-window-process-updates`. 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 GTK 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 GTK+ programs enables this debug option at application startup time. That's usually more useful than calling `gdk-window-set-debug-updates` yourself, though you might want to use this function to enable updates sometime after application startup time.

setting '`#t`' to turn on update debugging

gdk-window-configure-finished (*self* `<gdk-window>`) [Function]
configure-finished [Method]

Signal to the window system that the application has finished handling 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 `gdk-window-enable-synchronized-configure` was called previously.

window a toplevel `<gdk-window>`

Since 2.6

gdk-window-set-override-redirect (*self* `<gdk-window>`) [Function]
(*override_redirect* `bool`)
set-override-redirect [Method]

An override redirect 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.

window a toplevel `<gdk-window>`

override-redirect
'#t' if window should be override redirect

`gdk-window-set-accept-focus` (*self* `<gdk-window>`) [Function]
(*accept_focus* `bool`)
`set-accept-focus` [Method]

Setting *accept-focus* to '#f' hints 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.

window a toplevel `<gdk-window>`

accept-focus
'#t' if the window should receive input focus

Since 2.4

`gdk-window-set-focus-on-map` (*self* `<gdk-window>`) [Function]
(*focus_on_map* `bool`)
`set-focus-on-map` [Method]

Setting *focus-on-map* to '#f' hints the desktop environment that the window doesn't want to receive input focus when it is mapped. focus_on_map should be turned off for windows that aren't triggered interactively (such as popups from network activity).

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.

window a toplevel `<gdk-window>`

focus-on-map
'#t' if the window should receive input focus when mapped

Since 2.6

`gdk-window-shape-combine-mask` (*self* `<gdk-window>`) [Function]
(*mask* `<gdk-drawable>`) (*x* `int`) (*y* `int`)
`shape-combine-mask` [Method]

Applies a shape mask to *window*. Pixels in *window* corresponding to set bits in the *mask* will be visible; pixels in *window* corresponding to unset bits in the *mask* will be transparent. This gives a non-rectangular window.

If *mask* is '#f', 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.

On the Win32 platform the functionality is always present.

This function works on both toplevel and child windows.

window a <gdk-window>

mask shape mask

x X position of shape mask with respect to *window*

y Y position of shape mask with respect to *window*

`gdk-window-shape-combine-region` (*self* `<gdk-window>`) (*shape_region* `<gdk-region>`) (*offset_x* `int`) (*offset_y* `int`) [Function]

`shape-combine-region` [Method]

Makes pixels in *window* outside *shape-region* be transparent, so that the window may be nonrectangular. See also `gdk-window-shape-combine-mask` to use a bitmap as the mask.

If *shape-region* is '`#f`', the shape will be unset, so the whole window will be opaque again. *offset-x* and *offset-y* are ignored if *shape-region* is '`#f`'.

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.

On the Win32 platform, this functionality is always present.

This function works on both toplevel and child windows.

window a <gdk-window>

shape-region
region of window to be non-transparent

offset-x X position of *shape-region* in *window* coordinates

offset-y Y position of *shape-region* in *window* coordinates

`gdk-window-set-child-shapes` (*self* `<gdk-window>`) [Function]

`set-child-shapes` [Method]

Sets the shape mask of *window* to the union of shape masks for all children of *window*, ignoring the shape mask of *window* itself. Contrast with `gdk-window-merge-child-shapes` which includes the shape mask of *window* in the masks to be merged.

window a <gdk-window>

`gdk-window-merge-child-shapes` (*self* `<gdk-window>`) [Function]

`merge-child-shapes` [Method]

Merges the shape masks for any child windows into the shape mask for *window*. i.e. the union of all masks for *window* and its children will become the new mask for *window*. See `gdk-window-shape-combine-mask`.

This function is distinct from `gdk-window-set-child-shapes` because it includes *window*'s shape mask in the set of shapes to be merged.

window a <gdk-window>

gdk-window-input-shape-combine-mask (*self* <gdk-window>) (*mask* <gdk-drawable>) (*x* int) (*y* int) [Function]

input-shape-combine-mask [Method]

Like `gdk-window-shape-combine-mask`, but the shape applies only to event handling. Mouse events which happen while the pointer position corresponds to an unset bit in the mask will be passed on the window below *window*.

An input shape is typically used with RGBA windows. The alpha channel of the window defines which pixels are invisible and allows for nicely antialiased borders, and the input shape controls where the window is "clickable".

On the X11 platform, this requires version 1.1 of the shape extension.

On the Win32 platform, this functionality is not present and the function does nothing.

window a `<gdk-window>`

mask shape mask

x X position of shape mask with respect to *window*

y Y position of shape mask with respect to *window*

Since 2.10

gdk-window-set-child-input-shapes (*self* <gdk-window>) [Function]

set-child-input-shapes [Method]

Sets the input shape mask of *window* to the union of input shape masks for all children of *window*, ignoring the input shape mask of *window* itself. Contrast with `gdk-window-merge-child-input-shapes` which includes the input shape mask of *window* in the masks to be merged.

window a `<gdk-window>`

Since 2.10

gdk-window-merge-child-input-shapes (*self* <gdk-window>) [Function]

merge-child-input-shapes [Method]

Merges the input shape masks for any child windows into the input shape mask for *window*. i.e. the union of all input masks for *window* and its children will become the new input mask for *window*. See `gdk-window-input-shape-combine-mask`.

This function is distinct from `gdk-window-set-child-input-shapes` because it includes *window*'s input shape mask in the set of shapes to be merged.

window a `<gdk-window>`

Since 2.10

gdk-window-set-static-gravities (*self* <gdk-window>) (*use_static* bool) ⇒ (*ret* bool) [Function]

set-static-gravities [Method]

Set the bit gravity of the given window to static, and flag it so all children get static subwindow gravity. This is used if you are implementing scary features that involve deep knowledge of the windowing system. Don't worry about it unless you have to.

window a `<gdk-window>`

use-static '`#t`' to turn on static gravity

ret '`#t`' if the server supports static gravity

`gdk-window-set-hints` (*self* `<gdk-window>`) (*x* `int`) (*y* `int`) (*min_width* `int`) (*min_height* `int`) (*max_width* `int`) (*max_height* `int`) (*flags* `int`) [Function]

`set-hints` [Method]

'`gdk_window_set_hints`' is deprecated and should not be used in newly-written code.

This function is broken and useless and you should ignore it. If using GTK+, use functions such as `gtk-window-resize`, `gtk-window-set-size-request`, `gtk-window-move`, `gtk-window-parse-geometry`, and `gtk-window-set-geometry-hints`, depending on what you're trying to do.

If using GDK directly, use `gdk-window-set-geometry-hints`.

window a `<gdk-window>`

x ignored field, does not matter

y ignored field, does not matter

min-width minimum width hint

min-height minimum height hint

max-width max width hint

max-height max height hint

flags logical OR of GDK_HINT_POS, GDK_HINT_MIN_SIZE, and/or GDK_HINT_MAX_SIZE

`gdk-window-set-title` (*self* `<gdk-window>`) (*title* `mchars`) [Function]

`set-title` [Method]

Sets the title of a toplevel window, to be displayed in the titlebar. If you haven't explicitly set the icon name for the window (using `gdk-window-set-icon-name`), the icon name will be set to *title* as well. *title* must be in UTF-8 encoding (as with all user-readable strings in GDK/GTK+). *title* may not be '`#f`'.

window a toplevel `<gdk-window>`

title title of *window*

`gdk-window-set-background` (*self* `<gdk-window>`) (*color* `<gdk-color>`) [Function]

`set-background` [Method]

Sets the background color of *window*. (However, when using GTK+, set the background of a widget with `gtk-widget-modify-bg` - if you're an application - or `gtk-style-set-background` - if you're implementing a custom widget.)

The *color* must be allocated; `gdk-rgb-find-color` is the best way to allocate a color.

See also `gdk-window-set-back-pixmap`.

window a `<gdk-window>`

color an allocated `<gdk-color>`

`gdk-window-set-back-pixmap` (*self* `<gdk-window>`) (*pixmap* `<gdk-pixmap>`) (*parent_relative* `bool`) [Function]

`set-back-pixmap` [Method]

Sets the background pixmap of *window*. May also be used to set a background of "None" on *window*, by setting a background pixmap of '`#f`'. A background pixmap will be tiled, positioning the first tile at the origin of *window*, or if *parent-relative* is '`#t`', 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).

A background pixmap of '`#f`' means that the window will have no background. A window with no background 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 `gdk-window-clear`.

window a `<gdk-window>`

pixmap a `<gdk-pixmap>`, or '`#f`'

parent-relative
whether the tiling origin is at the origin of *window*'s parent

`gdk-window-set-cursor` (*self* `<gdk-window>`) (*cursor* `<gdk-cursor>`) [Function]

`set-cursor` [Method]

Sets the mouse pointer for a `<gdk-window>`. Use `gdk-cursor-new` or `gdk-cursor-new-from-pixmap` to create the cursor. To make the cursor invisible, use `gdk-cursor-new-from-pixmap` to create a cursor with no pixels in it. Passing '`#f`' for the *cursor* argument to `gdk-window-set-cursor` means that *window* will use the cursor of its parent window. Most windows should use this default.

window a `<gdk-window>`

cursor a cursor

`gdk-window-get-geometry` (*self* `<gdk-window>`) ⇒ (*x* `int`) (*y* `int`) (*width* `int`) (*height* `int`) (*depth* `int`) [Function]

`get-geometry` [Method]

Any of the return location arguments to this function may be '`#f`', if you aren't interested in getting the value of that field.

The X and Y coordinates returned are relative to the parent window of *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 *window*; this may be out-of-sync with the position of *window* delivered in the most-recently-processed `<gdk-event-configure>`. `gdk-window-get-position` in contrast gets the position from the most recent configure event.

> If *window* is not a toplevel, it is *much* better to call `gdk-window-get-position` and `gdk-drawable-get-size` instead, because it avoids the roundtrip to the X server and because `gdk-drawable-get-size` supports the full 32-bit coordinate space, whereas `gdk-window-get-geometry` is restricted to the 16-bit coordinates of X11.

window a `<gdk-window>`

x return location for X coordinate of window (relative to its parent)

y return location for Y coordinate of window (relative to its parent)

width return location for width of window

height return location for height of window

depth return location for bit depth of window

`gdk-window-set-icon-list` (*self* `<gdk-window>`) (*pixbufs* `glist-of`) [Function]
`set-icon-list` [Method]

Sets a list of icons for the window. 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.

window The `<gdk-window>` toplevel window to set the icon of.

pixbufs A list of pixbufs, of different sizes.

`gdk-window-set-modal-hint` (*self* `<gdk-window>`) (*modal* `bool`) [Function]
`set-modal-hint` [Method]

The application can use this hint to tell the window manager that a certain window has modal behaviour. The window manager can use this information to handle modal windows in a special way.

You should only use this on windows for which you have previously called `gdk-window-set-transient-for`

window A toplevel `<gdk-window>`

modal '`#t`' if the window is modal, '`#f`' otherwise.

`gdk-window-set-type-hint` (*self* `<gdk-window>`) [Function]
(*hint* `<gdk-window-type-hint>`)
`set-type-hint` [Method]

The application can use this call to provide a hint to the window manager about the functionality of a window. The window manager can use this information when determining the decoration and behaviour of the window.

The hint must be set before the window is mapped.

window A toplevel `<gdk-window>`

hint A hint of the function this window will have

gdk-window-get-type-hint (*self* `<gdk-window>`) [Function]
⇒ (*ret* `<gdk-window-type-hint>`)

get-type-hint [Method]

This function returns the type hint set for a window.

window A toplevel `<gdk-window>`

ret The type hint set for *window*

Since 2.10

gdk-window-set-skip-taskbar-hint (*self* `<gdk-window>`) [Function]
(*skips_taskbar* `bool`)

set-skip-taskbar-hint [Method]

Toggles whether a window should appear in a task list or window list. If a window's semantic type as specified with `gdk-window-set-type-hint` already fully describes the window, this function should *not* be called in addition, instead you should allow the window to be treated according to standard policy for its semantic type.

window a toplevel `<gdk-window>`

skips-taskbar
'`#t`' to skip the taskbar

Since 2.2

gdk-window-set-skip-pager-hint (*self* `<gdk-window>`) [Function]
(*skips_pager* `bool`)

set-skip-pager-hint [Method]

Toggles whether a window should appear in a pager (workspace switcher, or other desktop utility program that displays a small thumbnail representation of the windows on the desktop). If a window's semantic type as specified with `gdk-window-set-type-hint` already fully describes the window, this function should *not* be called in addition, instead you should allow the window to be treated according to standard policy for its semantic type.

window a toplevel `<gdk-window>`

skips-pager
'`#t`' to skip the pager

Since 2.2

gdk-window-set-urgency-hint (*self* `<gdk-window>`) (*urgent* `bool`) [Function]

set-urgency-hint [Method]

Toggles whether a window needs the user's urgent attention.

window a toplevel `<gdk-window>`

urgent '`#t`' if the window is urgent

Since 2.8

`gdk-window-get-position` (*self* `<gdk-window>`) ⇒ (*x* `int`) (*y* `int`) [Function]
`get-position` [Method]

Obtains the position of the window as reported in the most-recently-processed `<gdk-event-configure>`. Contrast with `gdk-window-get-geometry` which queries the X server for the current window position, regardless of which events have been received or processed.

The position coordinates are relative to the window's parent window.

window a `<gdk-window>`

x X coordinate of window

y Y coordinate of window

`gdk-window-get-root-origin` (*self* `<gdk-window>`) ⇒ (*x* `int`) (*y* `int`) [Function]
`get-root-origin` [Method]

Obtains the top-left corner of the window manager frame in root window coordinates.

window a toplevel `<gdk-window>`

x return location for X position of window frame

y return location for Y position of window frame

`gdk-window-get-origin` (*self* `<gdk-window>`) ⇒ (*ret* `int`) (*x* `int`) (*y* `int`) [Function]
`get-origin` [Method]

Obtains the position of a window in root window coordinates. (Compare with `gdk-window-get-position` and `gdk-window-get-geometry` which return the position of a window relative to its parent window.)

window a `<gdk-window>`

x return location for X coordinate

y return location for Y coordinate

ret not meaningful, ignore

`gdk-window-get-deskrelative-origin` (*self* `<gdk-window>`) ⇒ (*ret* `bool`) (*x* `int`) (*y* `int`) [Function]
`get-deskrelative-origin` [Method]

'`gdk_window_get_deskrelative_origin`' is deprecated and should not be used in newly-written code.

This gets the origin of a `<gdk-window>` relative to an Enlightenment-window-manager desktop. As long as you don't assume that the user's desktop/workspace covers the entire root window (i.e. you don't assume that the desktop begins at root window coordinate 0,0) this function is not necessary. It's deprecated for that reason.

window a toplevel `<gdk-window>`

x return location for X coordinate

y return location for Y coordinate

ret not meaningful

gdk-window-get-parent (*self* <gdk-window>) ⇒ (*ret* <gdk-window>) [Function]
get-parent [Method]

Obtains the parent of *window*, as known to GDK. Does not query the X server; thus this returns the parent as passed to `gdk-window-new`, not the actual parent. This should never matter unless you're using Xlib calls mixed with GDK calls on the X11 platform. It may also matter for toplevel windows, because the window manager may choose to reparent them.

window a `<gdk-window>`

ret parent of *window*

gdk-window-get-toplevel (*self* <gdk-window>) ⇒ (*ret* <gdk-window>) [Function]
get-toplevel [Method]

Gets the toplevel window that's an ancestor of *window*.

window a `<gdk-window>`

ret the toplevel window containing *window*

gdk-window-get-children (*self* <gdk-window>) ⇒ (*ret* glist-of) [Function]
get-children [Method]

Gets the list of children of *window* known to GDK. This function only returns children created via GDK, so for example it's useless when used with the root window; it only returns windows an application created itself.

The returned list must be freed, but the elements in the list need not be.

window a `<gdk-window>`

ret list of child windows inside *window*

gdk-window-get-events (*self* <gdk-window>) ⇒ (*ret* <gdk-event-mask>) [Function]
get-events [Method]

Gets the event mask for *window*. See `gdk-window-set-events`.

window a `<gdk-window>`

ret event mask for *window*

gdk-window-set-events (*self* <gdk-window>) (*event_mask* <gdk-event-mask>) [Function]
set-events [Method]

The event mask for a window determines which events will be reported for that window. For example, an event mask including `<gdk-button-press-mask>` means the window should report button press events. The event mask is the bitwise OR of values from the `<gdk-event-mask>` enumeration.

window a `<gdk-window>`

event-mask
event mask for *window*

`gdk-window-set-icon` (*self* `<gdk-window>`) (*icon_window* `<gdk-window>`) (*pixmap* `<gdk-pixmap>`) (*mask* `<gdk-drawable>`) [Function]
`set-icon` [Method]

Sets the icon of *window* as a pixmap or window. If using GTK+, investigate `gtk-window-set-default-icon-list` first, and then `gtk-window-set-icon-list` and `gtk-window-set-icon`. If those don't meet your needs, look at `gdk-window-set-icon-list`. Only if all those are too high-level do you want to fall back to `gdk-window-set-icon`.

window a toplevel `<gdk-window>`

icon-window
a `<gdk-window>` to use for the icon, or '`#f`' to unset

pixmap a `<gdk-pixmap>` to use as the icon, or '`#f`' to unset

mask a 1-bit pixmap (`<gdk-bitmap>`) to use as mask for *pixmap*, or '`#f`' to have none

`gdk-window-set-icon-name` (*self* `<gdk-window>`) (*name* `mchars`) [Function]
`set-icon-name` [Method]

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 function, if you like.

window a toplevel `<gdk-window>`

name name of window while iconified (minimized)

`gdk-window-set-transient-for` (*self* `<gdk-window>`) (*parent* `<gdk-window>`) [Function]
`set-transient-for` [Method]

Indicates to the window manager that *window* is a transient dialog associated with the application window *parent*. This allows the window manager to do things like center *window* on *parent* and keep *window* above *parent*.

See `gtk-window-set-transient-for` if you're using `<gtk-window>` or `<gtk-dialog>`.

window a toplevel `<gdk-window>`

parent another toplevel `<gdk-window>`

`gdk-window-set-role` (*self* `<gdk-window>`) (*role* `mchars`) [Function]
`set-role` [Method]

When using GTK+, typically you should use `gtk-window-set-role` 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.

window a toplevel `<gdk-window>`

role a string indicating its role

gdk-window-set-group (*self* `<gdk-window>`) (*leader* `<gdk-window>`) [Function]
set-group [Method]

Sets the group leader window for *window*. By default, GDK sets the group leader for all toplevel windows to a global window implicitly created by GDK. With this function 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/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.

window a toplevel `<gdk-window>`

leader group leader window, or '`#f`' to restore the default group leader window

gdk-window-get-group (*self* `<gdk-window>`) ⇒ (*ret* `<gdk-window>`) [Function]
get-group [Method]

Returns the group leader window for *window*. See `gdk-window-set-group`.

window a toplevel `<gdk-window>`

ret the group leader window for *window*

Since 2.4

gdk-window-set-decorations (*self* `<gdk-window>`) (*decorations* `<gdk-wm-decoration>`) [Function]
set-decorations [Method]

"Decorations" are the features the window manager adds to a toplevel `<gdk-window>`. This function 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 `gtk-window-set-decorated` on a `<gtk-window>` instead of using the GDK function directly.

The *decorations* argument is the logical OR of the fields in the `<gdk-wm-decoration>` enumeration. If `<gdk-decor-all>` is included in the mask, the other bits indicate which decorations should be turned off. If `<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.

window a toplevel `<gdk-window>`

decorations
decoration hint mask

`gdk-window-set-functions` (*self* `<gdk-window>`) (*functions* `<gdk-wm-function>`) [Function]

`set-functions` [Method]

Sets hints about the window management functions to make available via buttons on the window frame.

On the X backend, this function sets the traditional Motif window manager hint for this purpose. However, few window managers do anything reliable or interesting with this hint. Many ignore it entirely.

The *functions* argument is the logical OR of values from the `<gdk-wm-function>` enumeration. If the bitmask includes `<gdk-func-all>`, then the other bits indicate which functions to disable; if it doesn't include `<gdk-func-all>`, it indicates which functions to enable.

window a toplevel `<gdk-window>`

functions bitmask of operations to allow on *window*

`gdk-window-get-toplevels` ⇒ (*ret* `glist-of`) [Function]

Obtains a list of all toplevel windows known to GDK on the default screen (see `gdk-screen-get-toplevel-windows`). A toplevel window is a child of the root window (see `gdk-get-default-root-window`).

The returned list should be freed with `g-list-free`, but its elements need not be freed.

ret list of toplevel windows, free with `g-list-free`

`gdk-get-default-root-window` ⇒ (*ret* `<gdk-window>`) [Function]

Obtains the root window (parent all other windows are inside) for the default display and screen.

ret the default root window

18 Events

Functions for handling events from the window system

18.1 Overview

This section describes functions dealing with events from the window system.

In GTK+ applications the events are handled automatically in `gtk-main-do-event` and passed on to the appropriate widgets, so these functions are rarely needed. Though some of the fields in the Event Structures are useful.

18.2 Usage

`gdk-events-pending` ⇒ (*ret* `bool`) [Function]

Checks if any events are ready to be processed for any display.

ret '`#t`' if any events are pending.

`gdk-event-peek` ⇒ (*ret* `<gdk-event>`) [Function]

If there is an event waiting in the event queue of some open display, returns a copy of it. See `gdk-display-peek-event`.

ret a copy of the first `<gdk-event>` on some event queue, or '`#f`' if no events are in any queues. The returned `<gdk-event>` should be freed with `gdk-event-free`.

`gdk-event-get` ⇒ (*ret* `<gdk-event>`) [Function]

Checks all open displays for a `<gdk-event>` to process,to be processed on, fetching events from the windowing system if necessary. See `gdk-display-get-event`.

ret the next `<gdk-event>` to be processed, or '`#f`' if no events are pending. The returned `<gdk-event>` should be freed with `gdk-event-free`.

`gdk-event-get-graphics-expose` (*window* `<gdk-window>`) ⇒ (*ret* `<gdk-event>`) [Function]

Waits for a GraphicsExpose or NoExpose event from the X server. This is used in the `<gtk-text>` and `<gtk-clist>` widgets in GTK+ to make sure any GraphicsExpose events are handled before the widget is scrolled.

window the `<gdk-window>` to wait for the events for.

ret a `<gdk-event-expose>` if a GraphicsExpose was received, or '`#f`' if a NoExpose event was received.

`gdk-event-put` (*self* `<gdk-event>`) [Function]

Appends a copy of the given event onto the front of the event queue for event->any.window's display, or the default event queue if event->any.window is '`#f`'. See `gdk-display-put-event`.

event a `<gdk-event>`.

gdk-event-copy (*self* <gdk-event>) ⇒ (*ret* <gdk-event>) [Function]

Copies a <gdk-event>, copying or incrementing the reference count of the resources associated with it (e.g. <gdk-window>'s and strings).

event a <gdk-event>

ret a copy of *event*. The returned <gdk-event> should be freed with gdk-event-free.

gdk-event-get-time (*self* <gdk-event>) ⇒ (*ret* unsigned-int32) [Function]

Returns the time stamp from *event*, if there is one; otherwise returns <gdk-current-time>. If *event* is '#f', returns <gdk-current-time>.

event a <gdk-event>

ret time stamp field from *event*

gdk-event-get-axis (*self* <gdk-event>) (*axis_use* <gdk-axis-use>) [Function]
⇒ (*ret* bool) (*value* double)

Extract the axis value for a particular axis use from an event structure.

event a <gdk-event>

axis-use the axis use to look for

value location to store the value found

ret '#t' if the specified axis was found, otherwise '#f'

gdk-event-get-coords (*self* <gdk-event>) ⇒ (*ret* bool) [Function]
(*x_win* double) (*y_win* double)

Extract the event window relative x/y coordinates from an event.

event a <gdk-event>

x-win location to put event window x coordinate

y-win location to put event window y coordinate

ret '#t' if the event delivered event window coordinates

gdk-event-get-root-coords (*self* <gdk-event>) ⇒ (*ret* bool) [Function]
(*x_root* double) (*y_root* double)

Extract the root window relative x/y coordinates from an event.

event a <gdk-event>

x-root location to put root window x coordinate

y-root location to put root window y coordinate

ret '#t' if the event delivered root window coordinates

gdk-get-show-events ⇒ (*ret* bool) [Function]

Gets whether event debugging output is enabled.

ret '#t' if event debugging output is enabled.

`gdk-set-show-events` (*show_events* `bool`) [Function]

Sets whether a trace of received events is output. Note that GTK+ must be compiled with debugging (that is, configured using the "–enable-debug") option) to use this option.

show-events
: '`#t`' to output event debugging information.

`gdk-event-get-screen` (*self* `<gdk-event>`) ⇒ (*ret* `<gdk-screen>`) [Function]

Returns the screen for the event. The screen is typically the screen for '`event->any.window`', but for events such as mouse events, it is the screen where the pointer was when the event occurs - that is, the screen which has the root window to which '`event->motion.x_root`' and '`event->motion.y_root`' are relative.

event a `<gdk-event>`

ret the screen for the event

Since 2.2

19 Event Structures

Data structures specific to each type of event

19.1 Overview

The event structs contain data specific to each type of event in GDK.

A common mistake is to forget to set the event mask of a widget so that the required events are received. See `gtk-widget-set-events`.

19.2 Usage

`<gdk-event-any>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`<gdk-event-key>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`<gdk-event-button>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`<gdk-event-scroll>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`<gdk-event-motion>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`<gdk-event-expose>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`<gdk-event-visibility>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`<gdk-event-crossing>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`<gdk-event-focus>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`<gdk-event-configure>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`<gdk-event-property>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`<gdk-event-selection>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`<gdk-event-dnd>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`<gdk-event-proximity>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`<gdk-event-client>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`<gdk-event-no-expose>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`<gdk-event-window-state>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

`<gdk-event-setting>` [Class]

Derives from `<gboxed>`.

This class defines no direct slots.

20 Key Values

Functions for manipulating keyboard codes

20.1 Overview

Key values are the codes which are sent whenever a key is pressed or released. They appear in the field of the `<gdk-event-key>` structure, which is passed to signal handlers for the "key-press-event" and "key-release-event" signals. The complete list of key values can be found in the '`<gdk/gdkkeysyms.h>`' header file. '`<gdk/gdkkeysyms.h>`' is not included in '`<gtk/gtk.h>`', it must be included independently, because the file is quite large.

Key values can be converted into a string representation using `gdk-keyval-name`. The reverse function, converting a string to a key value, is provided by `gdk-keyval-from-name`.

The case of key values can be determined using `gdk-keyval-is-upper` and `gdk-keyval-is-lower`. Key values can be converted to upper or lower case using `gdk-keyval-to-upper` and `gdk-keyval-to-lower`.

When it makes sense, key values can be converted to and from Unicode characters with `gdk-keyval-to-unicode` and `gdk-unicode-to-keyval`.

One `<gdk-keymap>` object exists for each user display. `gdk-keymap-get-default` returns the `<gdk-keymap>` for the default display; to obtain keymaps for other displays, use `gdk-keymap-get-for-display`. A keymap is a mapping from `<gdk-keymap-key>` to key values. You can think of a `<gdk-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 `<gdk-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.

In order to use a keymap to interpret a key event, it's necessary to first convert the keyboard state into an effective group and level. This is done via a set of rules that varies widely according to type of keyboard and user configuration. The function `gdk-keymap-translate-keyboard-state` accepts a keyboard state – consisting of hardware keycode pressed, active modifiers, and active group – applies the appropriate rules, and returns the group/level to be used to index the keymap, along with the modifiers which did not affect the group and level. i.e. it returns "unconsumed modifiers." The keyboard group may differ from the effective group used for keymap lookups because some keys don't have multiple groups - e.g. the Enter key is always in group 0 regardless of keyboard state.

Note that `gdk-keymap-translate-keyboard-state` also returns the keyval, i.e. it goes ahead and performs the keymap lookup in addition to telling you which effective group/level values were used for the lookup. `<gdk-event-key>` already contains this keyval, however, so you don't normally need to call `gdk-keymap-translate-keyboard-state` just to get the keyval.

20.2 Usage

`<gdk-keymap>` [Class]

Derives from `<gobject>`.

This class defines no direct slots.

`direction-changed` [Signal on `<gdk-keymap>`]

The ::direction_changed signal gets emitted when the direction of the keymap changes.

Since 2.0

`keys-changed` [Signal on `<gdk-keymap>`]

The ::keys_changed signal is emitted when the mapping represented by *keymap* changes.

Since 2.2

`gdk-keymap-get-default` ⇒ (*ret* `<gdk-keymap>`) [Function]

Returns the `<gdk-keymap>` attached to the default display.

ret the `<gdk-keymap>` attached to the default display.

`gdk-keymap-get-for-display` (*display* `<gdk-display>`) ⇒ (*ret* `<gdk-keymap>`) [Function]

Returns the `<gdk-keymap>` attached to *display*.

display the `<gdk-display>`.

ret the `<gdk-keymap>` attached to *display*.

Since 2.2

`gdk-keymap-get-direction` (*self* `<gdk-keymap>`) ⇒ (*ret* `<pango-direction>`) [Function]

`get-direction` [Method]

Returns the direction of the keymap.

keymap a `<gdk-keymap>` or '`#f`' to use the default keymap. Returns: '`PANGO_DIRECTION_LTR`' or '`PANGO_DIRECTION_RTL`'.

ret the direction of the keymap.

`gdk-keyval-name` (*keyval* `unsigned-int`) ⇒ (*ret* `mchars`) [Function]

Converts a key value into a symbolic name. The names are the same as those in the '`<gdk/gdkkeysyms.h>`' header file but without the leading "GDK_".

keyval a key value.

ret a string containing the name of the key, or '`#f`' if *keyval* is not a valid key. The string should not be modified.

`gdk-keyval-from-name` (*keyval_name* `mchars`) ⇒ (*ret* `unsigned-int`) [Function]
Converts a key name to a key value.

keyval-name
a key name.

ret the corresponding key value, or '`GDK_VoidSymbol`' if the key name is not a valid key.

`gdk-keyval-convert-case` (*symbol* `unsigned-int`) [Function]
⇒ (*lower* `unsigned-int`) (*upper* `unsigned-int`)
Obtains the upper- and lower-case versions of the keyval *symbol*. Examples of keyvals are `<gdk-a>`, `<gdk--enter>`, `<gdk-f1>`, etc.

symbol a keyval

lower return location for lowercase version of *symbol*

upper return location for uppercase version of *symbol*

`gdk-keyval-to-upper` (*keyval* `unsigned-int`) ⇒ (*ret* `unsigned-int`) [Function]
Converts a key value to upper case, if applicable.

keyval a key value.

ret the upper case form of *keyval*, or *keyval* itself if it is already in upper case or it is not subject to case conversion.

`gdk-keyval-to-lower` (*keyval* `unsigned-int`) ⇒ (*ret* `unsigned-int`) [Function]
Converts a key value to lower case, if applicable.

keyval a key value.

ret the lower case form of *keyval*, or *keyval* itself if it is already in lower case or it is not subject to case conversion.

`gdk-keyval-is-upper` (*keyval* `unsigned-int`) ⇒ (*ret* `bool`) [Function]
Returns '`#t`' if the given key value is in upper case.

keyval a key value.

ret '`#t`' if *keyval* is in upper case, or if *keyval* is not subject to case conversion.

`gdk-keyval-is-lower` (*keyval* `unsigned-int`) ⇒ (*ret* `bool`) [Function]
Returns '`#t`' if the given key value is in lower case.

keyval a key value.

ret '`#t`' if *keyval* is in lower case, or if *keyval* is not subject to case conversion.

`gdk-keyval-to-unicode` (*keyval* `unsigned-int`) [Function]
⇒ (*ret* `unsigned-int32`)
Convert from a GDK key symbol to the corresponding ISO10646 (Unicode) character.

keyval a GDK key symbol

ret the corresponding unicode character, or 0 if there is no corresponding character.

`gdk-unicode-to-keyval` (*wc* `unsigned-int32`) [Function]
⇒ (*ret* `unsigned-int`)

Convert from a ISO10646 character to a key symbol.

wc a ISO10646 encoded character

ret the corresponding GDK key symbol, if one exists. or, if there is no corresponding symbol, wc `|` 0x01000000

21 Selections

Functions for transfering data via the X selection mechanism

21.1 Overview

The X selection mechanism provides a way to transfer arbitrary chunks of data between programs. A *selection* is a essentially a named clipboard, identified by a string interned as a `<gdk-atom>`. By claiming ownership of a selection, an application indicates that it will be responsible for supplying its contents. The most common selections are '`PRIMARY`' and '`CLIPBOARD`'.

The contents of a selection can be represented in a number of formats, called *targets*. Each target is identified by an atom. A list of all possible targets supported by the selection owner can be retrieved by requesting the special target '`TARGETS`'. When a selection is retrieved, the data is accompanied by a type (an atom), and a format (an integer, representing the number of bits per item). See Properties and Atoms for more information.

The functions in this section only contain the lowlevel parts of the selection protocol. A considerably more complicated implementation is needed on top of this. GTK+ contains such an implementation in the functions in '`gtkselection.h`' and programmers should use those functions instead of the ones presented here. If you plan to implement selection handling directly on top of the functions here, you should refer to the X Inter-client Communication Conventions Manual (ICCCM).

21.2 Usage

`gdk-selection-owner-set` (*owner* `<gdk-window>`) [Function]
(*selection* `<gdk-atom>`) (*time_* `unsigned-int32`) (*send_event* `bool`)
⇒ (*ret* `bool`)

Sets the owner of the given selection.

owner a `<gdk-window>` or '`#f`' to indicate that the the owner for the given should be unset.

selection an atom identifying a selection.

time timestamp to use when setting the selection. If this is older than the timestamp given last time the owner was set for the given selection, the request will be ignored.

send-event if '`#t`', and the new owner is different from the current owner, the current owner will be sent a SelectionClear event.

ret '`#t`' if the selection owner was successfully changed to *owner*, otherwise '`#f`'.

`gdk-selection-owner-set-for-display` (*display* `<gdk-display>`) [Function]
(*owner* `<gdk-window>`) (*selection* `<gdk-atom>`) (*time_* `unsigned-int32`)
(*send_event* `bool`) ⇒ (*ret* `bool`)

Sets the `<gdk-window>`*owner* as the current owner of the selection *selection*.

display the `<gdk-display>`.

owner a `<gdk-window>` or '`#f`' to indicate that the owner for the given should be unset.

selection an atom identifying a selection.

time timestamp to use when setting the selection. If this is older than the timestamp given last time the owner was set for the given selection, the request will be ignored.

send-event if '`#t`', and the new owner is different from the current owner, the current owner will be sent a SelectionClear event.

ret '`#t`' if the selection owner was successfully changed to owner, otherwise '`#f`'.

Since 2.2

gdk-selection-owner-get (*selection* `<gdk-atom>`) [Function]
⇒ (*ret* `<gdk-window>`)

Determines the owner of the given selection.

selection an atom indentifying a selection.

ret if there is a selection owner for this window, and it is a window known to the current process, the `<gdk-window>` that owns the selection, otherwise '`#f`'. 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.

gdk-selection-owner-get-for-display (*display* `<gdk-display>`) [Function]
(*selection* `<gdk-atom>`) ⇒ (*ret* `<gdk-window>`)

Determine the owner of the given selection.

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.

display a `<gdk-display>`.

selection an atom indentifying a selection.

ret if there is a selection owner for this window, and it is a window known to the current process, the `<gdk-window>` that owns the selection, otherwise '`#f`'.

Since 2.2

gdk-selection-convert (*requestor* `<gdk-window>`) [Function]
(*selection* `<gdk-atom>`) (*target* `<gdk-atom>`) (*time_* `unsigned-int32`)

Retrieves the contents of a selection in a given form.

requestor a `<gdk-window>`.

selection an atom identifying the selection to get the contents of.

target the form in which to retrieve the selection.

time the timestamp to use when retrieving the selection. The selection owner may refuse the request if it did not own the selection at the time indicated by the timestamp.

`gdk-selection-send-notify` (*requestor* `unsigned-int32`) (*selection* `<gdk-atom>`) (*target* `<gdk-atom>`) (*property* `<gdk-atom>`) (*time_* `unsigned-int32`) [Function]

Sends a response to SelectionRequest event.

requestor window to which to deliver response.

selection selection that was requested.

target target that was selected.

property property in which the selection owner stored the data, or 'GDK_NONE' to indicate that the request was rejected.

time timestamp.

22 Drag and Drop

Functions for controlling drag and drop handling

22.1 Overview

These functions provide a low level interface for drag and drop. The X backend of GDK supports both the Xdnd and Motif drag and drop protocols transparently, the Win32 backend supports the WM_DROPFILES protocol.

GTK+ provides a higher level abstraction based on top of these functions, and so they are not normally needed in GTK+ applications. See the Drag and Drop section of the GTK+ documentation for more information.

22.2 Usage

`<gdk-drag-context>` [Class]

Derives from `<gobject>`.

This class defines no direct slots.

gdk-drag-get-selection (*context* `<gdk-drag-context>`) ⇒ (*ret* `<gdk-atom>`) [Function]

Returns the selection atom for the current source window.

context a `<gdk-drag-context>`.

ret the selection atom.

gdk-drag-abort (*context* `<gdk-drag-context>`) (*time_* `unsigned-int32`) [Function]

Aborts a drag without dropping.

This function is called by the drag source.

context a `<gdk-drag-context>`.

time the timestamp for this operation.

gdk-drop-reply (*context* `<gdk-drag-context>`) (*ok* `bool`) (*time_* `unsigned-int32`) [Function]

Accepts or rejects a drop.

This function is called by the drag destination in response to a drop initiated by the drag source.

context a `<gdk-drag-context>`.

ok '`#t`' if the drop is accepted.

time the timestamp for this operation.

gdk-drag-context-new ⇒ (*ret* `<gdk-drag-context>`) [Function]

Creates a new `<gdk-drag-context>`.

ret the newly created `<gdk-drag-context>`.

gdk-drag-drop (*context* <gdk-drag-context>) (*time_* unsigned-int32) [Function]

Drops on the current destination.

This function is called by the drag source.

context a <gdk-drag-context>.

time the timestamp for this operation.

gdk-drag-motion (*context* <gdk-drag-context>) (*dest_window* <gdk-window>) (*protocol* <gdk-drag-protocol>) (*x_root* int) (*y_root* int) (*suggested_action* <gdk-drag-action>) (*possible_actions* <gdk-drag-action>) (*time_* unsigned-int32) ⇒ (*ret* bool) [Function]

Updates the drag context when the pointer moves or the set of actions changes.

This function is called by the drag source.

context a <gdk-drag-context>.

dest-window
the new destination window, obtained by `gdk-drag-find-window`.

protocol the DND protocol in use, obtained by `gdk-drag-find-window`.

x-root the x position of the pointer in root coordinates.

y-root the y position of the pointer in root coordinates.

suggested-action
the suggested action.

possible-actions
the possible actions.

time the timestamp for this operation.

ret FIXME

gdk-drop-finish (*context* <gdk-drag-context>) (*success* bool) (*time_* unsigned-int32) [Function]

Ends the drag operation after a drop.

This function is called by the drag destination.

context a <gtk-drag-context>.

success '#t' if the data was successfully received.

time the timestamp for this operation.

gdk-drag-status (*context* <gdk-drag-context>) (*action* <gdk-drag-action>) (*time_* unsigned-int32) [Function]

Selects one of the actions offered by the drag source.

This function is called by the drag destination in response to `gdk-drag-motion` called by the drag source.

context a <gdk-drag-context>.

action the selected action which will be taken when a drop happens, or 0 to indicate that a drop will not be accepted.

time the timestamp for this operation.

`gdk-drag-drop-succeeded` (*context* `<gdk-drag-context>`) [Function]
⇒ (*ret* `bool`)

Returns wether the dropped data has been successfully transferred. This function is intended to be used while handling a '`GDK_DROP_FINISHED`' event, its return value is meaningless at other times.

context a `<gdk-drag-context>`

ret '`#t`' if the drop was successful.

Since 2.6

23 Properties and Atoms

Functions to manipulate properties on windows

23.1 Overview

Each window under X can have any number of associated *properties* attached to it. Properties are arbitrary chunks of data identified by *atom*s. (An *atom* is a numeric index into a string table on the X server. They are used to transfer strings efficiently between clients without having to transfer the entire string.) A property has an associated type, which is also identified using an atom.

A property has an associated *format*, an integer describing how many bits are in each unit of data inside the property. It must be 8, 16, or 32. When data is transferred between the server and client, if they are of different endianesses it will be byteswapped as necessary according to the format of the property. Note that on the client side, properties of format 32 will be stored with one unit per *long*, even if a long integer has more than 32 bits on the platform. (This decision was apparently made for Xlib to maintain compatibility with programs that assumed longs were 32 bits, at the expense of programs that knew better.)

The functions in this section are used to add, remove and change properties on windows, to convert atoms to and from strings and to manipulate some types of data commonly stored in X window properties.

23.2 Usage

`<gdk-atom>` [Class]

Opaque pointer.

This class defines no direct slots.

gdk-atom-intern (*atom_name* `mchars`) (*only_if_exists* `bool`) [Function]
⇒ (*ret* `<gdk-atom>`)

Finds or creates an atom corresponding to a given string.

atom-name
a string.

only-if-exists
if '`#t`', GDK is allowed to not create a new atom, but just return '`GDK_NONE`' if the requested atom doesn't already exists. Currently, the flag is ignored, since checking the existance of an atom is as expensive as creating it.

ret the atom corresponding to *atom-name*.

gdk-atom-name (*atom* `<gdk-atom>`) ⇒ (*ret* `mchars`) [Function]

Determines the string corresponding to an atom.

atom a `<gdk-atom>`.

ret a newly-allocated string containing the string corresponding to *atom*. When you are done with the return value, you should free it using `g-free`.

gdk-property-delete (*window* `<gdk-window>`) (*property* `<gdk-atom>`) [Function]

Deletes a property from a window.

window a `<gdk-window>`.

property the property to delete.

24 Threads

Functions for using GDK in multi-threaded programs

24.1 Overview

For thread safety, GDK relies on the thread primitives in GLib, and on the thread-safe GLib main loop.

GLib is completely thread safe (all global data is automatically locked), but individual data structure instances are not automatically locked for performance reasons. So e.g. you must coordinate accesses to the same `<g-hash-table>` from multiple threads.

GTK+ is "thread aware" but not thread safe — it provides a global lock controlled by `gdk-threads-enter`/`gdk-threads-leave` which protects all use of GTK+. That is, only one thread can use GTK+ at any given time.

Unfortunately the above holds with the X11 backend only. With the Win32 backend, GDK calls should not be attempted from multiple threads at all.

You must call `g-thread-init` and `gdk-threads-init` before executing any other GTK+ or GDK functions in a threaded GTK+ program.

Idles, timeouts, and input functions are executed outside of the main GTK+ lock. So, if you need to call GTK+ inside of such a callback, you must surround the callback with a `gdk-threads-enter`/`gdk-threads-leave` pair. (However, signals are still executed within the main GTK+ lock.)

In particular, this means, if you are writing widgets that might be used in threaded programs, you *must* surround timeouts and idle functions in this matter.

As always, you must also surround any calls to GTK+ not made within a signal handler with a `gdk-threads-enter`/`gdk-threads-leave` pair.

Before calling `gdk-threads-leave` from a thread other than your main thread, you probably want to call `gdk-flush` to send all pending commands to the windowing system. (The reason you don't need to do this from the main thread is that GDK always automatically flushes pending commands when it runs out of incoming events to process and has to sleep while waiting for more events.)

A minimal main program for a threaded GTK+ application looks like:

```
int
main (int argc, char *argv[])
{
  GtkWidget *window;

  g_thread_init (NULL);
  gdk_threads_init ();
  gdk_threads_enter ();

  gtk_init (&argc, &argv);

  window = create_window ();
```

```
  gtk_widget_show (window);

  gtk_main ();
  gdk_threads_leave ();

  return 0;
}
```

Callbacks require a bit of attention. Callbacks from GTK+ signals are made within the GTK+ lock. However callbacks from GLib (timeouts, IO callbacks, and idle functions) are made outside of the GTK+ lock. So, within a signal handler you do not need to call `gdk-threads-enter`, but within the other types of callbacks, you do.

Erik Mouw contributed the following code example to illustrate how to use threads within GTK+ programs.

```
/*-------------------------------------------------------------------------
 * Filename:      gtk-thread.c
 * Version:       0.99.1
 * Copyright:     Copyright (C) 1999, Erik Mouw
 * Author:        Erik Mouw <J.A.K.Mouw@its.tudelft.nl>
 * Description:   GTK threads example.
 * Created at:    Sun Oct 17 21:27:09 1999
 * Modified by:   Erik Mouw <J.A.K.Mouw@its.tudelft.nl>
 * Modified at:   Sun Oct 24 17:21:41 1999
 *-----------------------------------------------------------------------*/
/*
 * Compile with:
 *
 * cc -o gtk-thread gtk-thread.c ‘gtk-config --cflags --libs gthread‘
 *
 * Thanks to Sebastian Wilhelmi and Owen Taylor for pointing out some
 * bugs.
 *
 */

&#x0023;include <stdio.h>
&#x0023;include <stdlib.h>
&#x0023;include <unistd.h>
&#x0023;include <time.h>
&#x0023;include <gtk/gtk.h>
&#x0023;include <glib.h>
&#x0023;include <pthread.h>

&#x0023;define YES_IT_IS    (1)
&#x0023;define NO_IT_IS_NOT (0)

typedef struct
```

```
{
  GtkWidget *label;
  int what;
} yes_or_no_args;

G_LOCK_DEFINE_STATIC (yes_or_no);
static volatile int yes_or_no = YES_IT_IS;

void destroy (GtkWidget *widget, gpointer data)
{
  gtk_main_quit ();
}

void *argument_thread (void *args)
{
  yes_or_no_args *data = (yes_or_no_args *)args;
  gboolean say_something;

  for (;;)
    {
      /* sleep a while */
      sleep(rand() / (RAND_MAX / 3) + 1);

      /* lock the yes_or_no_variable */
      G_LOCK(yes_or_no);

      /* do we have to say something? */
      say_something = (yes_or_no != data->what);

      if(say_something)
{
  /* set the variable */
  yes_or_no = data->what;
}

      /* Unlock the yes_or_no variable */
      G_UNLOCK (yes_or_no);

      if (say_something)
{
  /* get GTK thread lock */
  gdk_threads_enter ();

  /* set label text */
  if(data->what == YES_IT_IS)
    gtk_label_set_text (GTK_LABEL (data->label), "O yes, it is!");
  else
```

```
    gtk_label_set_text (GTK_LABEL (data->label), "O no, it isn't!");

  /* release GTK thread lock */
  gdk_threads_leave ();
}
    }

  return NULL;
}

int main (int argc, char *argv[])
{
  GtkWidget *window;
  GtkWidget *label;
  yes_or_no_args yes_args, no_args;
  pthread_t no_tid, yes_tid;

  /* init threads */
  g_thread_init (NULL);
  gdk_threads_init ();
  gdk_threads_enter ();

  /* init gtk */
  gtk_init(&argc, &argv);

  /* init random number generator */
  srand ((unsigned int) time (NULL));

  /* create a window */
  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

  gtk_signal_connect (GTK_OBJECT (window), "destroy",
      GTK_SIGNAL_FUNC (destroy), NULL);

  gtk_container_set_border_width (GTK_CONTAINER (window), 10);

  /* create a label */
  label = gtk_label_new ("And now for something completely different ...");
  gtk_container_add (GTK_CONTAINER (window), label);

  /* show everything */
  gtk_widget_show (label);
  gtk_widget_show (window);

  /* create the threads */
  yes_args.label = label;
  yes_args.what = YES_IT_IS;
```

```
  pthread_create (&yes_tid, NULL, argument_thread, &yes_args);

  no_args.label = label;
  no_args.what = NO_IT_IS_NOT;
  pthread_create (&no_tid, NULL, argument_thread, &no_args);

  /* enter the GTK main loop */
  gtk_main ();
  gdk_threads_leave ();

  return 0;
}
```

24.2 Usage

`gdk-threads-init` [Function]

Initializes GDK so that it can be used from multiple threads in conjunction with `gdk-threads-enter` and `gdk-threads-leave`. `g-thread-init` must be called previous to this function.

This call must be made before any use of the main loop from GTK+; to be safe, call it before `gtk-init`.

`gdk-threads-enter` [Function]

This macro marks the beginning of a critical section in which GDK and GTK+ functions can be called. Only one thread at a time can be in such a critial section.

`gdk-threads-leave` [Function]

Leaves a critical region begun with `gdk-threads-enter`.

25 Input Devices

Functions for handling extended input devices

25.1 Overview

In addition to the normal keyboard and mouse input devices, GTK+ 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 `<gdk-window>` individually using `gdk-input-set-extension-events`. (Or, more typically, for GtkWidgets, using `gtk-widget-set-extension-events`). 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 '`GDK_EXTENSION_EVENTS_CURSOR`' value to `gdk-input-set-extension-events`. Otherwise, the application must retrieve the device information using `gdk-devices-list`, check the field, and, if it is '`#f`', 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 field in the event structure. The events generated by pointer devices have also been extended to contain , and fields which contain the extended information reported as additional *valuators* from the device. The field is a a double value ranging from 0.0 to 1.0, while the tilt fields are double values ranging 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 field in each event is the field, which contains an enumeration value describing the type of device; this currently can be one of '`GDK_SOURCE_MOUSE`', '`GDK_SOURCE_PEN`', '`GDK_SOURCE_ERASER`', or '`GDK_SOURCE_CURSOR`'. This field 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-input-dialog>` widget in GTK+. 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 `gdk-devices-list`. Each device must be activated using `gdk-device-set-mode`, 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 `gdk-device-set-axis-use`. And the source type for each device can be set with `gdk-device-set-source`.

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 `gdk-device-set-key`.

The interfaces in this section will most likely be considerably modified in the future to accomodate devices that may have different sets of additional valuators than the pressure and .

25.2 Usage

`<gdk-device>` [Class]

Derives from `<gobject>`.

This class defines no direct slots.

`gdk-devices-list` ⇒ (*ret* `glist-of`) [Function]

Returns the list of available input devices for the default display. The list is statically allocated and should not be freed.

ret a list of `<gdk-device>`

`gdk-device-set-source` (*self* `<gdk-device>`) (*source* `<gdk-input-source>`) [Function]

`set-source` [Method]

Sets the source type for an input device.

device a `<gdk-device>`.

source the source type.

`gdk-device-set-mode` (*self* `<gdk-device>`) (*mode* `<gdk-input-mode>`) ⇒ (*ret* `bool`) [Function]

`set-mode` [Method]

Sets a the mode of an input device. The mode controls if the device is active and whether the device's range is mapped to the entire screen or to a single window.

device a `<gdk-device>`.

mode the input mode.

ret '`#t`' if the mode was successfully changed.

`gdk-device-set-key` (*self* `<gdk-device>`) (*index_* `unsigned-int`) (*keyval* `unsigned-int`) (*modifiers* `<gdk-modifier-type>`) [Function]

`set-key` [Method]

Specifies the X key event to generate when a macro button of a device is pressed.

device a `<gdk-device>`.

index the index of the macro button to set.

keyval the keyval to generate.

modifiers the modifiers to set.

`gdk-device-set-axis-use` (*self* `<gdk-device>`) (*index_* `unsigned-int`) (*use* `<gdk-axis-use>`) [Function]

`set-axis-use` [Method]

Specifies how an axis of a device is used.

device a `<gdk-device>`.

index the index of the axis.

use specifies how the axis is used.

`gdk-device-get-core-pointer` ⇒ (*ret* `<gdk-device>`) [Function]

Returns the core pointer device for the default display.

ret the core pointer device; this is owned by the display and should not be freed.

`gdk-device-get-axis` (*self* `<gdk-device>`) (*use* `<gdk-axis-use>`) [Function]
⇒ (*ret* `bool`) (*axes* `double`) (*value* `double`)

`get-axis` [Method]

Interprets an array of double as axis values for a given device, and locates the value in the array for a given axis use.

device a `<gdk-device>`

axes pointer to an array of axes

use the use to look for

value location to store the found value.

ret '`#t`' if the given axis use was found, otherwise '`#f`'

`gdk-input-set-extension-events` (*window* `<gdk-window>`) [Function]
(*mask* `int`) (*mode* `<gdk-extension-mode>`)

Turns extension events on or off for a particular window, and specifies the event mask for extension events.

window a `<gdk-window>`.

mask the event mask

mode the type of extension events that are desired.

26 Pango Interaction

Using Pango in GDK

26.1 Overview

Pango is the text layout system used by GDK and GTK+. The functions and types in this section are used to render Pango objects to GDK. drawables, and also extend the set of Pango attributes to include stippling and embossing.

Creating a <pango-layout> object is the first step in rendering text, and requires getting a handle to a <pango-context>. For GTK+ programs, you'll usually want to use gtk-widget-get-pango-context, or gtk-widget-create-pango-layout, rather than using the lowlevel gdk-pango-context-get-for-screen. Once you have a <pango-layout>, you can set the text and attributes of it with Pango functions like pango-layout-set-text and get its size with pango-layout-get-size. (Note that Pango uses a fixed point system internally, so converting between Pango units and pixels using PANGO_SCALE or the pango-pixels macro.)

Rendering a Pango layout is done most simply with gdk-draw-layout; you can also draw pieces of the layout with gdk-draw-layout or gdk-draw-glyphs. <gdk-pango-renderer> is a subclass of <pango-renderer> that is used internally to implement these functions. Using it directly or subclassing it can be useful in some cases. See the <gdk-pango-renderer> documentation for details.

```
#define RADIUS 100
#define N_WORDS 10
#define FONT "Sans Bold 18"

GdkScreen *screen = gdk_drawable_get_screen (drawable);
PangoRenderer *renderer;
GdkGC *gc;

PangoMatrix matrix = PANGO_MATRIX_INIT;
PangoContext *context;
PangoLayout *layout;
PangoFontDescription *desc;

double device_radius;
int width, height;
int i;

/* Get the default renderer for the screen, and set it up for drawing  */
renderer = gdk_pango_renderer_get_default (screen);
gdk_pango_renderer_set_drawable (GDK_PANGO_RENDERER (renderer), drawable);

gc = gdk_gc_new (drawable);
gdk_pango_renderer_set_gc (GDK_PANGO_RENDERER (renderer), gc);
```

```
/* Set up a transformation matrix so that the user space coordinates for
 * where we are drawing are [-RADIUS, RADIUS], [-RADIUS, RADIUS]
 * We first center, then change the scale */
gdk_drawable_get_size (drawable, &width, &height);
device_radius = MIN (width, height) / 2.;

pango_matrix_translate (&matrix,
                        device_radius + (width - 2 * device_radius) / 2,
                        device_radius + (height - 2 * device_radius) / 2);
pango_matrix_scale (&matrix, device_radius / RADIUS, device_radius / RADIUS);

/* Create a PangoLayout, set the font and text */
context = gdk_pango_context_get_for_screen (screen);
layout = pango_layout_new (context);
pango_layout_set_text (layout, "Text", -1);
desc = pango_font_description_from_string (FONT);
pango_layout_set_font_description (layout, desc);
pango_font_description_free (desc);

/* Draw the layout N_WORDS times in a circle */
for (i = 0; i < N_WORDS; i++)
  {
    GdkColor color;
    PangoMatrix rotated_matrix = matrix;
    int width, height;
    double angle = (360. * i) / N_WORDS;

    /* Gradient from red at angle == 60 to blue at angle == 300 */
    color.red   = 65535 * (1 + cos ((angle - 60) * M_PI / 180.)) / 2;
    color.green = 0;
    color.blue  = 65535  - color.red;

    gdk_pango_renderer_set_override_color (GDK_PANGO_RENDERER (renderer),
                                           PANGO_RENDER_PART_FOREGROUND, &color);

    pango_matrix_rotate (&rotated_matrix, angle);

    pango_context_set_matrix (context, &rotated_matrix);

    /* Inform Pango to re-layout the text with the new transformation matrix */
    pango_layout_context_changed (layout);

    pango_layout_get_size (layout, &width, &height);
    pango_renderer_draw_layout (renderer, layout,
                                - width / 2, - RADIUS * PANGO_SCALE);
  }
```

```
/* Clean up default renderer, since it is shared */
gdk_pango_renderer_set_override_color (GDK_PANGO_RENDERER (renderer),
                                       PANGO_RENDER_PART_FOREGROUND, NULL);
gdk_pango_renderer_set_drawable (GDK_PANGO_RENDERER (renderer), NULL);
gdk_pango_renderer_set_gc (GDK_PANGO_RENDERER (renderer), NULL);

/* free the objects we created */
g_object_unref (layout);
g_object_unref (context);
g_object_unref (gc);
```

26.2 Usage

`<gdk-pango-renderer>` [Class]

Derives from `<pango-renderer>`.

This class defines the following slots:

`screen` the GdkScreen for the renderer

`gdk-pango-renderer-new` (*screen* `<gdk-screen>`) [Function]
⇒ (*ret* `<pango-renderer>`)

Creates a new `<pango-renderer>` for *screen*. Normally you can use the results of `gdk-pango-renderer-get-default` rather than creating a new renderer.

screen a `<gdk-screen>`

ret a newly created `<pango-renderer>`. Free with `g-object-unref`.

Since 2.6

`gdk-pango-renderer-get-default` (*screen* `<gdk-screen>`) [Function]
⇒ (*ret* `<pango-renderer>`)

Gets the default `<pango-renderer>` for a screen. This default renderer is shared by all users of the display, so properties such as the color or transformation matrix set for the renderer may be overwritten by functions such as `gdk-draw-layout`.

Before using the renderer, you need to call `gdk-pango-renderer-set-drawable` and `gdk-pango-renderer-set-gc` to set the drawable and graphics context to use for drawing.

screen a `<gdk-screen>`

ret the default `<pango-renderer>` for *screen*. The renderer is owned by GTK+ and will be kept around until the screen is closed.

Since 2.6

`gdk-pango-renderer-set-drawable` (*self* `<gdk-pango-renderer>`) [Function]
(*drawable* `<gdk-drawable>`)

`set-drawable` [Method]

Sets the drawable the renderer draws to.

gdk-renderer
a `<gdk-pango-renderer>`

drawable the new target drawable, or '`#f`'

Since 2.6

`gdk-pango-renderer-set-gc` (*self* `<gdk-pango-renderer>`) (*gc* `<gdk-gc>`) [Function]

`set-gc` [Method]

Sets the GC the renderer draws with. Note that the GC must not be modified until it is unset by calling the function again with '`#f`' for the *gc* parameter, since GDK may make internal copies of the GC which won't be updated to follow changes to the original GC.

gdk-renderer
a `<gdk-pango-renderer>`

gc the new GC to use for drawing, or '`#f`'

Since 2.6

`gdk-pango-renderer-set-stipple` (*self* `<gdk-pango-renderer>`) (*part* `<pango-render-part>`) (*stipple* `<gdk-drawable>`) [Function]

`set-stipple` [Method]

Sets the stipple for one render part (foreground, background, underline, etc.) Note that this is overwritten when iterating through the individual styled runs of a `<pango-layout>` or `<pango-layout-line>`. This function is thus only useful when you call low level functions like `pango-renderer-draw-glyphs` directly, or in the 'prepare_run' virtual function of a subclass of `<gdk-pango-renderer>`.

gdk-renderer
a `<gdk-pango-renderer>`

part the part to render with the stipple

stipple the new stipple value.

Since 2.6

`gdk-pango-context-get` ⇒ (*ret* `<pango-context>`) [Function]

Creates a `<pango-context>` for the default GDK screen.

The context must be freed when you're finished with it.

When using GTK+, normally you should use `gtk-widget-get-pango-context` instead of this function, to get the appropriate context for the widget you intend to render text onto.

The newly created context will have the default font options (see `<cairo-font-options-t>`) for the default screen; if these options change it will not be updated. Using `gtk-widget-get-pango-context` is more convenient if you want to keep a context around and track changes to the screen's font rendering settings.

ret a new `<pango-context>` for the default display

`gdk-pango-context-get-for-screen` (*screen* `<gdk-screen>`) [Function]
⇒ (*ret* `<pango-context>`)

Creates a `<pango-context>` for *screen*.

The context must be freed when you're finished with it.

When using GTK+, normally you should use `gtk-widget-get-pango-context` instead of this function, to get the appropriate context for the widget you intend to render text onto.

The newly created context will have the default font options (see `<cairo-font-options-t>`) for the screen; if these options change it will not be updated. Using `gtk-widget-get-pango-context` is more convenient if you want to keep a context around and track changes to the screen's font rendering settings.

screen the `<gdk-screen>` for which the context is to be created.

ret a new `<pango-context>` for *screen*

Since 2.2

`gdk-pango-context-set-colormap` (*context* `<pango-context>`) (*colormap* `<gdk-colormap>`) [Function]

'`gdk_pango_context_set_colormap`' is deprecated and should not be used in newly-written code.

This function used to set the colormap to be used for drawing with *context*. The colormap is now always derived from the graphics context used for drawing, so calling this function is no longer necessary.

context a `<pango-context>`

colormap a `<gdk-colormap>`

`gdk-pango-attr-embossed-new` (*embossed* `bool`) [Function]
⇒ (*ret* `<pango-attribute>`)

Creates a new attribute containing a embossed bitmap to be used when rendering the text.

embossed a bitmap to be set as embossed

ret new `<pango-attribute>`

`gdk-pango-attr-stipple-new` (*stipple* `<gdk-drawable>`) [Function]
⇒ (*ret* `<pango-attribute>`)

Creates a new attribute containing a stipple bitmap to be used when rendering the text.

stipple a bitmap to be set as stipple

ret new `<pango-attribute>`

27 Cairo Interaction

Functions to support using Cairo

27.1 Overview

Cairo is a graphics library that supports vector graphics and image compositing that can be used with GDK. Since 2.8, GTK+ does most of its drawing using Cairo.

GDK does not wrap the Cairo API, instead it allows to create Cairo contexts which can be used to draw on GDK drawables. Additional functions allow to convert GDK's rectangles and regions into Cairo paths and to use pixbufs as sources for drawing operations.

27.2 Usage

`gdk-cairo-create` (*drawable* `<gdk-drawable>`) ⇒ (*ret* `cairo-t`) [Function]

Creates a Cairo context for drawing to *drawable*.

drawable a `<gdk-drawable>`

ret A newly created Cairo context. Free with `cairo-destroy` when you are done drawing.

Since 2.8

`gdk-cairo-set-source-color` (*cr* `cairo-t`) (*color* `<gdk-color>`) [Function]

Sets the specified `<gdk-color>` as the source color of *cr*.

cr a `<cairo-t>`

color a `<gdk-color>`

Since 2.8

`gdk-cairo-set-source-pixbuf` (*cr* `cairo-t`) (*pixbuf* `<gdk-pixbuf>`) (*pixbuf_x* `double`) (*pixbuf_y* `double`) [Function]

Sets the given pixbuf as the source pattern for the Cairo context. The pattern has an extend mode of '`CAIRO_EXTEND_NONE`' and is aligned so that the origin of *pixbuf* is *pixbuf-x*, *pixbuf-y*

cr a `<cairo>` context

pixbuf a `<gdk-pixbuf>`

pixbuf-x X coordinate of location to place upper left corner of *pixbuf*

pixbuf-y Y coordinate of location to place upper left corner of *pixbuf*

Since 2.8

`gdk-cairo-set-source-pixmap` (*cr* `cairo-t`) (*pixmap* `<gdk-pixmap>`) (*pixmap_x* `double`) (*pixmap_y* `double`) [Function]

Sets the given pixmap as the source pattern for the Cairo context. The pattern has an extend mode of '`CAIRO_EXTEND_NONE`' and is aligned so that the origin of *pixbuf* is *pixbuf-x*, *pixbuf-y* *cr* a `<cairo>` context

pixmap a `<gdk-pixmap>`

pixmap-x X coordinate of location to place upper left corner of *pixmap*

pixmap-y Y coordinate of location to place upper left corner of *pixmap*

Since 2.10

`gdk-cairo-rectangle` (*cr* `cairo-t`) (*rectangle* `<gdk-rectangle>`) [Function]

Adds the given rectangle to the current path of *cr*.

cr a `<cairo-t>`

rectangle a `<gdk-rectangle>`

Since 2.8

`gdk-cairo-region` (*cr* `cairo-t`) (*region* `<gdk-region>`) [Function]

Adds the given region to the current path of *cr*.

cr a `<cairo-t>`

region a `<gdk-region>`

Since 2.8

28 X Window System Interaction

X backend-specific functions

28.1 Overview

28.2 Usage

`gdk-pixmap-foreign-new-for-display` (*display* `<gdk-display>`) (*anid* `unsigned-long`) ⇒ (*ret* `<gdk-pixmap>`) [Function]

Wraps a native pixmap in a `<gdk-pixmap>`. This may fail if the pixmap has been destroyed.

For example in the X backend, a native pixmap handle is an Xlib `<xid>`.

display The `<gdk-display>` where *anid* is located.

anid a native pixmap handle.

ret the newly-created `<gdk-pixmap>` wrapper for the native pixmap or '`#f`' if the pixmap has been destroyed.

Since 2.2

`gdk-pixmap-foreign-new-for-screen` (*screen* `<gdk-screen>`) (*anid* `unsigned-long`) (*width* `int`) (*height* `int`) (*depth* `int`) ⇒ (*ret* `<gdk-pixmap>`) [Function]

Wraps a native pixmap in a `<gdk-pixmap>`. This may fail if the pixmap has been destroyed.

For example in the X backend, a native pixmap handle is an Xlib `<xid>`.

This function is an alternative to `gdk-pixmap-foreign-new-for-display` for cases where the dimensions of the pixmap are known. For the X backend, this avoids a roundtrip to the server.

screen a `<gdk-screen>`

anid a native pixmap handle

width the width of the pixmap identified by *anid*

height the height of the pixmap identified by *anid*

depth the depth of the pixmap identified by *anid*

ret the newly-created `<gdk-pixmap>` wrapper for the native pixmap or '`#f`' if the pixmap has been destroyed.

Since 2.10

`gdk-window-foreign-new-for-display` (*display* `<gdk-display>`) (*anid* `unsigned-long`) ⇒ (*ret* `<gdk-window>`) [Function]

Wraps a native window in a `<gdk-window>`. This may fail if the window has been destroyed. If the window was already known to GDK, a new reference to the existing `<gdk-window>` is returned.

For example in the X backend, a native window handle is an Xlib `<xid>`.

display the `<gdk-display>` where the window handle comes from.

anid a native window handle.

ret a `<gdk-window>` wrapper for the native window or '`#f`' if the window has been destroyed. The wrapper will be newly created, if one doesn't exist already.

Since 2.2

`gdk-window-lookup` (*anid* `unsigned-long`) ⇒ (*ret* `<gdk-window>`) [Function]

Looks up the `<gdk-window>` that wraps the given native window handle.

For example in the X backend, a native window handle is an Xlib `<xid>`.

anid a native window handle.

ret the `<gdk-window>` wrapper for the native window, or '`#f`' if there is none.

`gdk-pixmap-lookup` (*anid* `unsigned-long`) ⇒ (*ret* `<gdk-pixmap>`) [Function]

Looks up the `<gdk-pixmap>` that wraps the given native pixmap handle.

For example in the X backend, a native pixmap handle is an Xlib `<xid>`.

anid a native pixmap handle.

ret the `<gdk-window>` wrapper for the native window, or '`#f`' if there is none.

29 The GdkPixbuf Structure

Information that describes an image.

29.1 Overview

The structure contains information that describes an image in memory.

Image data in a pixbuf is stored in memory in uncompressed, packed format. Rows in the image are stored top to bottom, and in each row pixels are stored from left to right. There may be padding at the end of a row. The "rowstride" value of a pixbuf, as returned by `gdk-pixbuf-get-rowstride`, indicates the number of bytes between rows.

The following code illustrates a simple put_pixel() function for RGB pixbufs with 8 bits per channel with an alpha channel. It is not included in the gdk-pixbuf library for performance reasons; rather than making several function calls for each pixel, your own code can take shortcuts.

```
static void
put_pixel (GdkPixbuf *pixbuf, int x, int y, guchar red, guchar green, guchar blue, guchar alpha)
{
  int width, height, rowstride, n_channels;
  guchar *pixels, *p;

  n_channels = gdk_pixbuf_get_n_channels (pixbuf);

  g_assert (gdk_pixbuf_get_colorspace (pixbuf) == GDK_COLORSPACE_RGB);
  g_assert (gdk_pixbuf_get_bits_per_sample (pixbuf) == 8);
  g_assert (gdk_pixbuf_get_has_alpha (pixbuf));
  g_assert (n_channels == 4);

  width = gdk_pixbuf_get_width (pixbuf);
  height = gdk_pixbuf_get_height (pixbuf);

  g_assert (x >= 0 && x < width);
  g_assert (y >= 0 && y < height);

  rowstride = gdk_pixbuf_get_rowstride (pixbuf);
  pixels = gdk_pixbuf_get_pixels (pixbuf);

  p = pixels + y * rowstride + x * n_channels;
  p[0] = red;
  p[1] = green;
  p[2] = blue;
  p[3] = alpha;
}
```

This function will not work for pixbufs with images that are other than 8 bits per sample or channel, but it will work for most of the pixbufs that GTK+ uses.

If you are doing `memcpy` of raw pixbuf data, note that the last row in the pixbuf may not be as wide as the full rowstride, but rather just as wide as the pixel data needs to be. That is, it is unsafe to do '`memcpy (dest, pixels, rowstride * height)`' to copy a whole pixbuf. Use `gdk-pixbuf-copy` instead, or compute the width in bytes of the last row as '`width * ((n_channels * bits_per_sample + 7) / 8)`'.

29.2 Usage

`<gdk-pixbuf>` [Class]

Derives from `<gobject>`.

This class defines the following slots:

`colorspace`
: The colorspace in which the samples are interpreted

`n-channels`
: The number of samples per pixel

`has-alpha`
: Whether the pixbuf has an alpha channel

`bits-per-sample`
: The number of bits per sample

`width`
: The number of columns of the pixbuf

`height`
: The number of rows of the pixbuf

`rowstride`
: The number of bytes between the start of a row and the start of the next row

`pixels`
: A pointer to the pixel data of the pixbuf

`gdk-pixbuf-get-colorspace` (*self* `<gdk-pixbuf>`) ⇒ (*ret* `<gdk-colorspace>`) [Function]

`get-colorspace` [Method]

Queries the color space of a pixbuf.

pixbuf A pixbuf.

ret Color space.

`gdk-pixbuf-get-n-channels` (*self* `<gdk-pixbuf>`) ⇒ (*ret* `int`) [Function]

`get-n-channels` [Method]

Queries the number of channels of a pixbuf.

pixbuf A pixbuf.

ret Number of channels.

`gdk-pixbuf-get-has-alpha` (*self* `<gdk-pixbuf>`) ⇒ (*ret* `bool`) [Function]

`get-has-alpha` [Method]

Queries whether a pixbuf has an alpha channel (opacity information).

pixbuf A pixbuf.

ret '#t' if it has an alpha channel, '#f' otherwise.

gdk-pixbuf-get-bits-per-sample (*self* <gdk-pixbuf>) ⇒ (*ret* int) [Function]
get-bits-per-sample [Method]

Queries the number of bits per color sample in a pixbuf.

pixbuf A pixbuf.

ret Number of bits per color sample.

gdk-pixbuf-get-width (*self* <gdk-pixbuf>) ⇒ (*ret* int) [Function]
get-width [Method]

Queries the width of a pixbuf.

pixbuf A pixbuf.

ret Width in pixels.

gdk-pixbuf-get-height (*self* <gdk-pixbuf>) ⇒ (*ret* int) [Function]
get-height [Method]

Queries the height of a pixbuf.

pixbuf A pixbuf.

ret Height in pixels.

gdk-pixbuf-get-rowstride (*self* <gdk-pixbuf>) ⇒ (*ret* int) [Function]
get-rowstride [Method]

Queries the rowstride of a pixbuf, which is the number of bytes between the start of a row and the start of the next row.

pixbuf A pixbuf.

ret Distance between row starts.

gdk-pixbuf-get-option (*self* <gdk-pixbuf>) (*key* mchars) ⇒ (*ret* mchars) [Function]
get-option [Method]

Looks up *key* in the list of options that may have been attached to the *pixbuf* when it was loaded.

pixbuf a <gdk-pixbuf>

key a nul-terminated string.

ret the value associated with *key*. This is a nul-terminated string that should not be freed or '#f' if *key* was not found.

30 File Loading

Loading a pixbuf from a file.

30.1 Overview

The gdk-pixbuf library provides a simple mechanism for loading an image from a file in synchronous fashion. This means that the library takes control of the application while the file is being loaded; from the user's point of view, the application will block until the image is done loading.

This interface can be used by applications in which blocking is acceptable while an image is being loaded. It can also be used to load small images in general. Applications that need progressive loading can use the `<gdk-pixbuf-loader>` functionality instead.

30.2 Usage

`gdk-pixbuf-new-from-file` (*filename* `mchars`) ⇒ (*ret* `<gdk-pixbuf>`) [Function]

Creates a new pixbuf by loading an image from a file. The file format is detected automatically. If '`#f`' is returned, then *error* will be set. Possible errors are in the `<gdk-pixbuf-error>` and `<g-file-error>` domains.

filename Name of file to load, in the GLib file name encoding

error Return location for an error

ret A newly-created pixbuf with a reference count of 1, or '`#f`' 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.

`gdk-pixbuf-new-from-file-at-size` (*filename* `mchars`) (*width* `int`) (*height* `int`) ⇒ (*ret* `<gdk-pixbuf>`) [Function]

Creates a new pixbuf by loading an image from a file. The file format is detected automatically. If '`#f`' is returned, then *error* will be set. Possible errors are in the `<gdk-pixbuf-error>` and `<g-file-error>` domains. The image will be scaled to fit in the requested size, preserving the image's aspect ratio.

filename Name of file to load, in the GLib file name encoding

width The width the image should have or -1 to not constrain the width

height The height the image should have or -1 to not constrain the height

error Return location for an error

ret A newly-created pixbuf with a reference count of 1, or '`#f`' 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.

Since 2.4

`gdk-pixbuf-new-from-file-at-scale` (*filename* `mchars`) (*width* `int`) (*height* `int`) (*preserve_aspect_ratio* `bool`) ⇒ (*ret* `<gdk-pixbuf>`) [Function]

Creates a new pixbuf by loading an image from a file. The file format is detected automatically. If '`#f`' is returned, then *error* will be set. Possible errors are in the `<gdk-pixbuf-error>` and `<g-file-error>` domains. The image will be scaled to fit in the requested size, optionally preserving the image's aspect ratio.

When preserving the aspect ratio, a *width* of -1 will cause the image to be scaled to the exact given height, and a *height* of -1 will cause the image to be scaled to the exact given width. When not preserving aspect ratio, a *width* or *height* of -1 means to not scale the image at all in that dimension. Negative values for *width* and *height* are allowed since 2.8.

filename Name of file to load, in the GLib file name encoding

width The width the image should have or -1 to not constrain the width

height The height the image should have or -1 to not constrain the height

preserve-aspect-ratio
'`#t`' to preserve the image's aspect ratio

error Return location for an error

ret A newly-created pixbuf with a reference count of 1, or '`#f`' 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.

Since 2.6

31 Image Data in Memory

Creating a pixbuf from image data that is already in memory.

31.1 Overview

The most basic way to create a pixbuf is to wrap an existing pixel buffer with a `<gdk-pixbuf>` structure. You can use the `gdk-pixbuf-new-from-data` function to do this You need to specify the destroy notification function that will be called when the data buffer needs to be freed; this will happen when a `<gdk-pixbuf>` is finalized by the reference counting functions If you have a chunk of static data compiled into your application, you can pass in '`#f`' as the destroy notification function so that the data will not be freed.

The `gdk-pixbuf-new` function can be used as a convenience to create a pixbuf with an empty buffer. This is equivalent to allocating a data buffer using `malloc` and then wrapping it with `gdk-pixbuf-new-from-data`. The `gdk-pixbuf-new` function will compute an optimal rowstride so that rendering can be performed with an efficient algorithm.

As a special case, you can use the `gdk-pixbuf-new-from-xpm-data` function to create a pixbuf from inline XPM image data.

You can also copy an existing pixbuf with the `gdk-pixbuf-copy` function. This is not the same as just doing a `g-object-ref` on the old pixbuf; the copy function will actually duplicate the pixel data in memory and create a new `<gdk-pixbuf>` structure for it.

31.2 Usage

gdk-pixbuf-new (*colorspace* `<gdk-colorspace>`) (*has_alpha* `bool`) (*bits_per_sample* `int`) (*width* `int`) (*height* `int`) ⇒ (*ret* `<gdk-pixbuf>`) [Function]

Creates a new `<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.

colorspace Color space for image

has-alpha Whether the image should have transparency information

bits-per-sample
Number of bits per color sample

width Width of image in pixels, must be > 0

height Height of image in pixels, must be > 0

ret A newly-created `<gdk-pixbuf>` with a reference count of 1, or '`#f`' if not enough memory could be allocated for the image buffer.

gdk-pixbuf-new-subpixbuf (*self* `<gdk-pixbuf>`) (*src_x* `int`) (*src_y* `int`) (*width* `int`) (*height* `int`) ⇒ (*ret* `<gdk-pixbuf>`) [Function]

new-subpixbuf [Method]

Creates a new pixbuf which represents a sub-region of *src-pixbuf*. The new pixbuf shares its pixels with the original pixbuf, so writing to one affects both. The new pixbuf holds a reference to *src-pixbuf*, so *src-pixbuf* will not be finalized until the new pixbuf is finalized.

src-pixbuf a `<gdk-pixbuf>`

src-x X coord in *src-pixbuf*

src-y Y coord in *src-pixbuf*

width width of region in *src-pixbuf*

height height of region in *src-pixbuf*

ret a new pixbuf

32 Scaling

Scaling pixbufs and scaling and compositing pixbufs

32.1 Overview

The gdk-pixbuf contains functions to scale pixbufs, to scale pixbufs and composite against an existing image, and to scale pixbufs and composite against a solid color or checkerboard. Compositing a checkerboard is a common way to show an image with an alpha channel in image-viewing and editing software.

Since the full-featured functions (`gdk-pixbuf-scale`, `gdk-pixbuf-composite`, and `gdk-pixbuf-composite-color`) are rather complex to use and have many arguments, two simple convenience functions are provided, `gdk-pixbuf-scale-simple` and `gdk-pixbuf-composite-color-simple` which create a new pixbuf of a given size, scale an original image to fit, and then return the new pixbuf.

The following example demonstrates handling an expose event by rendering the appropriate area of a source image (which is scaled to fit the widget) onto the widget's window. The source image is rendered against a checkerboard, which provides a visual representation of the alpha channel if the image has one. If the image doesn't have an alpha channel, calling `gdk-pixbuf-composite-color` function has exactly the same effect as calling `gdk-pixbuf-scale`.

```
gboolean
expose_cb (GtkWidget *widget, GdkEventExpose *event, gpointer data)
{
  GdkPixbuf *dest;

  dest = gdk_pixbuf_new (GDK_COLORSPACE_RGB, FALSE, 8, event->area.width, event->area.height);

  gdk_pixbuf_composite_color (pixbuf, dest,
                              0, 0, event->area.width, event->area.height,
                              -event->area.x, -event->area.y,
                              (double) widget->allocation.width / gdk_pixbuf_get_width (pixbuf),
                              (double) widget->allocation.height / gdk_pixbuf_get_height (pixbuf),
                              GDK_INTERP_BILINEAR, 255,
                              event->area.x, event->area.y, 16, 0xaaaaaa, 0x555555);

  gdk_pixbuf_render_to_drawable (dest, widget->window, widget->style->fg_gc[GTK_STATE_NORMAL],
                                 0, 0, event->area.x, event->area.y,
                                 event->area.width, event->area.height,
                                 GDK_RGB_DITHER_NORMAL, event->area.x, event->area.y);

  gdk_pixbuf_unref (dest);

  return TRUE;
}
```

32.2 Usage

gdk-pixbuf-scale-simple (*self* <gdk-pixbuf>) (*dest_width* int) (*dest_height* int) (*interp_type* <gdk-interp-type>) ⇒ (*ret* <gdk-pixbuf>) [Function]

scale-simple [Method]

Create a new <gdk-pixbuf> containing a copy of *src* scaled to *dest-width* x *dest-height*. Leaves *src* unaffected. *interp-type* should be <gdk-interp-nearest> if you want maximum speed (but when scaling down <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 gdk-pixbuf-new-subpixbuf.

For more complicated scaling/compositing see gdk-pixbuf-scale and gdk-pixbuf-composite.

src a <gdk-pixbuf>

dest-width the width of destination image

dest-height
the height of destination image

interp-type
the interpolation type for the transformation.

ret the new <gdk-pixbuf>, or '#f' if not enough memory could be allocated for it.

gdk-pixbuf-scale (*self* <gdk-pixbuf>) (*dest* <gdk-pixbuf>) (*dest_x* int) (*dest_y* int) (*dest_width* int) (*dest_height* int) (*offset_x* double) (*offset_y* double) (*scale_x* double) (*scale_y* double) (*interp_type* <gdk-interp-type>) [Function]

scale [Method]

Creates a transformation of the source image *src* by scaling by *scale-x* and *scale-y* then translating by *offset-x* and *offset-y*, then renders the rectangle (*dest-x*, *dest-y*, *dest-width*, *dest-height*) of the resulting image onto the destination image replacing the previous contents.

Try to use gdk-pixbuf-scale-simple first, this function is the industrial-strength power tool you can fall back to if gdk-pixbuf-scale-simple isn't powerful enough.

src a <gdk-pixbuf>

dest the <gdk-pixbuf> into which to render the results

dest-x the left coordinate for region to render

dest-y the top coordinate for region to render

dest-width the width of the region to render

dest-height
the height of the region to render

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.

`gdk-pixbuf-composite-color-simple` (*self* `<gdk-pixbuf>`) [Function]
(*dest_width* `int`) (*dest_height* `int`) (*interp_type* `<gdk-interp-type>`)
(*overall_alpha* `int`) (*check_size* `int`) (*color1* `unsigned-int32`)
(*color2* `unsigned-int32`) ⇒ (*ret* `<gdk-pixbuf>`)

`composite-color-simple` [Method]

Creates a new `<gdk-pixbuf>` by scaling *src* to *dest-width* x *dest-height* and compositing the result with a checkboard of colors *color1* and *color2*.

src a `<gdk-pixbuf>`

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

ret the new `<gdk-pixbuf>`, or '`#f`' if not enough memory could be allocated for it.

`gdk-pixbuf-composite` (*self* `<gdk-pixbuf>`) (*dest* `<gdk-pixbuf>`) [Function]
(*dest_x* `int`) (*dest_y* `int`) (*dest_width* `int`) (*dest_height* `int`)
(*offset_x* `double`) (*offset_y* `double`) (*scale_x* `double`) (*scale_y* `double`)
(*interp_type* `<gdk-interp-type>`) (*overall_alpha* `int`)

`composite` [Method]

Creates a transformation of the source image *src* by scaling by *scale-x* and *scale-y* then translating by *offset-x* and *offset-y*. This gives an image in the coordinates of the destination pixbuf. The rectangle (*dest-x*, *dest-y*, *dest-width*, *dest-height*) is then composited onto the corresponding rectangle of the original destination image.

When the destination rectangle contains parts not in the source image, the data at the edges of the source image is replicated to infinity.

(The missing figure, pixbuf-composite-diagram

src a `<gdk-pixbuf>`

dest the <gdk-pixbuf> into which to render the results

dest-x the left coordinate for region to render

dest-y the top coordinate for region to render

dest-width the width of the region to render

dest-height
the height of the region to render

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)

`gdk-pixbuf-composite-color` (*self* `<gdk-pixbuf>`) [Function]
(*dest* `<gdk-pixbuf>`) (*dest_x* `int`) (*dest_y* `int`) (*dest_width* `int`)
(*dest_height* `int`) (*offset_x* `double`) (*offset_y* `double`) (*scale_x* `double`)
(*scale_y* `double`) (*interp_type* `<gdk-interp-type>`) (*overall_alpha* `int`)
(*check_x* `int`) (*check_y* `int`) (*check_size* `int`) (*color1* `unsigned-int32`)
(*color2* `unsigned-int32`)

`composite-color` [Method]

Creates a transformation of the source image *src* by scaling by *scale-x* and *scale-y* then translating by *offset-x* and *offset-y*, then composites 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.

See `gdk-pixbuf-composite-color-simple` for a simpler variant of this function suitable for many tasks.

src a <gdk-pixbuf>

dest the <gdk-pixbuf> into which to render the results

dest-x the left coordinate for region to render

dest-y the top coordinate for region to render

dest-width the width of the region to render

dest-height
the height of the region to render

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

`gdk-pixbuf-rotate-simple` (*self* `<gdk-pixbuf>`) (*angle* `<gdk-pixbuf-rotation>`) ⇒ (*ret* `<gdk-pixbuf>`) [Function]

`rotate-simple` [Method]

Rotates a pixbuf by a multiple of 90 degrees, and returns the result in a new pixbuf.

src a `<gdk-pixbuf>`

angle the angle to rotate by

ret a new pixbuf

Since 2.6

`gdk-pixbuf-flip` (*self* `<gdk-pixbuf>`) (*horizontal* `bool`) ⇒ (*ret* `<gdk-pixbuf>`) [Function]

`flip` [Method]

Flips a pixbuf horizontally or vertically and returns the result in a new pixbuf.

src a `<gdk-pixbuf>`

horizontal '`#t`' to flip horizontally, '`#f`' to flip vertically

ret a new pixbuf.

Since 2.6

33 Utilities

Utility and miscellaneous convenience functions.

33.1 Overview

These functions provide miscellaneous utilities for manipulating pixbufs. The pixel data in pixbufs may of course be manipulated directly by applications, but several common operations can be performed by these functions instead.

33.2 Usage

`gdk-pixbuf-add-alpha` (*self* `<gdk-pixbuf>`) (*substitute_color* `bool`) (*r* `unsigned-char`) (*g* `unsigned-char`) (*b* `unsigned-char`) ⇒ (*ret* `<gdk-pixbuf>`) [Function]

`add-alpha` [Method]

Takes an existing pixbuf and adds an alpha channel to it. If the existing pixbuf already had an alpha channel, the channel values are copied from the original; otherwise, the alpha channel is initialized to 255 (full opacity).

If *substitute-color* is '`#t`', 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.

pixbuf A `<gdk-pixbuf>`.

substitute-color
Whether to set a color to zero opacity. If this is '`#f`', then the (*r*, *g*, *b*) arguments will be ignored.

r Red value to substitute.

g Green value to substitute.

b Blue value to substitute.

ret A newly-created pixbuf with a reference count of 1.

`gdk-pixbuf-copy-area` (*self* `<gdk-pixbuf>`) (*src_x* `int`) (*src_y* `int`) (*width* `int`) (*height* `int`) (*dest_pixbuf* `<gdk-pixbuf>`) (*dest_x* `int`) (*dest_y* `int`) [Function]

`copy-area` [Method]

Copies a rectangular area from *src-pixbuf* to *dest-pixbuf*. Conversion of pixbuf formats is done automatically.

src-pixbuf Source pixbuf.

src-x Source X coordinate within *src-pixbuf*.

src-y Source Y coordinate within *src-pixbuf*.

width Width of the area to copy.

height Height of the area to copy.

dest-pixbuf
Destination pixbuf.

dest-x X coordinate within *dest-pixbuf*.

dest-y Y coordinate within *dest-pixbuf*.

`gdk-pixbuf-saturate-and-pixelate` (*self* `<gdk-pixbuf>`) (*dest* `<gdk-pixbuf>`) (*saturation* `float`) (*pixelate* `bool`) [Function]
`saturate-and-pixelate` [Method]

Modifies saturation and optionally pixelates *src*, placing the result in *dest*. *src* and *dest* may be the same 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 turns toward grayscale); if greater than 1.0, saturation is increased (the image gets more vivid colors). If *pixelate* is '`#t`', then pixels are faded in a checkerboard pattern to create a pixelated image. *src* and *dest* must have the same image format, size, and rowstride.

src source image

dest place to write modified version of *src*

saturation saturation factor

pixelate whether to pixelate

`gdk-pixbuf-fill` (*self* `<gdk-pixbuf>`) (*pixel* `unsigned-int32`) [Function]
`fill` [Method]

Clears a pixbuf to the given RGBA value, converting the RGBA value into the pixbuf's pixel format. The alpha will be ignored if the pixbuf doesn't have an alpha channel.

pixbuf a `<gdk-pixbuf>`

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

34 Animations

Animated images.

34.1 Overview

The gdk-pixbuf library provides a simple mechanism to load and represent animations. 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, however; you just ask gdk-pixbuf what should be displayed at a given point in time.

34.2 Usage

`<gdk-pixbuf-animation>` [Class]

Derives from `<gobject>`.

This class defines no direct slots.

`<gdk-pixbuf-animation-iter>` [Class]

Derives from `<gobject>`.

This class defines no direct slots.

`<gdk-pixbuf-simple-anim>` [Class]

Derives from `<gdk-pixbuf-animation>`.

This class defines no direct slots.

`gdk-pixbuf-animation-new-from-file` (*filename* `mchars`) [Function]
⇒ (*ret* `<gdk-pixbuf-animation>`)

Creates a new animation 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. Possible errors are in the `<gdk-pixbuf-error>` and `<g-file-error>` domains.

filename Name of file to load, in the GLib file name encoding

error return location for error

ret A newly-created animation with a reference count of 1, or '`#f`' if any of several error conditions ocurred: 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.

`gdk-pixbuf-animation-get-width` (*self* `<gdk-pixbuf-animation>`) [Function]
⇒ (*ret* `int`)

`get-width` [Method]

Queries the width of the bounding box of a pixbuf animation.

animation An animation.

ret Width of the bounding box of the animation.

`gdk-pixbuf-animation-get-height` (*self* `<gdk-pixbuf-animation>`) [Function]
⇒ (*ret* `int`)

`get-height` [Method]

Queries the height of the bounding box of a pixbuf animation.

animation An animation.

ret Height of the bounding box of the animation.

`gdk-pixbuf-simple-anim-new` (*width* `int`) (*height* `int`) (*rate* `float`) [Function]
⇒ (*ret* `<gdk-pixbuf-simple-anim>`)

Creates a new, empty animation.

width the width of the animation

height the height of the animation

rate the speed of the animation, in frames per second

ret a newly allocated `<gdk-pixbuf-simple-anim>`

Since 2.8

`gdk-pixbuf-simple-anim-add-frame` [Function]
(*self* `<gdk-pixbuf-simple-anim>`) (*pixbuf* `<gdk-pixbuf>`)

`add-frame` [Method]

Adds a new frame to *animation*. The *pixbuf* must have the dimensions specified when the animation was constructed.

animation a `<gdk-pixbuf-simple-anim>`

pixbuf the pixbuf to add

Since 2.8

35 GdkPixbufLoader

Application-driven progressive image loading.

35.1 Overview

`<gdk-pixbuf-loader>` 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 functionality instead of `gdk-pixbuf-new-from-file` or `gdk-pixbuf-animation-new-from-file` 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 `<gdk-pixbuf-loader>` to load an image, just create a new one, and call `gdk-pixbuf-loader-write` to send the data to it. When done, `gdk-pixbuf-loader-close` should be called to end the stream and finalize everything. The loader will emit three important signals throughout the process. The first, "size_prepared", will be called as soon as the image has enough information to determine the size of the image to be used. If you want to scale the image while loading it, you can call `gdk-pixbuf-loader-set-size` in response to this signal.

The second signal, "area_prepared", will be called as soon as the pixbuf of the desired has been allocated. You can obtain it by calling `gdk-pixbuf-loader-get-pixbuf`. If you want to use it, simply ref it. In addition, no actual information will be passed in yet, so the pixbuf can be safely filled with any temporary graphics (or an initial color) as needed. You can also call `gdk-pixbuf-loader-get-pixbuf` later and get the same pixbuf.

The last signal, "area_updated" gets called 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.

35.2 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 `gdk-pixbuf-loader-get-animation` to get the `<gdk-pixbuf-animation>` struct and `gdk-pixbuf-animation-get-iter` to get an `<gdk-pixbuf-animation-iter>` for displaying it.

35.3 Usage

`<gdk-pixbuf-loader>` [Class]

Derives from `<gobject>`.

This class defines no direct slots.

`closed` [Signal on `<gdk-pixbuf-loader>`]

This signal is emitted when `gdk-pixbuf-loader-close` is called. It can be used by different parts of an application to receive notification when an image loader is closed by the code that drives it.

size-prepared (*arg0* `<gint>`) (*arg1* `<gint>`) [Signal on `<gdk-pixbuf-loader>`]

This 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 `gdk-pixbuf-loader-set-size` in response to this signal to set the desired size to which the image should be scaled.

area-prepared [Signal on `<gdk-pixbuf-loader>`]

This signal is emitted when the pixbuf loader has allocated the pixbuf in the desired size. After this signal is emitted, applications can call `gdk-pixbuf-loader-get-pixbuf` to fetch the partially-loaded pixbuf.

area-updated (*arg0* `<gint>`) (*arg1* `<gint>`) (*arg2* `<gint>`) (*arg3* `<gint>`) [Signal on `<gdk-pixbuf-loader>`]

This signal is emitted when a significant area of the image being loaded has been updated. Normally it means that a complete scanline has been read in, but it could be a different area as well. Applications can use this signal to know when to repaint areas of an image that is being loaded.

gdk-pixbuf-loader-new ⇒ (*ret* `<gdk-pixbuf-loader>`) [Function]

Creates a new pixbuf loader object.

ret A newly-created pixbuf loader.

gdk-pixbuf-loader-new-with-type (*image_type* `mchars`) ⇒ (*ret* `<gdk-pixbuf-loader>`) [Function]

Creates a new pixbuf loader object that always attempts to parse image data as if it were an image of type *image-type*, instead of identifying the type automatically. 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.

The list of supported image formats depends on what image loaders are installed, but typically "png", "jpeg", "gif", "tiff" and "xpm" are among the supported formats. To obtain the full list of supported image formats, call `gdk-pixbuf-format-get-name` on each of the `<gdk-pixbuf-format>` structs returned by `gdk-pixbuf-get-formats`.

image-type
name of the image format to be loaded with the image

error return location for an allocated `<g-error>`, or '`#f`' to ignore errors

ret A newly-created pixbuf loader.

gdk-pixbuf-loader-get-format (*self* `<gdk-pixbuf-loader>`) ⇒ (*ret* `<gdk-pixbuf-format*>`) [Function]

get-format [Method]

Obtains the available information about the format of the currently loading image file.

loader A pixbuf loader.

ret A `<gdk-pixbuf-format>` or '`#f`'. The return value is owned by GdkPixbuf and should not be freed.

Since 2.2

`gdk-pixbuf-loader-set-size` (*self* `<gdk-pixbuf-loader>`) (*width* `int`) (*height* `int`) [Function]

`set-size` [Method]

Causes the image to be scaled while it is loaded. The desired image size can be determined relative to the original size of the image by calling `gdk-pixbuf-loader-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 signal.

loader A pixbuf loader.

width The desired width of the image being loaded.

height The desired height of the image being loaded.

Since 2.2

`gdk-pixbuf-loader-get-pixbuf` (*self* `<gdk-pixbuf-loader>`) ⇒ (*ret* `<gdk-pixbuf>`) [Function]

`get-pixbuf` [Method]

Queries the `<gdk-pixbuf>` that a pixbuf loader is currently creating. In general it only makes sense to call this function after the "area_prepared" signal has been emitted by the loader; this 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 `gdk-pixbuf-loader-write`, then this function returns '`#f`'. The returned pixbuf will be the same in all future calls to the loader, so simply calling `g-object-ref` should be sufficient to continue using it. Additionally, if the loader is an animation, it will return the "static image" of the animation (see `gdk-pixbuf-animation-get-static-image`).

loader A pixbuf loader.

ret The `<gdk-pixbuf>` that the loader is creating, or '`#f`' if not enough data has been read to determine how to create the image buffer.

`gdk-pixbuf-loader-get-animation` (*self* `<gdk-pixbuf-loader>`) ⇒ (*ret* `<gdk-pixbuf-animation>`) [Function]

`get-animation` [Method]

Queries the `<gdk-pixbuf-animation>` that a pixbuf loader is currently creating. In general it only makes sense to call this function 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 function will return '`#f`'.

loader A pixbuf loader

ret The `<gdk-pixbuf-animation>` that the loader is loading, or '`#f`' if not enough data has been read to determine the information.

`gdk-pixbuf-loader-close` (*self* `<gdk-pixbuf-loader>`) ⇒ (*ret* `bool`) [Function]

`close` [Method]

Informs a pixbuf loader that no further writes with `gdk-pixbuf-loader-write` will occur, so that it can free its internal loading structures. Also, tries to parse any data that hasn't yet been parsed; if the remaining data is partial or corrupt, an error will be returned. If '#f' is returned, *error* will be set to an error from the <gdk-pixbuf-error> or <g-file-error> domains. If you're just cancelling a load rather than expecting it to be finished, passing '#f' for *error* to ignore it is reasonable.

loader A pixbuf loader.

error return location for a <g-error>, or '#f' to ignore errors

ret '#t' if all image data written so far was successfully passed out via the update_area signal

36 Module Interface

Extending gdk-pixbuf

36.1 Overview

If gdk-pixbuf has been compiled with GModule support, it can be extended by modules which can load (and perhaps also save) new image and animation formats. Each loadable module must export a `<gdk-pixbuf-module-fill-info-func>` function named `fill-info` and a `<gdk-pixbuf-module-fill-vtable-func>` function named `fill-vtable`.

In order to make format-checking work before actually loading the modules (which may require dlopening image libraries), modules export their signatures (and other information) via the `fill-info` function. An external utility, , uses this to create a text file containing a list of all available loaders and their signatures. This file is then read at runtime by gdk-pixbuf to obtain the list of available loaders and their signatures.

Modules may only implement a subset of the functionality available via `<gdk-pixbuf-module>`. If a particular functionality is not implemented, the `fill-vtable` function will simply not set the corresponding function pointers of the `<gdk-pixbuf-module>` structure. If a module supports incremental loading (i.e. provides `<begin-load>`, `<stop-load>` and `<load-increment>`), it doesn't have to implement `<load>`, since gdk-pixbuf can supply a generic `<load>` implementation wrapping the incremental loading.

Installing a module is a two-step process:

copy the module file(s) to the loader directory (normally '`/gtk-2.0//loaders`', unless overridden by the environment variable `GDK_PIXBUF_MODULEDIR`)

call to update the module file (normally '`/gtk-2.0/gdk-pixbuf.loaders`', unless overridden by the environment variable `GDK_PIXBUF_MODULE_FILE`)

The gdk-pixbuf interfaces needed for implementing modules are contained in '`gdk-pixbuf-io.h`' (and '`gdk-pixbuf-animation.h`' if the module supports animations). They are not covered by the same stability guarantees as the regular gdk-pixbuf API. To underline this fact, they are protected by '`#ifdef GDK_PIXBUF_ENABLE_BACKEND`'.

36.2 Usage

`gdk-pixbuf-get-formats` ⇒ (*ret* `gslist-of`) [Function]

Obtains the available information about the image formats supported by GdkPixbuf.

ret A list of `<gdk-pixbuf-format>`s describing the supported image formats. The list should be freed when it is no longer needed, but the structures themselves are owned by `<gdk-pixbuf>` and should not be freed.

Since 2.2

37 Undocumented

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

37.1 (gnome gw gdk)

`<gdk-pixbuf-format*>` [Variable]

`gdk-display-manager-get-default-display` [Variable]

`gdk-display-manager-set-default-display` [Variable]

`gdk-display-request-selection-notification` [Variable]

`gdk-display-set-double-click-distance` [Variable]

`gdk-display-supports-clipboard-persistence` [Variable]

`gdk-display-supports-selection-notification` [Variable]

`gdk-event->vector` [Variable]

`gdk-pango-renderer-set-override-color` [Variable]

`gdk-pixbuf-animation-get-static-image` [Variable]

`gdk-pixbuf-animation-is-static-image` [Variable]

`gdk-pixbuf-animation-iter-get-delay-time` [Variable]

`gdk-pixbuf-animation-iter-get-pixbuf` [Variable]

`gdk-pixbuf-animation-iter-on-currently-loading-frame` [Variable]

`gdk-pixbuf-error-quark` [Variable]

`gdk-pixbuf-loader-new-with-mime-type` [Function]

`gdk-pixbuf-save-to-port` [Function]

`gdk-selection-send-notify-for-display` [Variable]

`gdk-window-enable-synchronized-configure` [Variable]

`gdk-window-input-shape-combine-region` [Variable]

Type Index

<gdk-atom> 108
<gdk-color> 54
<gdk-colormap> 54
<gdk-cursor> 65
<gdk-device> 116
<gdk-display-manager> 14
<gdk-display> 7
<gdk-drag-context> 105
<gdk-drawable> 32
<gdk-event-any> 96
<gdk-event-button> 96
<gdk-event-client> 97
<gdk-event-configure> 97
<gdk-event-crossing> 96
<gdk-event-dnd> 97
<gdk-event-expose> 96
<gdk-event-focus> 96
<gdk-event-key> 96
<gdk-event-motion> 96
<gdk-event-no-expose> 97
<gdk-event-property> 97
<gdk-event-proximity> 97
<gdk-event-scroll> 96
<gdk-event-selection> 97
<gdk-event-setting> 97
<gdk-event-visibility> 96
<gdk-event-window-state> 97
<gdk-font> 60
<gdk-gc> 27
<gdk-image> 48
<gdk-keymap> 99
<gdk-pango-renderer> 120
<gdk-pixbuf-animation-iter> 141
<gdk-pixbuf-animation> 141
<gdk-pixbuf-loader> 143
<gdk-pixbuf-simple-anim> 141
<gdk-pixbuf> 128
<gdk-rectangle> 24
<gdk-region> 24
<gdk-screen> 15
<gdk-visual> 56

Function Index

A

add-alpha 139
add-frame 142
area-prepared on <gdk-pixbuf-loader> 144
area-updated on <gdk-pixbuf-loader> 144

B

beep 9
begin-move-drag 77
begin-paint-rect 77
begin-paint-region 77
begin-resize-drag 76
broadcast-client-message 20

C

clear 75
clear-area 75
clear-area-e 75
close 10, 145
closed on <gdk-display> 7
closed on <gdk-pixbuf-loader> 143
composite 136
composite-color 137
composite-color-simple 136
composited-changed on <gdk-screen> 15
configure-finished 80
copy 31
copy-area 139
copy-to-image 40

D

deiconify 71
destroy 69
direction-changed on <gdk-keymap> 99
display-opened on <gdk-display-manager> 14

E

end-paint 78

F

fill 140
flip 138
flush 9
focus 76
freeze-updates 79
fullscreen 72

G

gdk-atom-intern 108
gdk-atom-name 108
gdk-beep 5
gdk-bitmap-create-from-data 42
gdk-cairo-create 123
gdk-cairo-rectangle 124
gdk-cairo-region 124
gdk-cairo-set-source-color 123
gdk-cairo-set-source-pixbuf 123
gdk-cairo-set-source-pixmap 123
gdk-char-height 64
gdk-char-measure 63
gdk-char-width 62
gdk-char-width-wc 62
gdk-color-alloc 55
gdk-color-black 55
gdk-color-change 55
gdk-color-copy 54
gdk-color-parse 55
gdk-color-white 54
gdk-cursor-get-display 68
gdk-cursor-get-image 68
gdk-cursor-new 65
gdk-cursor-new-for-display 67
gdk-cursor-new-from-name 67
gdk-cursor-new-from-pixbuf 66
gdk-cursor-new-from-pixmap 65
gdk-device-get-axis 117
gdk-device-get-core-pointer 117
gdk-device-set-axis-use 116
gdk-device-set-key 116
gdk-device-set-mode 116
gdk-device-set-source 116
gdk-devices-list 116
gdk-display-beep 9
gdk-display-close 10
gdk-display-flush 9
gdk-display-get-core-pointer 14
gdk-display-get-default 7
gdk-display-get-default-cursor-size 12
gdk-display-get-default-group 12
gdk-display-get-default-screen 8
gdk-display-get-event 10
gdk-display-get-maximal-cursor-size 12
gdk-display-get-n-screens 8
gdk-display-get-name 7
gdk-display-get-screen 8
gdk-display-get-window-at-pointer 11
gdk-display-keyboard-ungrab 8
gdk-display-list-devices 10
gdk-display-manager-get 14
gdk-display-manager-list-displays 14
gdk-display-open 7

gdk-display-peek-event 10
gdk-display-pointer-is-grabbed 9
gdk-display-pointer-ungrab 8
gdk-display-put-event 10
gdk-display-set-double-click-time 11
gdk-display-supports-cursor-alpha 12
gdk-display-supports-cursor-color 11
gdk-display-supports-input-shapes 13
gdk-display-supports-shapes 12
gdk-display-sync 9
gdk-display-warp-pointer 11
gdk-drag-abort 105
gdk-drag-context-new 105
gdk-drag-drop 106
gdk-drag-drop-succeeded 107
gdk-drag-get-selection 105
gdk-drag-motion 106
gdk-drag-status 106
gdk-draw-arc 35
gdk-draw-drawable 39
gdk-draw-glyphs 36
gdk-draw-glyphs-transformed 36
gdk-draw-image 39
gdk-draw-layout 37
gdk-draw-layout-line 37
gdk-draw-layout-line-with-colors 37
gdk-draw-layout-with-colors 38
gdk-draw-line 34
gdk-draw-pixbuf 34
gdk-draw-point 34
gdk-draw-rectangle 35
gdk-draw-string 38
gdk-drawable-copy-to-image 40
gdk-drawable-get-clip-region 33
gdk-drawable-get-colormap 33
gdk-drawable-get-depth 33
gdk-drawable-get-display 32
gdk-drawable-get-image 40
gdk-drawable-get-screen 32
gdk-drawable-get-size 33
gdk-drawable-get-visible-region 34
gdk-drawable-get-visual 32
gdk-drawable-set-colormap 33
gdk-drop-finish 106
gdk-drop-reply 105
gdk-error-trap-pop 6
gdk-error-trap-push 6
gdk-event-copy 94
gdk-event-get 93
gdk-event-get-axis 94
gdk-event-get-coords 94
gdk-event-get-graphics-expose 93
gdk-event-get-root-coords 94
gdk-event-get-screen 95
gdk-event-get-time 94
gdk-event-peek 93
gdk-event-put 93
gdk-events-pending 93
gdk-flush 3
gdk-font-from-description 61
gdk-font-id 61
gdk-font-load 60
gdk-fontset-load 60
gdk-gc-copy 31
gdk-gc-get-colormap 31
gdk-gc-get-screen 27
gdk-gc-new 27
gdk-gc-offset 31
gdk-gc-set-background 28
gdk-gc-set-clip-mask 29
gdk-gc-set-clip-origin 29
gdk-gc-set-clip-rectangle 30
gdk-gc-set-clip-region 30
gdk-gc-set-colormap 31
gdk-gc-set-exposures 30
gdk-gc-set-fill 29
gdk-gc-set-font 28
gdk-gc-set-foreground 27
gdk-gc-set-function 28
gdk-gc-set-line-attributes 30
gdk-gc-set-rgb-bg-color 28
gdk-gc-set-rgb-fg-color 28
gdk-gc-set-stipple 29
gdk-gc-set-subwindow 30
gdk-gc-set-tile 29
gdk-gc-set-ts-origin 29
gdk-get-default-root-window 92
gdk-get-display 3
gdk-get-display-arg-name 2
gdk-get-program-class 3
gdk-get-show-events 94
gdk-get-use-xshm 5
gdk-image-get 48
gdk-image-get-colormap 49
gdk-image-get-pixel 49
gdk-image-new 48
gdk-image-put-pixel 49
gdk-image-set-colormap 49
gdk-input-set-extension-events 117
gdk-keyboard-grab 5
gdk-keyboard-ungrab 5
gdk-keymap-get-default 99
gdk-keymap-get-direction 99
gdk-keymap-get-for-display 99
gdk-keyval-convert-case 100
gdk-keyval-from-name 100
gdk-keyval-is-lower 100
gdk-keyval-is-upper 100
gdk-keyval-name 99
gdk-keyval-to-lower 100
gdk-keyval-to-unicode 100
gdk-keyval-to-upper 100
gdk-list-visuals 56
gdk-notify-startup-complete 2
gdk-pango-attr-embossed-new 122
gdk-pango-attr-stipple-new 122

gdk-pango-context-get 121
gdk-pango-context-get-for-screen 122
gdk-pango-context-set-colormap 122
gdk-pango-renderer-get-default 120
gdk-pango-renderer-new 120
gdk-pango-renderer-set-drawable 120
gdk-pango-renderer-set-gc 121
gdk-pango-renderer-set-stipple 121
gdk-pixbuf-add-alpha 139
gdk-pixbuf-animation-get-height 142
gdk-pixbuf-animation-get-width 141
gdk-pixbuf-animation-new-from-file 141
gdk-pixbuf-composite 136
gdk-pixbuf-composite-color 137
gdk-pixbuf-composite-color-simple 136
gdk-pixbuf-copy-area 139
gdk-pixbuf-fill 140
gdk-pixbuf-flip 138
gdk-pixbuf-get-bits-per-sample 129
gdk-pixbuf-get-colorspace 128
gdk-pixbuf-get-formats 147
gdk-pixbuf-get-from-drawable 52
gdk-pixbuf-get-from-image 53
gdk-pixbuf-get-has-alpha 128
gdk-pixbuf-get-height 129
gdk-pixbuf-get-n-channels 128
gdk-pixbuf-get-option 129
gdk-pixbuf-get-rowstride 129
gdk-pixbuf-get-width 129
gdk-pixbuf-loader-close 145
gdk-pixbuf-loader-get-animation 145
gdk-pixbuf-loader-get-format 144
gdk-pixbuf-loader-get-pixbuf 145
gdk-pixbuf-loader-new 144
gdk-pixbuf-loader-new-with-mime-type 148
gdk-pixbuf-loader-new-with-type 144
gdk-pixbuf-loader-set-size 145
gdk-pixbuf-new 132
gdk-pixbuf-new-from-file 130
gdk-pixbuf-new-from-file-at-scale 131
gdk-pixbuf-new-from-file-at-size 130
gdk-pixbuf-new-subpixbuf 132
gdk-pixbuf-render-threshold-alpha 50
gdk-pixbuf-render-to-drawable 50
gdk-pixbuf-render-to-drawable-alpha 51
gdk-pixbuf-rotate-simple 138
gdk-pixbuf-saturate-and-pixelate 140
gdk-pixbuf-save-to-port 148
gdk-pixbuf-scale 135
gdk-pixbuf-scale-simple 135
gdk-pixbuf-simple-anim-add-frame 142
gdk-pixbuf-simple-anim-new 142
gdk-pixmap-create-from-data 42
gdk-pixmap-foreign-new-for-display 125
gdk-pixmap-foreign-new-for-screen 125
gdk-pixmap-lookup 126
gdk-pixmap-new 42
gdk-pointer-grab 4
gdk-pointer-is-grabbed 5
gdk-pointer-ungrab 4
gdk-property-delete 109
gdk-region-copy 24
gdk-region-destroy 24
gdk-region-empty 25
gdk-region-equal 25
gdk-region-intersect 26
gdk-region-new 24
gdk-region-offset 25
gdk-region-point-in 25
gdk-region-rect-in 25
gdk-region-rectangle 24
gdk-region-shrink 25
gdk-region-subtract 26
gdk-region-union 26
gdk-region-union-with-rect 26
gdk-region-xor 26
gdk-rgb-colormap-ditherable 47
gdk-rgb-ditherable 47
gdk-rgb-gc-set-background 46
gdk-rgb-gc-set-foreground 45
gdk-rgb-get-colormap 46
gdk-rgb-get-visual 46
gdk-rgb-init 45
gdk-rgb-set-install 46
gdk-rgb-set-min-colors 46
gdk-rgb-set-verbose 47
gdk-rgb-xpixel-from-rgb 46
gdk-screen-broadcast-client-message 20
gdk-screen-get-active-window 22
gdk-screen-get-default 15
gdk-screen-get-default-colormap 15
gdk-screen-get-display 18
gdk-screen-get-font-options 21
gdk-screen-get-height 18
gdk-screen-get-height-mm 19
gdk-screen-get-monitor-at-point 20
gdk-screen-get-monitor-at-window 20
gdk-screen-get-monitor-geometry 20
gdk-screen-get-n-monitors 19
gdk-screen-get-number 18
gdk-screen-get-resolution 22
gdk-screen-get-rgb-colormap 16
gdk-screen-get-rgb-visual 16
gdk-screen-get-rgba-colormap 17
gdk-screen-get-rgba-visual 17
gdk-screen-get-root-window 17
gdk-screen-get-setting 21
gdk-screen-get-system-colormap 16
gdk-screen-get-system-visual 16
gdk-screen-get-toplevel-windows 19
gdk-screen-get-width 18
gdk-screen-get-width-mm 18
gdk-screen-get-window-stack 22
gdk-screen-height 3
gdk-screen-height-mm 3
gdk-screen-is-composited 17

gdk-screen-list-visuals 19
gdk-screen-make-display-name 19
gdk-screen-set-default-colormap 16
gdk-screen-set-font-options 21
gdk-screen-set-resolution 22
gdk-screen-width 3
gdk-screen-width-mm 3
gdk-selection-convert 103
gdk-selection-owner-get 103
gdk-selection-owner-get-for-display 103
gdk-selection-owner-set 102
gdk-selection-owner-set-for-display 102
gdk-selection-send-notify 104
gdk-set-double-click-time 5
gdk-set-locale 2
gdk-set-program-class 3
gdk-set-show-events 95
gdk-set-sm-client-id 2
gdk-set-use-xshm 6
gdk-spawn-command-line-on-screen 23
gdk-string-extents 61
gdk-string-height 63
gdk-string-measure 62
gdk-string-width 61
gdk-text-height 63
gdk-text-measure 63
gdk-text-width 62
gdk-threads-enter 114
gdk-threads-init 114
gdk-threads-leave 114
gdk-unicode-to-keyval 101
gdk-visual-get-best 57
gdk-visual-get-best-depth 56
gdk-visual-get-best-type 57
gdk-visual-get-best-with-both 57
gdk-visual-get-best-with-depth 57
gdk-visual-get-best-with-type 57
gdk-visual-get-screen 57
gdk-visual-get-system 57
gdk-window-at-pointer 69
gdk-window-begin-move-drag 77
gdk-window-begin-paint-rect 77
gdk-window-begin-paint-region 77
gdk-window-begin-resize-drag 76
gdk-window-clear 75
gdk-window-clear-area 75
gdk-window-clear-area-e 75
gdk-window-configure-finished 80
gdk-window-deiconify 71
gdk-window-destroy 69
gdk-window-end-paint 78
gdk-window-focus 76
gdk-window-foreign-new-for-display 125
gdk-window-freeze-updates 79
gdk-window-fullscreen 72
gdk-window-get-children 89
gdk-window-get-deskrelative-origin 88
gdk-window-get-events 89
gdk-window-get-geometry 85
gdk-window-get-group 91
gdk-window-get-origin 88
gdk-window-get-parent 89
gdk-window-get-position 88
gdk-window-get-root-origin 88
gdk-window-get-state 70
gdk-window-get-toplevel 89
gdk-window-get-toplevels 92
gdk-window-get-type-hint 87
gdk-window-get-update-area 79
gdk-window-get-window-type 69
gdk-window-hide 70
gdk-window-iconify 71
gdk-window-input-shape-combine-mask 83
gdk-window-invalidate-rect 78
gdk-window-invalidate-region 78
gdk-window-is-viewable 70
gdk-window-is-visible 70
gdk-window-lookup 126
gdk-window-lower 76
gdk-window-maximize 71
gdk-window-merge-child-input-shapes 83
gdk-window-merge-child-shapes 82
gdk-window-move 73
gdk-window-move-region 74
gdk-window-move-resize 74
gdk-window-process-all-updates 79
gdk-window-process-updates 79
gdk-window-raise 76
gdk-window-register-dnd 76
gdk-window-reparent 75
gdk-window-resize 73
gdk-window-scroll 74
gdk-window-set-accept-focus 81
gdk-window-set-back-pixmap 85
gdk-window-set-background 84
gdk-window-set-child-input-shapes 83
gdk-window-set-child-shapes 82
gdk-window-set-cursor 85
gdk-window-set-debug-updates 80
gdk-window-set-decorations 91
gdk-window-set-events 89
gdk-window-set-focus-on-map 81
gdk-window-set-functions 92
gdk-window-set-group 91
gdk-window-set-hints 84
gdk-window-set-icon 90
gdk-window-set-icon-list 86
gdk-window-set-icon-name 90
gdk-window-set-keep-above 73
gdk-window-set-keep-below 73
gdk-window-set-modal-hint 86
gdk-window-set-override-redirect 80
gdk-window-set-role 90
gdk-window-set-skip-pager-hint 87
gdk-window-set-skip-taskbar-hint 87
gdk-window-set-static-gravities 83

gdk-window-set-title 84
gdk-window-set-transient-for 90
gdk-window-set-type-hint 86
gdk-window-set-urgency-hint 87
gdk-window-shape-combine-mask 81
gdk-window-shape-combine-region 82
gdk-window-show 69
gdk-window-show-unraised 70
gdk-window-stick 71
gdk-window-thaw-updates 79
gdk-window-unfullscreen 72
gdk-window-unmaximize 72
gdk-window-unstick 71
gdk-window-withdraw 71
get-active-window 22
get-animation 145
get-axis 117
get-bits-per-sample 129
get-children 89
get-clip-region 33
get-colormap 31, 33, 49
get-colorspace 128
get-core-pointer 14
get-default-colormap 15
get-default-cursor-size 12
get-default-group 12
get-default-screen 8
get-depth 33
get-deskrelative-origin 88
get-direction 99
get-display 18, 32
get-event 10
get-events 89
get-font-options 21
get-format 144
get-from-drawable 52
get-from-image 53
get-geometry 85
get-group 91
get-has-alpha 128
get-height 18, 129, 142
get-height-mm 19
get-image 40
get-maximal-cursor-size 12
get-monitor-at-point 20
get-monitor-at-window 20
get-monitor-geometry 20
get-n-channels 128
get-n-monitors 19
get-n-screens 8
get-name 7
get-number 18
get-option 129
get-origin 88
get-parent 89
get-pixbuf 145
get-pixel 49
get-position 88
get-resolution 22
get-rgb-colormap 16
get-rgb-visual 16
get-rgba-colormap 17
get-rgba-visual 17
get-root-origin 88
get-root-window 17
get-rowstride 129
get-screen 8, 27, 32, 57
get-setting 21
get-size 33
get-state 70
get-system-colormap 16
get-system-visual 16
get-toplevel 89
get-toplevel-windows 19
get-type-hint 87
get-update-area 79
get-visible-region 34
get-visual 32
get-width 18, 129, 141
get-width-mm 18
get-window-at-pointer 11
get-window-stack 22
get-window-type 69

H

hide 70

I

iconify 71
input-shape-combine-mask 83
invalidate-rect 78
invalidate-region 78
is-composited 17
is-viewable 70
is-visible 70

K

keyboard-ungrab 8
keys-changed on <gdk-keymap> 99

L

list-devices 10
list-displays 14
list-visuals 19
lower 76

M

make-display-name 19
maximize 71
merge-child-input-shapes 83
merge-child-shapes 82

move . . . 73
move-region . . . 74
move-resize . . . 74

N

new-subpixbuf . . . 132

O

offset . . . 31

P

peek-event . . . 10
pointer-is-grabbed . . . 9
pointer-ungrab . . . 8
process-updates . . . 79
put-event . . . 10
put-pixel . . . 49

R

raise . . . 76
register-dnd . . . 76
render-threshold-alpha . . . 50
render-to-drawable . . . 50
render-to-drawable-alpha . . . 51
reparent . . . 75
resize . . . 73
rotate-simple . . . 138

S

saturate-and-pixelate . . . 140
scale . . . 135
scale-simple . . . 135
scroll . . . 74
set-accept-focus . . . 81
set-axis-use . . . 116
set-back-pixmap . . . 85
set-background . . . 28, 84
set-child-input-shapes . . . 83
set-child-shapes . . . 82
set-clip-mask . . . 29
set-clip-origin . . . 29
set-clip-rectangle . . . 30
set-clip-region . . . 30
set-colormap . . . 31, 33, 49
set-cursor . . . 85
set-decorations . . . 91
set-default-colormap . . . 16
set-double-click-time . . . 11
set-drawable . . . 120
set-events . . . 89
set-exposures . . . 30
set-fill . . . 29
set-focus-on-map . . . 81
set-font . . . 28
set-font-options . . . 21
set-foreground . . . 27
set-function . . . 28
set-functions . . . 92
set-gc . . . 121
set-group . . . 91
set-hints . . . 84
set-icon . . . 90
set-icon-list . . . 86
set-icon-name . . . 90
set-keep-above . . . 73
set-keep-below . . . 73
set-key . . . 116
set-line-attributes . . . 30
set-modal-hint . . . 86
set-mode . . . 116
set-override-redirect . . . 80
set-resolution . . . 22
set-rgb-bg-color . . . 28
set-rgb-fg-color . . . 28
set-role . . . 90
set-size . . . 145
set-skip-pager-hint . . . 87
set-skip-taskbar-hint . . . 87
set-source . . . 116
set-static-gravities . . . 83
set-stipple . . . 29, 121
set-subwindow . . . 30
set-tile . . . 29
set-title . . . 84
set-transient-for . . . 90
set-ts-origin . . . 29
set-type-hint . . . 86
set-urgency-hint . . . 87
shape-combine-mask . . . 81
shape-combine-region . . . 82
show . . . 69
show-unraised . . . 70
size-changed on <gdk-screen> . . . 15
size-prepared on <gdk-pixbuf-loader> . . . 144
stick . . . 71
supports-cursor-alpha . . . 12
supports-cursor-color . . . 11
supports-input-shapes . . . 13
supports-shapes . . . 12
sync . . . 9

T

thaw-updates . . . 79

U

unfullscreen . . . 72
unmaximize . . . 72
unstick . . . 71

W

warp-pointer 11
withdraw 71